UMA METODOLOGIA AUTOMATIZÁVEL PARA CONVERSÃO SÍNCRONO-ASSÍNCRONO DE CIRCUITOS DIGITAIS

Ricardo de Freire Cassia

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE DOUTOR EM CIÊNCIAS EM ENGENHARIA ELÉTRICA

Aprovada por:

_____
Prof. Felipe Maia Galvão França, Ph.D.

_____
Prof. Antônio Carneiro de Mesquita Filho, Dr. d'État

_____
Prof. Nadia Nedjah, Ph.D.

_____
Prof. Antonio Petraglia, Ph.D.

_____
Prof. Luiza de Macedo Mourelle, Ph.D.

RIO DE JANEIRO, RJ – BRASIL
ABRIL DE 2007

CASSIA, RICARDO DE FREIRE

Uma Metodologia Automatizável para Conversão Síncrono-Assíncrono de Circuitos Digitais [Rio de Janeiro] 2007

XI, 100 p. 29,7 cm (COPPE/UFRJ, D.Sc., Engenharia Elétrica, 2007)

Tese – Universidade Federal do Rio de Janeiro, COPPE

1. Conversão Síncrono-Assíncrono

I. COPPE/UFRJ II. Título (série)

Agradecimentos

Esse trabalho não teria sido concluído sem o apoio técnico e moral e sem o amor de pessoas ao meu redor, fisicamente e espiritualmente.

Toda a paciência, amparo e colaboração da esposa Fernanda, que fez atingível a realização dessa tarefa, ao contrário do que parecia.

Vladimir Castro Alves, mais que um amigo, pronto pra assistência, discussões, e diversão, assistência, discussões e diversão, assistência...

Realização de um sonho da mãe Maria Luiza, que merecia se tornar realidade.

Símbolo de garra e vontade da irmã Flavia de tocar a vida pra frente debaixo de chuva de canivete, sem guarda-chuvas e com vento contra.

Como ser pesquisador sem ter curiosidade de abrir, ver como funciona e, se possível remontar as coisas da vida, devidamente estimulada pelo pai Umberto e pelo avô José?

José Luiz, o exemplo de cientista, tio e padrinho, colocando *aquela* pressão sem se fazer notar.

A confiança e orientação alto astral e essencial do prof. Felipe França.

Vários nós que consegui dar desatados pela sempre calma Solange Coelho de Oliveira.

Dividir as angústias da produção de uma tese com Victor Mauro Goulart Ferreira.

Empurrões virtuais e pitacos de Federico Gálvez-Durand Besnard.

O exemplo de João Marcelo Alcântara, que a vida não é só trabalho e estudo, mas viver também.

A tese referência padrão de Edson do Prado Granja.

A todos aqueles que não foram nomeados aqui mas sabem que foram essenciais para a conclusão do trabalho. Vocês sabem quem são.

Esse trabalho teve apoio financeiro do Conselho Nacional de Desenvolvimento Científico e Tecnológico.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Doutor em Ciências (D.Sc.)

UMA METODOLOGIA AUTOMATIZÁVEL PARA CONVERSÃO SÍNCRONO-ASSÍNCRONO DE CIRCUITOS DIGITAIS

Ricardo de Freire Cassia

Abril/2007

Orientador: Felipe Maia Galvão França

Programa: Engenharia Elétrica

Nesse trabalho é apresentado um método automático de conversão de circuitos síncronos em circuitos assíncronos equivalentes. A técnica utiliza como entrada a descrição do circuito síncrono sintetizado e mapeado em portas lógicas, e emprega automaticamente a metodologia ASERT - *Asynchronous Scheduling by Edge Reversal Timing*, um algoritmo de temporização descentralizado, como método de sinalização entre as unidades funcionais assíncronas baseadas na divisão hierárquica dos blocos funcionais do circuito síncrono original. A metodologia foi implementada na forma de um programa dedicado, empregando ferramentas de *CAD* e bibliotecas de células básicas para circuitos síncronos tradicionais, e sua funcionalidade foi comprovada convertendo circuitos como cifradores AES e RSA, MDCT, filtros FIR dentre outros.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Doctor of Science (D.Sc.)

A METHODOLOGY FOR AUTOMATED SYNCHRONOUS-ASYNCHRONOUS CONVERSION OF DIGITAL CIRCUITS

Ricardo de Freire Cassia

April/2007

Advisor: Felipe Maia Galvão França

Department: Electrical Engineering

This work introduces a methodology to automatically convert synchronous circuits into equivalent asynchronous ones. The technique takes a synthesized and mapped gate level netlist as input, and applies ASERT— Asynchronous Scheduling by Edge Reversal Timing, a fully decentralized timing algorithm as a signaling method between the asynchronous functional blocks based on the existent hierarchical organization of the original synchronous circuit. The methodology was implemented as a program that uses CAD tools and standard cell libraries intended for synchronous circuits design, and its functionality was verified as it successfully converted systems like AES and RSA ciphers, MDCT, FIR filters among others.

Índice

Índice de Figuras

Índice de Tabelas

1 Introdução

Tecnologias atuais de projeto e implementação de circuitos integrados atingiram a ordem de dezenas de nanômetros, viabilizando extraordinárias densidades de transistores por área. Essa enorme capacidade de compactação permite que os mais diversos núcleos funcionais, tanto do tipo *soft* quanto do tipo *hard*, possivelmente desenvolvidos por fabricantes distintos, possam ser reutilizados e combinados, até certo ponto facilmente, com o objetivo de criarem sistemas completos em uma única pastilha de silício, conhecidos como sistemas em um único chip ou *SoC* (*System on a Chip*) [1, 2, 3, 4]. Esses circuitos são capazes de realizar funções extremamente complexas de forma mais eficiente em consumo e velocidade de operação, permitindo fazer uso da reutilização de blocos pré-projetados [5, 6, 7] para outros sistemas. O custo da aplicação é reduzido e seu uso é simplificado já que vários chips com funções mais simples conectados em uma placa de circuito impresso são substituídos por apenas um. *SoCs* contribuem para o aumento de freqüência de processamento pois o gargalo devido ao roteamento entre chips é reduzido.

A complexidade alcançada na metodologia da manufatura dos circuitos integrados foi acompanhada pelo desenvolvimento de ferramentas de *CAD*, bibliotecas de células, linguagens de descrição de *hardware* e técnicas de projetos de circuitos, os quais, na sua grande maioria, foram dominados pelos circuitos digitais síncronos desde a década de 1960. No entanto, o avanço da mesma tecnologia revelou algumas desvantagens dos sistemas síncronos, dentre elas um maior consumo de potência, dificuldades de interface com diferentes sistemas síncronos ou assíncronos, limitação na freqüência de operação por caminhos críticos e maior geração de ruído e emissão eletromagnética.

1.1 Motivação

Circuitos assíncronos possuem diversas propriedades que os fazem desejáveis em aplicações móveis e de segurança, e têm sido defendidos como um método para redução de consumo de potência em diversas situações [8, 9, 10, 11]. A ausência de um sistema central de sincronismo desobriga o projetista de distribuir um ou mais sinais de *clock* a todos os elementos seqüenciais e se preocupar com a defasagem entre eles. Esse sinal é substituído por um protocolo de sinalização local, geralmente mais simples que as

árvores de distribuição de *clock,* resultando numa considerável economia de potência, pois os métodos empregados na redução da defasagem consomem uma enorme quantidade de corrente [12]. Além disso, esses circuitos chaveiam (e consomem potência) tipicamente somente quando são requisitados por terem novos valores apresentados a suas entradas, caso contrário permanecem naturalmente em repouso. Essa característica elimina a necessidade de aplicação de mecanismos de *glock gating*.

Circuitos assíncronos ainda contribuem para a melhoria no campo de emissão eletromagnética [13], tanto na forma de ruído, que afetam circuitos analógicos ou digitais que estejam próximos, quanto na forma de radiação, cujo espectro de potência máximo é limitado por agências regulamentadoras. Esse benefício é resultado do fato dos registradores não transicionarem todos simultaneamente, o que reduz o ruído de potência. A ausência de um ponto de referência também dificulta o ataque em dispositivos de segurança [14], como um cifrador ou decifrador de chave pública, na tentativa de desvendar chaves secretas através da análise de dados em instantes de um período de *clock*.

A substituição de um sistema central de sincronismo por um método de sinalização assíncrona em núcleos independentes (*cores*) simplifica drasticamente sua integração em *SoCs* por não dependerem de um tratamento especial de interface entre diferentes domínios de *clock*. Dessa forma, cada núcleo opera com suas características temporais particulares e a troca de informações entre os mesmos ocorre naturalmente através do método de sinalização assíncrona empregado.

Em contrapartida, todas essas vantagens são acompanhadas pela dificuldade de implementação por não existirem ainda ferramentas de automação de projeto adequadas. Diversas metodologias [15 , 16] têm sido propostas para o auxílio do desenvolvimento de circuitos assíncronos, cada qual com suas vantagens e desvantagens.

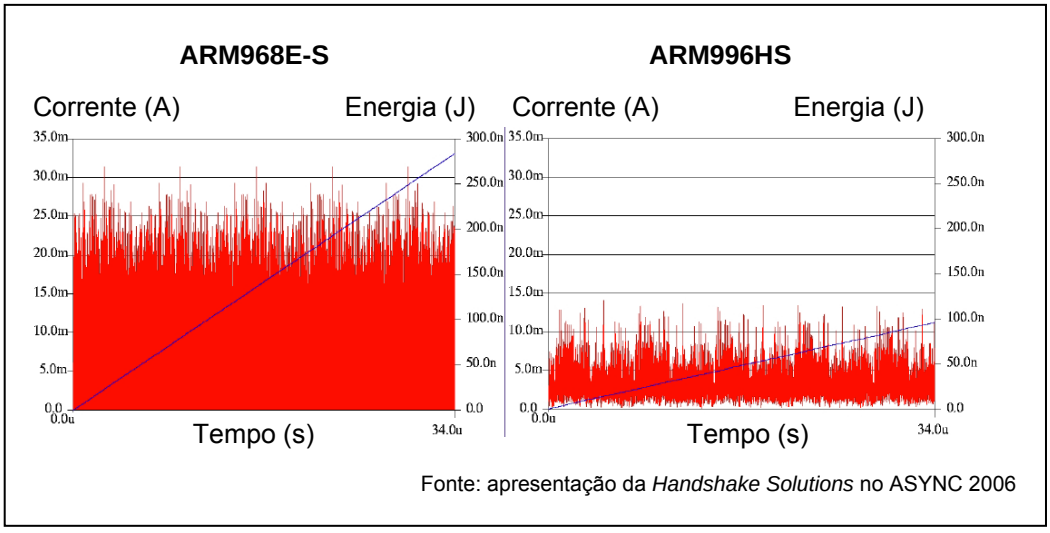
Um dos métodos de implementação de circuitos assíncronos é através do processo de conversão de circuitos originalmente projetados como síncronos. Essa abordagem permite o reaproveitamento de circuitos síncronos já existentes e dispensa experiência em projeto de circuitos assíncronos. O desenvolvimento segue um fluxo padrão de projeto de circuitos síncronos, desde a fase de especificações até a implementação da descrição em uma linguagem de *hardware*. As ferramentas de EDA, linguagens de descrição de *hardware* e bibliotecas de células empregadas são iguais às aplicadas em projetos síncronos, não sendo necessária nenhuma adaptação específica.

1.2 Estado da Arte

Circuitos síncronos têm sido empregados na grande maioria das aplicações lógicas digitais por ser mais fácil de serem projetados e testados do que circuitos assíncronos. No entanto, a evolução tecnológica do processo de manufatura de circuitos integrados revelou os limites dos circuitos síncronos em certas aplicações onde os circuitos assíncronos representam uma melhor alternativa.

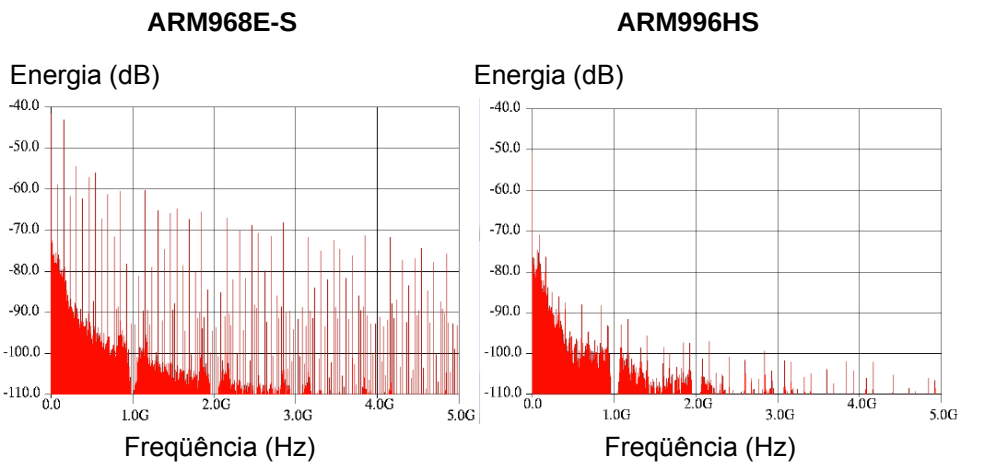
As técnicas de sinalização assíncrona serão indispensáveis para o projeto de *SoCs* pois as variações dos parâmetros através de toda a extensão dos circuitos integrados tornarão impossível o controle eficiente dos atrasos das redes de distribuição de um sinal único de sincronismo como o *clock* e outros sinais globais [17].

ARM968E-S, um processador de uso embutido empregado em inúmeras áreas de projetos digitais, já possui sua versão assíncrona comercialmente disponível. O ARM996HS, primeiro núcleo comercial de um processador de 32 bits sem *clock*, provou consumir aproximadamente 2,8 vezes menos corrente que seu equivalente síncrono, como apresentado na Figura 1. A emissão de ondas eletromagnéticas, representada na Figura 2 foi reduzida em 25dB, com pouco mais de 10% de aumento em área [44]. Nesse caso, o projeto foi desenvolvido utilizando linguagem e bibliotecas especialmente criadas para substituir as células padrões dependentes de *clock*, por equivalentes baseadas em protocolo de *handshake*.



Fonte: apresentação da *Handshake Solutions* no ASYNC 2006

Figura 1: Consumo de corrente e dissipação de energia em um processador RISC e sua contra parte assíncrona



Fonte: apresentação da *Handshake Solutions* no ASYNC 2006

Figura 2: Emissão de energia eletromagnética em um processador RISC e sua contra parte assíncrona

A evolução da indústria indica que o emprego de técnicas de projeto de circuitos assíncronos será essencial para acomodar projetos agressivos ou de materiais inovadores. A tecnologia de manufatura de componentes como LCDs ou circuitos integrados em superfícies flexíveis permite que a implementação desses componentes tenham não somente sua espessura reduzida, como também seu custo, possibilitando a utilização desse tipo de elemento em inúmeras aplicações [18]. Uma das maiores barreiras enfrentadas pelos projetistas nesse caso é a grande variação dos parâmetros em toda a superfície do elemento, devido basicamente à alteração da espessura de óxido e dos tamanhos dos grãos de cristal empregados. Até o momento, essas variações foram consideradas além da capacidade de controle a partir de uma lógica síncrona, especialmente para grandes circuitos como microprocessadores controlados por um *clock* global, no entanto são automaticamente absorvidas pelas características de circuitos assíncronos auto-temporizados.

Apesar da tecnologia de produção de semicondutores ter sido capaz de aumentar drasticamente as freqüências de funcionamento dos circuitos, conseguindo ainda baixar o custo da manufatura, a densidade de potência tem também sido ampliada ao ponto que a máxima capacidade disponibilizada pela tecnologia não poderá ser utilizada caso a metodologia de desenvolvimento de circuitos não seja alterada. Um *roadmap* alternativo [19] prevê que a potência dissipada será o fator mais importante a ser lidado,

tendo como passos obrigatórios para solução a implantação de modos de operação assíncrona [20, 21, 22, 23] e a diminuição de freqüência de operação, criando estruturas paralelas de processamento.

Circuitos síncronos aplicam técnicas de sincronização sem realimentação, incluindo a distribuição de *clock* e métodos de implementação de *clock gating*. A entidade que gera o sinal de sincronismo, seja um oscilador ou uma PLL (*Phase Locked Loop*), não recebe nenhuma informação qualitativa ou quantitativa retornada do que está sendo entregue às células seqüenciais e todas as incertezas dos atrasos envolvidos na árvore de *clock* ou em lógicas combinacionais são tratadas como as piores possíveis [24]. A necessidade de estimativa do resultado da fabricação do silício obriga o projeto do sistema com características muito acima das especificações, de forma a cobrir possíveis variações no processo de fabricação, o que nem sempre acontece da maneira prevista. Uma alternativa é usar circuitos que sejam capazes de aproveitar inerentemente ao máximo suas características pós-fabricação, o que acontece naturalmente em técnicas de projeto para circuitos assíncronos.

Diversas metodologias [15, 16] foram elaboradas especialmente para o desenvolvimento de sistemas assíncronos. Entretanto, não existem boas ferramentas ou metodologias capazes de auxiliar por completo o desenvolvimento de sistemas digitais assíncronos de alta complexidade. Micropipelines [60], por exemplo, podem ser utilizados somente em estruturas de dados que não necessitem de realimentação, inviabilizando sua aplicação direta em diversos modelos de sistemas digitais, como DSPs ou até simples máquinas de estados. Soluções para situações particulares foram apresentadas [25, 26], todavia uma solução genérica ainda não foi elaborada. O CHP (*Communicating Hardware Processes*) foi um dos precursores de linguagens dedicadas a descrever *hardware* permitindo associação de sinalizações assíncronas a seus blocos básicos, o que possibilitou uma forma de alto nível de elaboração de sistemas assíncronos [27], tendo ainda alguns conversores de CHP para VHDL desenvolvidos por outras propostas [28]. Tangram [29] é uma linguagem que permite a descrição de circuitos implementados com sinalização por *handshake*, também utilizados no desevolvimento de circuitos assíncronos. Apesar de atingirem alguns dos benefícios associados aos assíncronos, esses circuitos, de acordo com [30], não são necessariamente otimizados em relação à dissipação de potência ou freqüência de operação, além de exigirem o desenvolvimento em linguagens de descrição não convencionais, como a CHP. Ferramentas de síntese baseadas em grafos de transições

de sinais [31] ou especificações de modo *burst* [32, 33] são capazes de trabalhar com pequenos circuitos de controle, sendo inapropriados para sistemas digitais maiores ou de processamento de sinais.

Em [34], é apresentada uma metodologia para implementação de *SoCs* GALS (Globalmente Assíncrono, Localmente Síncrono) baseados em descrições VHDL e Verilog dos sistemas, onde as "ilhas" síncronas (blocos localmente síncronos) são empacotadas em interfaces assíncronas que interagem entre si através de um protocolo de *request* e *acknowledge*. Essa metodologia utiliza geração local de sinais de *clock* e somente a intercomunicação entre ilhas é realizada de forma assíncrona. Todas as especificações das definições de blocos síncronos devem ser manualmente discernidas através de primitivas específicas em uma linguagem própria, que são analisados por *scripts* antes de serem convertidos em uma descrição puramente compatível com a linguagem Verilog.

A metodologia descrita em [35] apresenta o mecanismo de de-sincronização de um circuito síncrono através da substituição de cada um dos registradores do circuito por um par de *latches*. O sinal de *clock* global e sua respectiva árvore de distribuição são substituídos por uma sinalização de *handshake* de quatro fases que controlam o estado de cada *latch* inserido no sistema, tornando desnecessário o emprego de um sinal único de sincronismo. No entanto foram detectadas 3 questões nessa metodologia: o *overhead* acrescido pelos controladores, pois cada registrador possui um controlador próprio, a impossibilidade de converter máquinas de estados síncronas em assíncronas por pura implementação do algoritmo sugerido, além de ser uma metodologia invasiva por alterar o *netlist* devido à substituição de registradores por *latches*. Máquinas de estados síncronas, como o próprio nome sugere, têm seu funcionamento baseado em estados mantidos por um conjunto de registradores que mantém as informações temporariamente ciclo após ciclo, tendo seus valores alterados pelo seu estado atual somado a possíveis entradas externas. Ao substituir um sistema de sincronismo central por um de sinalização assíncrona, o conceito de estado se perde, pois os registradores passam a ser submetidos a mudanças seriais, causando o funcionamento incorreto da mesma.

Um processo automatizado de conversão de circuitos síncronos em circuitos assíncronos é proposto em [30]. Nessa metodologia, bastante similar à descrita acima, todas as células registradoras (flip-flops) do circuito são substituídas por DLAPs (*Doubly Latched Asynchronous Pipeline*) [36], e um protocolo de comunicação de

quatro fases é construído entre os novos elementos interdependentes, para que o fluxo de dados seja propagado assincronamente. Como essa metodologia é praticamente análoga ao método descrito em [35], as questões apontadas como problemáticas estão também presentes nesse caso.

1.3 Objetivo

A metodologia para conversão de circuitos síncronos em circuitos assíncronos equivalentes proposta nesse trabalho possibilita uma simples forma de implementação de circuitos assíncronos utilizando ferramentas e bibliotecas de células básicas disponíveis no mercado, essencialmente aplicando o método ASERT [37, 38] de sinalização assíncrona de maneira automática. Serão realizadas conversões de diversos tipos de circuitos síncronos, cujas variações em arquitetura, dimensão e complexidade testarão a flexibilidade do algoritmo em tratar de estruturas como máquinas de estados, *pipelines*, realimentações, tabelas indexadas estáticas e dinâmicas, dentre outras.

Circuitos podem ser projetados, depurados, convertidos e depositados em uma biblioteca para aplicações futuras. Dessa forma, a implementação de um sistema tem seu tempo de desenvolvimento reduzido, facilitando também o casamento entre blocos com diferentes domínios de freqüência.

O método empregado para particionamento de sub-blocos capazes de serem transformados em unidades assíncronas é extraído da divisão hierárquica do sistema original. Caso o projetista tenha o conhecimento *a priori* de usar a metodologia de conversão proposta, a estrutura de divisão modular e hierárquica do circuito síncrono em desenvolvimento pode ser orientada a contribuir para uma conversão mais bem otimizada, através do modo como o agrupamento funcional em módulos do sistema é realizado.

O circuito pode ser inteiramente projetado, implementado, testado e aperfeiçoado no domínio síncrono antes de ser processado pelo método de conversão, além de permitir o uso de todo o suporte provido ao desenvolvimento de circuitos síncronos, como sintetizadores, simuladores e depuradores.

A implementação da metodologia será realizada por programas escritos nas linguagens *bourne shell*, responsável por controlar a execução de todos os passos do método; *tcl*, forma de entrada de comandos e controle das ferramentas de CAD, e *awk*,

responsável por criar, instanciar e interconectar os devidos controladores de nós e arestas necessários para a conversão.

1.4 Estrutura do Texto

O Capítulo 2 relembra conceitos básicos úteis para o entendimento da metodologia proposta. A Seção 2.1 menciona os fundamentos de máquinas síncronas e assíncronas, enquanto a Seção 2.2 explica o projeto de sistemas assíncronos. Os elementos utilizados na síntese de circuitos síncronos e assíncronos estão apresentados na Seção 2.3, e as linguagens de descrição de *hardware* utilizadas tanto como entrada do processo de conversão quanto no código final processado estão descritas na Seção 2.4. A Seção 2.5 introduz o conceito de cones de dependências, utilizados no levantamento de interconexões entre unidades funcionais e a Seção 2.6 lista as ferramentas de CAD empregadas na implementação do processo. As Seções 2.7 e 2.8 elaboram o método de sinalização assíncrona ASERT aplicado no circuito final convertido.

O processo de conversão é introduzido no Capítulo 3. As Seções 3.1, 3.2 e 3.3 mostram as correlações existentes entre elementos de uma descrição de *hardware* e os recursos de sinalização assíncrona empregados, e como as associações mútuas são estabelecidas. Uma comparação entre 2 métodos de geração dos sinais de fim de operação de blocos individuais são apresentados na Seção 3.4.

A Seção 4.1 expõe como é realizada a implementação da metodologia proposta e a Seção 4.2 reporta os resultados conseguidos em diferentes casos.

Finalmente, as conclusões tiradas do trabalho estão apresentadas no Capítulo 5.

2 Conceitos Básicos

Neste capítulo estão apresentadas algumas informações básicas sobre tópicos mencionados do decorrer deste trabalho.

2.1 Máquinas Síncronas e Assíncronas

2.1.1 Fundamentos

Máquinas síncronas podem ser descritas por sinais binários, permitindo a utilização de lógica Booleana, em uma base de tempo discreto, onde estados transientes de sinais não precisam ser levados em consideração [39].

A Figura 3 descreve um sistema síncrono. Um conjunto de registradores armazena o estado atual da máquina. A mudança de estado acontece após a aplicação de um pulso de *clock*, sendo o valor do próximo estado calculado por circuitos lógicos combinacionais que atuam sobre um vetor de entradas e sobre um vetor que define o estado atual.

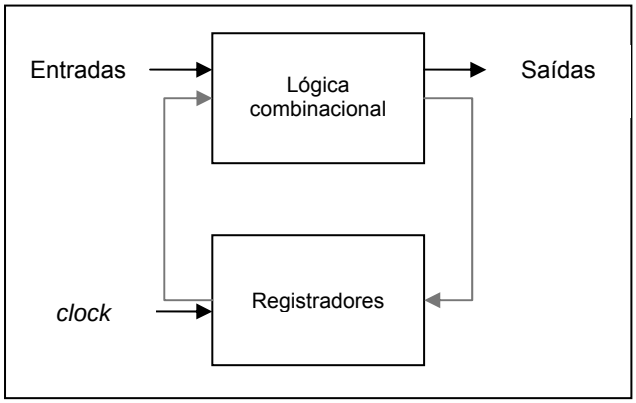


Figura 3: Máquina síncrona genérica

Já nas máquinas assíncronas, a mudança de estado ocorre de forma independente em cada registrador, estando sujeita somente a restrições de caráter local, como por exemplo, a ocupação dos recursos vizinhos. Uma descrição semelhante a da Figura 3 pode ser feita substituindo o sinal de *clock* por outros de sincronismo, gerados juntamente com os dados.

2.1.2 Consumo de Corrente

Registradores dissipam potência a cada transição do *clock*, independentemente de haver ou não mudança de estado em circuitos síncronos. Similarmente, no caso de circuitos combinacionais com lógica dinâmica, haverá dissipação de potência a cada pulso de *clock*. Ao se considerar circuitos integrados do tipo CMOS, a corrente quiescente consumida pelo sistema é muito pequena, e pode ser desconsiderada na medição do consumo, de modo que a dissipação de potência aumenta de forma linear com a freqüência de operação. Para reduzir esse efeito, os fabricantes fazem uso de tensões de alimentação cada vez mais baixas nos circuitos integrados. Por sua vez, os circuitos assíncronos apresentam consumo de corrente apenas quando e onde houver atividade.

As máquinas síncronas podem apresentar economias de potência ao se desligar módulos, acarretando perda temporária de funcionalidade. Tais máquinas podem também fazer uso de circuitos especiais para diminuir a freqüência de *clock* ou mesmo para chaveamento seletivo de parte dos circuitos. Os circuitos assíncronos têm a vantagem de obter tais resultados com granularidade mais fina, sem perda de funcionalidade e sem necessidade de uso de circuitos especiais.

O sinal de *clock* global em um circuito síncrono é responsável por grande parte do consumo de corrente pelas seguintes razões:

- árvores de distribuição de *clock* requerem uma rede complexa de *buffers* para atingir características aceitáveis da qualidade do sinal nos elementos seqüenciais, como baixo desvio de fase e tempo de transição limitado;
- o sinal de *clock* é o que chaveia mais freqüentemente;
- praticamente todos os nós de um circuito síncrono chaveiam devido a uma transição do sinal de *clock*.

Em circuitos assíncronos, o sinal global de *clock* é substituído por múltiplos sinais locais de *handshake*. Como cada ligação local tende a ser mais curta e sua respectiva capacitância é menor que a de um sinal global, a potência total dissipada pelo controle temporal no caso assíncrono pode ser menor do que a do circuito síncrono equivalente. Por exemplo, o processador assíncrono Amulet3 [40] é funcionalmente compatível com o processador síncrono ARM9TDMI, disponibilizado pela ARM, Inc. Ambos atingem aproximadamente a mesma performance (120 MIPS) e eficiência (780 MIPS/Watt) [41].

Os circuitos síncronos têm a desvantagem de serem ruidosos em razão da corrente consumida no entorno da transição dos pulsos de *clock*. No interior de um circuito integrado, onde as impedâncias são mais elevadas, há uma significativa modulação nos barramentos de alimentação, que gera interferência nas vias de sinais. O uso de tensões de alimentação mais baixas também contribui para aumentar a vulnerabilidade das máquinas aos diversos tipos de ruído. Os sinais de *clock* propiciam uma considerável irradiação de energia eletromagnética na freqüência do *clock* e seus harmônicos. Dada esta irradiação, deve-se sempre considerar a blindagem de circuitos digitais na presença de circuitos analógicos de baixo nível ao redor dos circuitos síncronos [42].

As máquinas assíncronas apresentam irradiação eletromagnética e geração de pulsos de ruído nas vias de alimentação em uma proporção muito menor [43]. No entanto, há um maior consumo de potência por função lógica das versões assíncronas, uma vez que elas exigem um circuito extra para prover o sincronismo, dada a ausência de um *clock* global. Esta ausência também contribui para a disseminação da interferência das operações de chaveamento, como visto em um exemplo de comparação entre o espectro dos pulsos de corrente de um *ARM996HS* assíncrono, da *Handshake Solutions*, e a versão original, o *ARM968E-S* da ARM, mostrando uma diminuição considerável no espectro mensurável da corrente de alimentação [44] na versão assíncrona.

2.2 Projeto de Sistemas Assíncronos

2.2.1 Transmissão de Dados por Sinalização

As máquinas assíncronas se comunicam através de protocolos. Um *request* de comunicação é feito por um circuito, que aguarda até receber um *acknowledge* do circuito destino. Para tanto, são usados os protocolos conhecidos como *four-phase RZ* apresentado na Figura 4 e *two-phase NRZ* apresentado na Figura 5 [45].

O protocolo de 2 fases é mais eficiente em relação à dissipação de potência, pois todas as transições são usadas para transferência de dados, mas requer uma maior quantidade de lógica auxiliar. Os protocolos de 2 fases e de 4 fases podem ser usados em diferentes módulos de um mesmo circuito, porém em interfaces distintas, e apresentam um possível método de conversão [45].

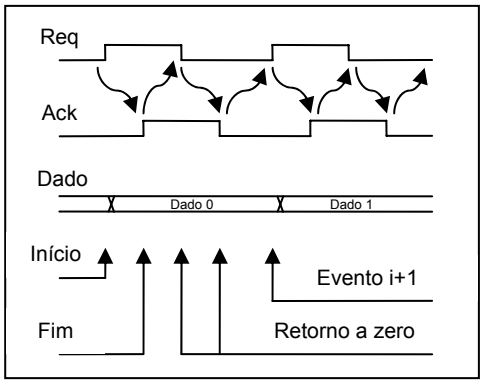


Figura 4: Protocolo de 4 fases

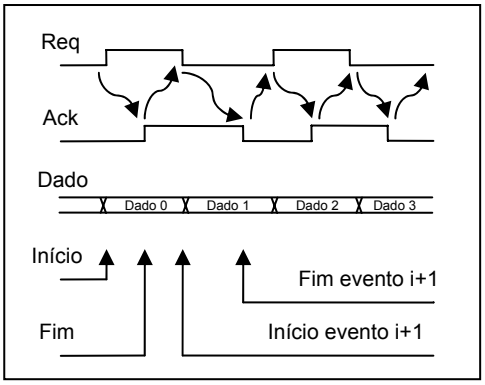


Figura 5: Protocolo de 2 fases

Para efetuar uma transferência de dados, os circuitos assíncronos precisam de uma validação do sinal transmitido. Os métodos usuais de temporização de dados incluem *bundled data* e codificação *dual rail* [45].

O método mais simples, sem redundâncias, é a codificação de *single rail*, mostrado na Figura 6, no qual um único sinal é usado para indicar que os dados de saída são válidos. Seu uso se dá principalmente para avaliar o retardo de uma mesma função lógica quando aplicada a um vetor de dados. Para tanto, faz-se uma avaliação do pior caso reservando uma margem para variações de parâmetros.

O método de codificação *dual rail* utiliza uma máquina seqüencial para codificar em dois bits o valor e o estado (válido ou não) de cada bit de dados, como apresentado na Figura 7:

- o valor 00 indica operação em andamento;
- o valor 10 indica 0;
- o valor 01 indica 1;
- o valor 11 é não válido.

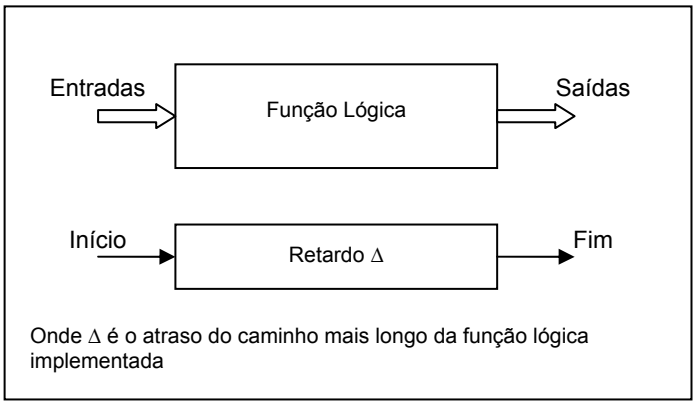


Figura 6: Codificação *single rail*

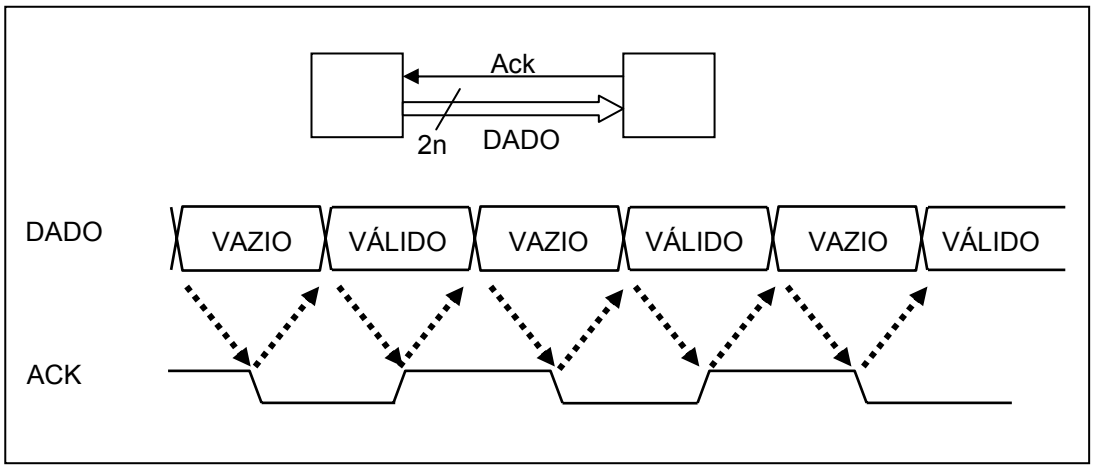


Figura 7: Codificação *dual rail*

2.2.2 Fim de Operação

A geração do sinal do fim de operação consiste em construir uma estrutura paralela com um atraso igual ou maior que o caminho crítico do bloco funcional a ser sinalizado. A estrutura paralela é responsável por atrasar um sinal aplicado à sua entrada, o de início de operação, de forma que o sinal de fim de operação seja ativado adequadamente quando a saída do circuito principal esteja estável. Na prática, como mencionado, ferramentas de CAD auxiliam na tarefa de identificar o tempo de propagação do caminho crítico. A estrutura de atraso é constituída por uma cadeia de células de retardo (*delay cell*) conectadas serialmente, cujo número de elementos é determinado pela relação do tempo de propagação intrínseco às células, baseado nas referências que descrevem a tecnologia utilizada, e o caminho crítico do circuito alvo.

Valores complementares de funções lógicas podem ser utilizados juntamente com uma porta *XOR* para detectar o fim de operação, como na implementação de circuitos DCVSL – *Differential Cascode Voltage Swing Logic*, mostrada na Figura 8. Neste caso, o sinal Req faz a pré-carga das saídas da função quando inativo. A função é calculada a partir do instante em que Req assume o valor lógico '1' e o sinal de fim de operação é gerado dos valores opostos resultante da avaliação complementar da função *F* [46].

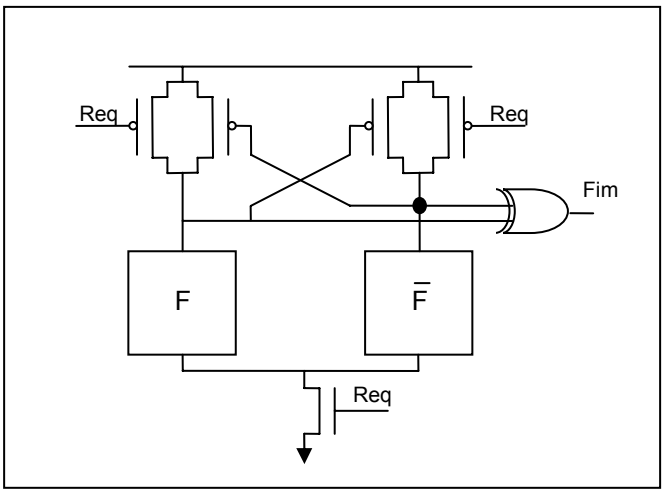


Figura 8: Lógica DCVSL

2.2.3 Modelos de Temporização de Circuitos Assíncronos

Circuitos assíncronos são agrupados em classes, dependendo do tipo de atrasos determinados pela sua análise, que podem ser:

- Atrasos fixos (*fixed delay*)
- Atrasos limitados (*bounded delay*)
- Atrasos ilimitados (*unbounded delay*)

O modo mais comum de usar circuitos assíncronos é o mesmo empregado em circuitos síncronos, chamado de *bounded delay*. Mais especificamente, é assumido que todos os atrasos relativos às portas e aos fios de um circuito são conhecidos, ou pelo menos limitados.

Circuitos insensíveis a atrasos (*delay-insensitive*) usam um modelo completamente oposto ao de atrasos limitados: os atrasos, nos componentes e fios, são

ilimitados. Nos circuitos com atrasos limitados, é assumido que todos os subcircuitos, providos de tempo suficiente, terão todos os seus nós estáveis e prontos para receberem uma nova entrada. Já em circuitos insensíveis a atrasos, independentemente do tempo disponibilizado para o circuito, não existe garantia que uma nova entrada será corretamente recebida. Essa situação força o bloco receptor a comunicar ao gerador de dados que está preparado para receber um novo valor. Dessa forma, o bloco gerador de dados deve sempre esperar que a sinalização de recebimento seja comunicada para enviar o próximo item [47].

Ao admitir uma aproximação em que tempos de propagação em caminhos que sejam bifurcados tenham a mesma duração, um novo modelo pode ser definido, chamado de circuitos quase insensíveis a atrasos (*quasi-delay-insensitive*). Nesse modelo, a comunicação de recebimento de dados pode ser realizada por somente um dos caminhos bifurcados (*forks*), o que simplifica tanto o tratamento de fluxo do bloco gerador, como diminui a quantidade de lógica dedicada nos blocos receptores [48].

Ao se considerar que o tempo de propagação nas interconexões do circuito pode ser negligenciado, mas, no entanto, que o tempo de propagação nas portas lógicas tem atraso ilimitado, define-se os circuitos como imunes à velocidade dos componentes (*speed-independent*) [49].

Por fim, um circuito composto por elementos com capacidade para determinar seu próprio tempo de operação é denominado auto-temporizado (*self-timed*). Tal circuito deve estar totalmente compreendido em uma única região equipotencial de tempo. Já a comunicação entre regiões não equipotenciais deve ser realizada como insensíveis a atrasos [47].

2.2.4 Máquinas de Estado

As máquinas de estados assíncronas são projetadas aplicando o modelo de atrasos limitados, o que torna seu projeto bastante similar ao projeto de máquinas de estados síncronas. O circuito a ser criado é geralmente expresso em termos de uma tabela de fluxo, uma forma similar à tabela verdade [50]. Como mostrado na Figura 9, a tabela de fluxo possui uma linha para cada estado interno e uma coluna para cada combinação de entradas. Os itens em cada posição da tabela indicam o próximo estado a ser tomado e as saídas geradas quando a combinação de entradas aparece na linha de estados. Os estados circundados correspondem a estados estáveis, cujo próximo estado é idêntico ao

atual. Geralmente é assumido que cada estado instável é levado diretamente a um estado estável, com no máximo uma transição ocorrendo em cada variável da saída, conhecido como o modo fundamental de Huffman [45]. Assim como a síntese de máquinas de estados em sistemas síncronos, a redução de estados e sua codificação são realizadas na tabela de fluxo, e mapas de Karnaugh são gerados para cada sinal resultante.

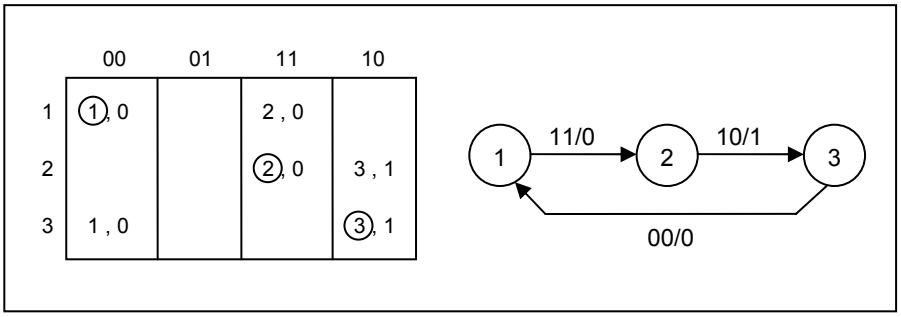


Figura 9: Exemplo de tabela de fluxo (esquerda) e sua máquina de estados correspondente (direita)

Cuidados especiais devem ser tomados ao implementar uma máquina de estados assíncrona ao contrário das versões síncronas. Como não há um sinal de *clock* que sincronize a chegada de entradas novas, o sistema deve se comportar devidamente em estados intermediários causados por variações múltiplas em suas entradas. Na tabela de fluxo da Figura 9, o sistema não transiciona diretamente da entrada "00" para "11", mas fica, na verdade, temporariamente em "01" ou "10". Portanto para o estado 1, as entradas "01" e "10" devem ser estabelecidas para que a máquina fique estável no estado 1.

Outro possível problema presente em máquinas de estados assíncronas é a corrida de sinais, ou *hazards*. O mapa de Karnaugh apresentado na Figura 10 está representado como soma de produtos e todas as portas lógicas têm um atraso intrínseco de '1', e o estado atual é (A, B, C) = (1, 1, 1). Nesse estado, AB é verdadeiro e a saída é igual a '1'. Se B se torna 0, o estado é alterado para (1, 0, 1) e a saída deve continuar '1'. No entanto, devido ao atraso da porta inversora, a porta *AND* superior se tornará falsa antes da inferior se tornar verdadeira, e um '0' será propagado até a saída. Esse *glitch* temporário na saída é conhecido como um *hazard* '1'-estático, e deve ser removido para o funcionamento correto do circuito. Um *hazard* '0'-estático tem o mesmo comportamento, mas com um valor estável em '0' que ao contrário fica '1' momentaneamente. Um *hazard* dinâmico é o caso onde o sinal que deveria ter uma transição única ($0 \rightarrow 1$ ou $1 \rightarrow 0$) tem 3 ou mais transições ($0 \rightarrow 1 \rightarrow 0 \rightarrow 1$).

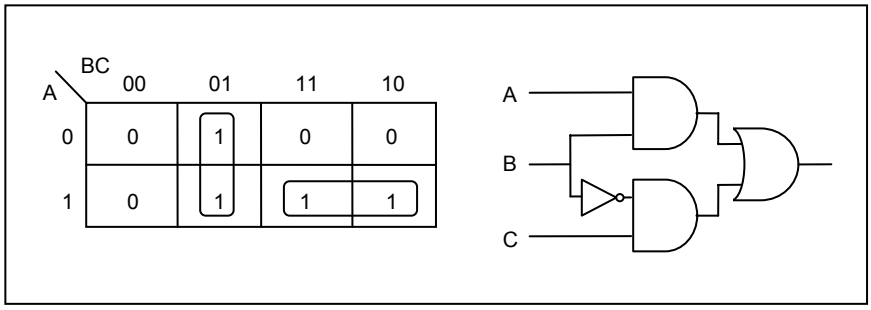


Figura 10: Mapa de Karnaugh (esquerda) e implementação de um circuito com hazards (direita)

Ao se considerar um modelo de atraso limitado, é possível eliminar o *hazard* através da introdução de célula de retardo nos caminhos críticos. Existem métodos de síntese capazes de criar circuitos sem *hazards* [50, 51].

Tendo em vista os circuitos combinacionais onde apenas uma variável se modifica a cada vez é possível impedir *hazards* em sistemas *AND-OR*, ao introduzir mintermos redundantes no mapa de Karnaugh [52].

Nos circuitos com alterações simultâneas de múltiplas entradas, a própria definição do valor da função de saída pode não ser monotônica, ou seja, pode haver mais de uma transição no mapa de Karnaugh até que a mesma se estabilize em um valor final [51].

Máquinas *self-synchronized* são similares às máquinas de Huffman, mas utilizam registradores para armazenar os estados, com o *clock* dos registradores sendo gerado a partir de variações das entradas. Máquinas auto-sincronizadas, representadas na Figura 11, operam em modo *burst*, o que permite a mudança simultânea de múltiplas entradas. Tais máquinas aguardam a chegada de uma coleção de entradas, e apenas então reagem com uma coleção de saídas, por isso são livres de *hazard*, e apresentam um alto desempenho [53].

2.2.5 Síntese por Rede de Petri

Rede de Petri [54] é uma das várias representações matemáticas para sistemas distribuídos discretos. Como uma linguagem de modelagem, ela define graficamente a estrutura de um sistema distribuído como um grafo direcionado com comentários. Possui nós de posição, nós de transição e arcos direcionados conectando posições com transições.

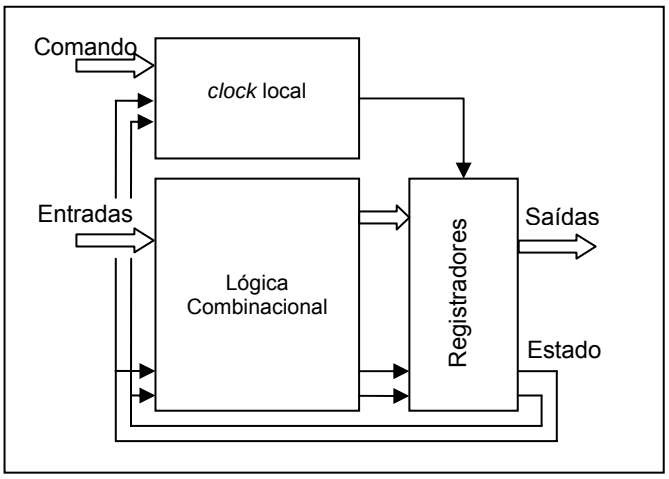


Figura 11: Máquina de estados auto-sincronizada

A qualquer momento durante a execução de uma rede de Petri, cada posição pode armazenar uma ou mais fichas (*tokens*). Diferente de sistemas mais tradicionais de processamento de dados, que podem processar somente um único fluxo de fichas entrantes, as transições de redes de Petri podem consumir fichas de múltiplas fontes e dar fichas a múltiplos destinos, mas uma transição só pode agir nas fichas se o número requisitado das mesmas aparecer em cada posição de entrada.

A síntese de circuitos assíncronos pode ser baseada em rede de Petri, por representar um funcionamento seqüencial de eventos parcialmente ordenados dependentes de estados de cada um dos nós de posição, como exemplificado na Figura 12.

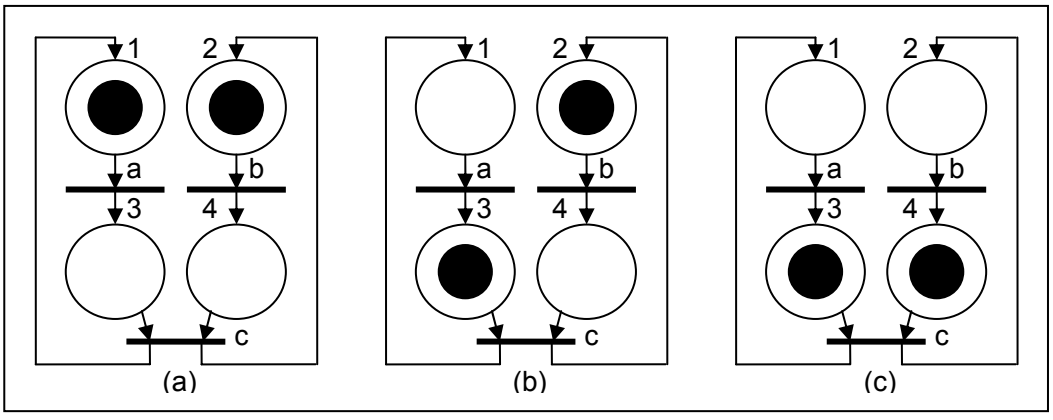


Figura 12: Exemplo de uma rede de Petri

Para que as transições a e b sejam disparadas, são necessárias as respectivas presenças de fichas em *1* e *2*, enquanto a transição *c* depende da presença de fichas tanto em *3* como em *4*. A cada transição concluída, a ficha é enviada à posição seguinte conectada através de seu arco.

Um circuito assíncrono pode ser implementado em *hardware* a partir do mapeamento direto da respectiva rede de Petri ou usá-la apenas como especificação comportamental para análise [55]. Uma variante restrita chamada Grafos de Transição de Sinais (*Signal Transition Graphs*) tem sido utilizada por permitir a síntese de sistemas do tipo independente da velocidade (*speed-independent*) [56]. Nesse caso existem algoritmos otimizados para particionamento [57], minimização e atribuição de estados [58].

2.3 Elementos de Síntese

2.3.1 Sincronizadores e Arbitradores

A porta C-Müller [61] é um elemento sincronizador tradicionalmente usado em circuitos assíncronos. De acordo com seu funcionamento, se houver igualdade de valores das entradas, então a saída acompanha esse valor. Na ausência deste fato, a saída mantém o estado anterior. A Figura 13 demonstra uma possível implementação para a porta C-Müller.

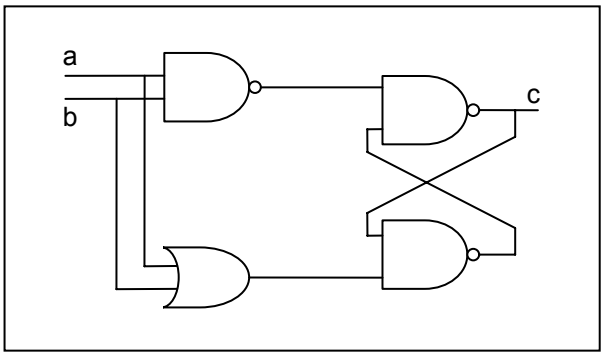


Figura 13: Implementação da porta C-Müller

A Figura 14 apresenta uma possível implementação de um circuito arbitrador, cuja função é a de discriminar a ordem de chegada dos sinais de entrada. Em um dado

momento, somente uma única saída pode estar ativa, e um pedido pendente somente é atendido em seguida ao término da atividade em andamento.

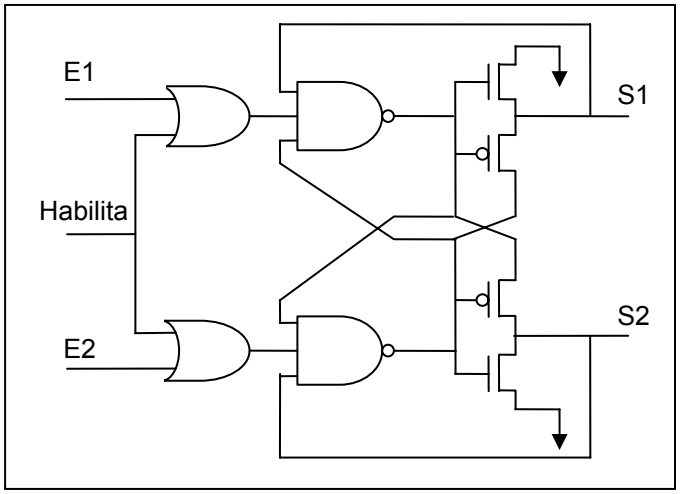


Figura 14: Implementação de um arbitrador com habilitação

2.3.2 Pipelines

Pipeline é um recurso desenvolvido para solucionar as questões relativas a longos caminhos de dados responsáveis por reduzir a freqüência máxima de operação de um sistema. Seu princípio é baseado na subdivisão funcional de caminhos combinacionais em estágios mais curtos, possibilitando a diminuição do período de acionamento dos elementos seqüenciais, aumentando o volume de dados processados. A seguir serão apresentados 3 diferentes tipos de *pipeline*.

2.3.2.1 Pipeline Síncrono

Em geral usa-se o *pipeline* síncrono clássico, representado na Figura 15, como base de comparação com o desempenho de *pipelines* assíncronos. Seguindo o princípio de atraso limitado, no pipeline síncrono deve-se calcular todos os atrasos introduzidos pela lógica combinacional e também os tempos de *setup* e *hold* dos elementos de memorização. O período do *clock* submete-se ao pior caso de todos os estágios, sem deixar de considerar uma margem para garantir o espalhamento de parâmetros na fabricação e o pior caso de variação de ambiente.

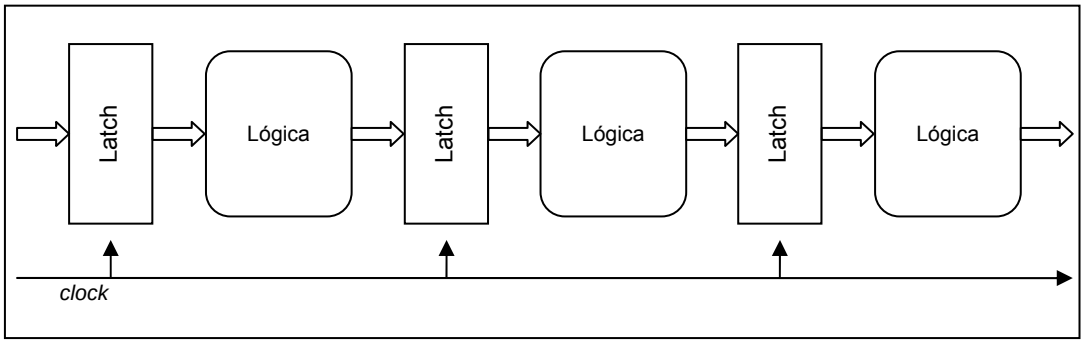


Figura 15: Pipeline síncrono

2.3.2.2 Pipeline Assíncrono Auto-temporizado

Os *pipelines* assíncronos não necessitam que todos os estágios contenham dados para o seu funcionamento normal. O princípio desses *pipelines* se assemelha ao de uma *FIFO*, ou seja, os dados migram da entrada para a saída sempre que houver um estágio disponível. Se a operação prevista não precisar usar todos os estágios do *pipeline*, pode ser descartada no interior do *pipeline* assim que as operações necessárias sejam completadas [59].

A Figura 16 demonstra um pipeline assíncrono que faz uso de protocolo *delay-insensitive* para habilitar a progressão de dados entre estágios. O tempo decorrido de cada operação é definido localmente.

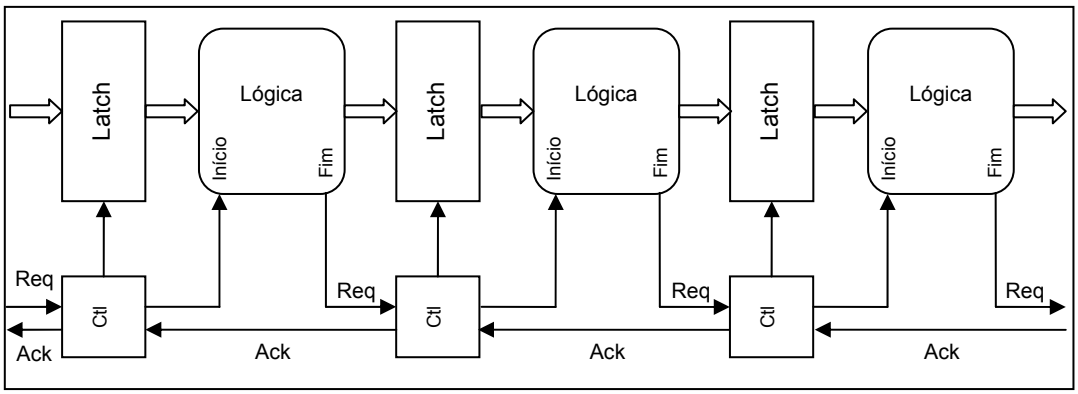


Figura 16: *Pipeline* assíncrono auto-temporizada

2.3.2.3 Micropipelines

A maior parte dos projetos atuais de *pipelines* assíncronos deriva do método de Micropipelines, cujo diagrama de blocos encontra-se na Figura 17 [60]. Micropipelines

combinam uma interface entre estágios do tipo *2-phase delay insensitive* com sinalização feita por *bundled data* [61]. O projeto e síntese da lógica de cada estágio do *pipeline* podem ser assistidos pelas ferramentas disponíveis para circuitos síncronos. O tempo de propagação de cada estágio, embora calculado para o pior caso, é usado para gerar a cadeia de células de retardo que determina o fim de operação [62].

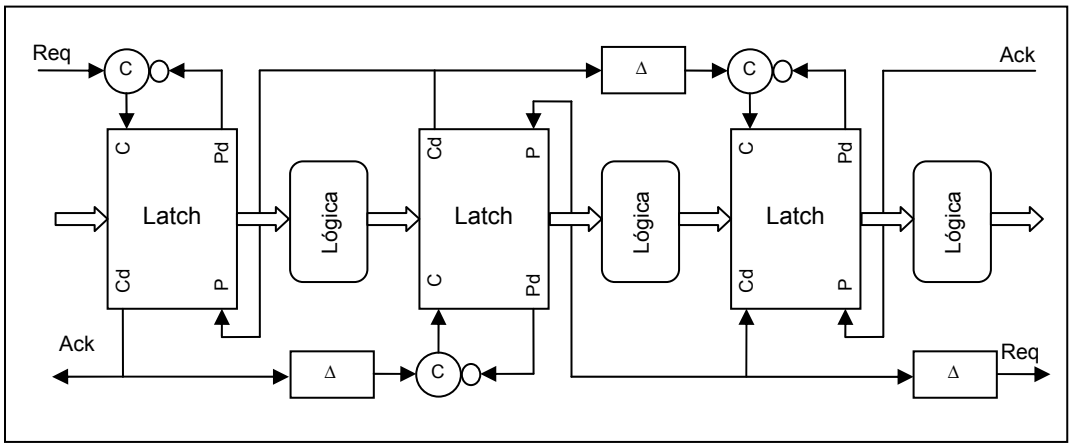


Figura 17: Estrutura de micropipeline

A captura de dados acontece quando um *Req* (*request*) é recebido, enquanto o *Ack* (*acknowledge*) é retornado através do sinal *Cd* (*capture done*). Ele é encaminhado ao estágio anterior, onde habilita o modo transparente do *latch* (*P*) e posteriormente habilitará uma nova captura. Após o devido retardo, esse mesmo sinal fará um *request* para o próximo estágio.

2.4 Linguagens de Descrição de Hardware (HDL)

Devido ao avanço da tecnologia de manufatura de circuitos integrados de alta densidade de transistores por unidade de área, capazes de acomodar dezenas de milhões de portas lógicas em uma única pastilha de silício, foram desenvolvidas linguagens de alto nível apropriadas para descrever sistemas digitais, facilitando o seu projeto, validação e reutilização em sistemas futuros. As *HDLs*, do inglês *Hardware Description Language*, permitem descrever textualmente tanto a funcionalidade como a temporização de circuitos digitais, possibilitando que elevados níveis de complexidade de projeto sejam alcançados sem que o projetista tenha que lidar diretamente com milhões de portas lógicas a serem interconectadas. Duas das mais importantes linguagens de descrição de *hardware* são apresentadas nas próximas secções.

2.4.1 VHDL

VHDL, que significa Linguagem de Descrição de *Hardware* de Circuitos Integrados de Alta Velocidade, do inglês *VHSIC Hardware Description Language*, é uma linguagem desenvolvida e otimizada para descrever o comportamento de circuitos e sistemas digitais eletrônicos provinda de um projeto de pesquisa do Departamento de Defesa dos EUA no início da década de 80. Já na época, circuitos digitais atingiram níveis de complexidade demasiadamente altos e projetá-los tornou-se um processo doloroso. VHDL foi desenvolvida com o objetivo de suprir as necessidades durante o processo de projetos de circuitos digitais que, descritos nessa linguagem, podem ser facilmente simulados, possivelmente sintetizados em diversas tecnologias alvo, além de poderem ser guardados e reutilizados posteriormente.

VHDL possui diversos mecanismos para descrição do comportamento de componentes eletrônicos, desde simples portas lógicas até microprocessadores completos ou circuitos integrados personalizados. A linguagem permite que características do circuito sejam precisamente definidas, como tempo de transição de subida e descida dos sinais, atrasos intrínsecos das portas lógicas e é claro sua função lógica.

Existem duas estruturas básicas na descrição de um circuito em *VHDL*: **entidade** (*entity*) e **arquitetura** (*architecture*). Comparando com o método de projeto esquemático de circuitos, a **entidade** é análoga à descrição simbólica, representando o subcircuito através de entradas e saídas, e a **arquitetura** é análoga à esquemática, onde a verdadeira função lógica do subcircuito é descrita. A Figura 18 mostra essa analogia graficamente.

VHDL é considerada uma linguagem de descrição de *hardware* e, diferente de uma linguagem de programação, é capaz de descrever eventos concorrentes como em um circuito real. Cada bloco concorrente descrito opera paralelamente aos demais blocos concorrentes. A ordem de cada linha no código VHDL não afeta o resultado da descrição, pois não existe relação com a ordem de execução, ao contrário de linguagens seqüenciais como por exemplo C, C++, Pascal ou Fortran.

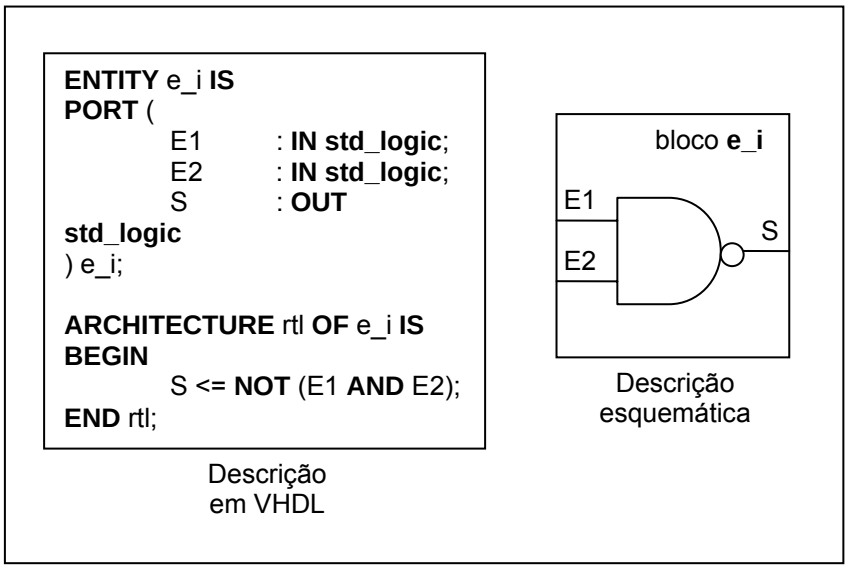


Figura 18: Comparação das descrições do mesmo circuito em esquemático e em *VHDL*

2.4.2 Linguagem Verilog

A segunda das duas mais importantes linguagens de descrição de *hardware* é Verilog. Introduzida em 1985 pela *Gateway Design System Corporation*, agora parte da *Cadence Design Systems*, a linguagem Verilog é largamente utilizada no meio industrial de projetos *VLSI*. Sua sintaxe é muito similar a da linguagem C de programação, o que incentiva seu uso por engenheiros elétricos e de computação já que muitos a aprendem durante o curso da universidade.

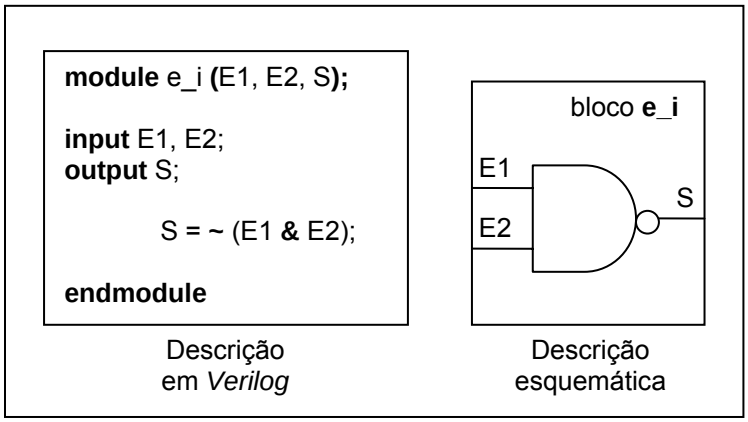


Figura 19: Comparação das descrições do mesmo circuito em esquemático e em Verilog

A Figura 19 mostra o mesmo circuito apresentado na Figura 18, agora descrito em Verilog.

A linguagem Verilog descreve um sistema digital como um conjunto de módulos (*module*), onde cada um possui uma interface para outros módulos a fim de que suas interconexões também possam ser relacionadas na descrição completa do sistema. Os módulos são capazes de operações concorrentes e podem representar desde simples portas lógicas até sistemas completos como, por exemplo, um micro-processador.

A grande maioria das ferramentas de *layout* aceita somente *netlist* sintetizada e mapeada na linguagem Verilog e portanto esse será o formato de entrada para o estudo aqui apresentado.

2.5 Cone de dependências

Cone de dependências [63] é uma técnica empregada na área de teste e verificação de circuitos eletrônicos para determinar quais das entradas de um circuito afetam cada saída do mesmo. Uma vez que essas informações são obtidas, padrões de entradas podem ser gerados de forma a testar cada saída individualmente. A Figura 20 apresenta um simples circuito a ser testado.

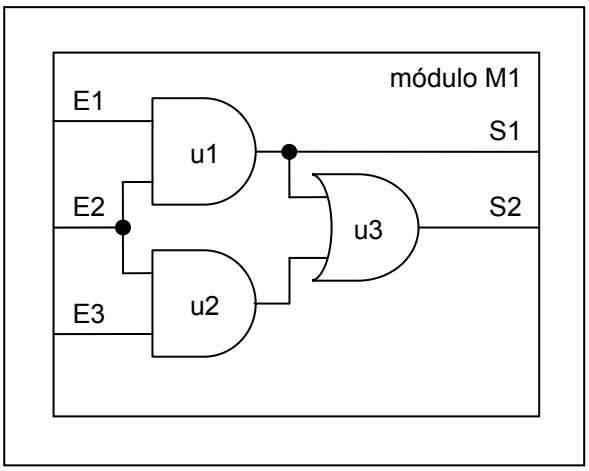


Figura 20: Circuito a ser testado

Nota-se que *S1* é uma saída dependente somente das entradas *E1* e *E2*, enquanto *S2* depende de *E1*, *E2* e *E3*. Os cones de dependências são representados na Figura 21.

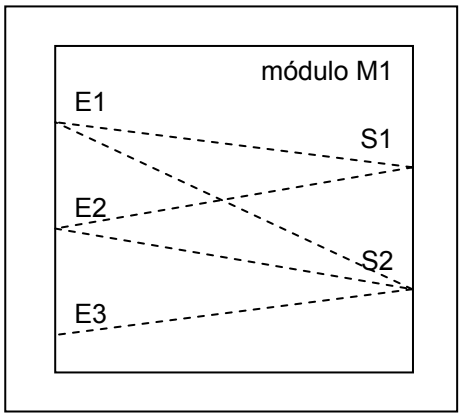


Figura 21: Cone de dependências do módulo M1

Nesse trabalho, a técnica de cone de dependências será utilizada para determinar como cada entrada é propagada dentro de seus módulos e como a sinalização de sincronismo deve ser estabelecida entre as unidades funcionais assíncronas vizinhas.

2.6 Ferramentas de CAD

A manufatura de circuitos integrados sempre foi acompanhada de ferramentas de auxílio por computador – *CAD* – em etapas do seu projeto. A complexidade dos circuitos aumentou ao ponto que, à medida que tecnologia de produção dos integrados evoluiu, é praticamente inviável projetá-los sem ferramentas que permitam a utilização de um nível mais alto de manipulação de informações de entrada, saída e análise dos circuitos. Para o desenvolvimento desse trabalho, as duas ferramentas necessárias são a de síntese e a de análise de tempo estático.

Ferramentas de síntese são as responsáveis por ler o código escrito pelo projetista em uma ou mais linguagens de descrição de *hardware*, compilar em portas lógicas genéricas, mapear na tecnologia adotada e otimizar em relação à performance. O resultado é um código composto por portas lógicas da tecnologia escolhida e a forma como são interligadas. Exemplos dessa ferramenta são *Synopsys Design Compiler* e *Cadence BuildGates*.

A análise de tempo estático é realizada por uma ferramenta específica, capaz de revelar os caminhos de atraso críticos, violações de *setup* e *hold* dos elementos seqüenciais. Provê características de atraso, carga, *fanout*, *fanin*, tempos de subida e

descida de todos os nós do circuito analisado. Exemplos dessa ferramenta são *Synopsys PrimeTime* e *Cadence Pearl*.

2.7 SER

A metodologia de Escalonamento por Reversão de Arestas (SER – *Scheduling by Edge Reversal*) [64] foi elaborada para o auxílio da implementação de processamento distribuído em uma rede de computadores. Neste trabalho a metodologia SER será empregada como uma estrutura de sinalização e sincronismo entre processos independentes, que, no entanto, dependem de recursos comuns. A forma de aplicação da metodologia em circuitos digitais será apresentada na Seção 2.8. Nesta seção será introduzido o conceito do SER.

SER funciona da seguinte forma: partindo de qualquer orientação acíclica ω em G, há pelo menos um nó sumidouro, ou seja, um nó cujas arestas estão todas direcionadas a ele mesmo. Todos os nós sumidouros podem operar enquanto os restantes devem permanecer inativos. Isto favorece a exclusão mútua a qualquer acesso feito aos recursos compartilhados pelos nós sumidouros. Após a operação, um nó sumidouro reverterá a orientação de suas arestas, tornando-se uma fonte, e assim possibilitando o acesso dos recursos aos seus vizinhos. Uma nova orientação acíclica é definida e todo o processo é então repetido para o novo conjunto de sumidouros. Assumindo que $\omega' = g\ (\omega)$ simboliza esta operação, SER pode ser considerada uma repetição infinita da aplicação de g(ω) sobre G. Assumindo que G é finito, é possível perceber que um conjunto de orientações acíclicas eventualmente se repetirá, definindo um período de duração p. Esta simples dinâmica garante que nenhum *deadlock* ou inanição (*starvation*) ocorrerá, já que em cada orientação acíclica há pelo menos um sumidouro, ou seja, um nó autorizado a operar. Além disso, está provado que em qualquer período todos os nós operam exatamente m vezes [64, 65].

SER é um algoritmo cuja dinâmica de grafo é completamente distribuída. Uma das suas propriedades muito interessantes é a generalidade, à medida que qualquer topologia terá seu próprio conjunto de possíveis dinâmicas de SER [64, 65]. A Figura 22 ilustra a dinâmica de SER.

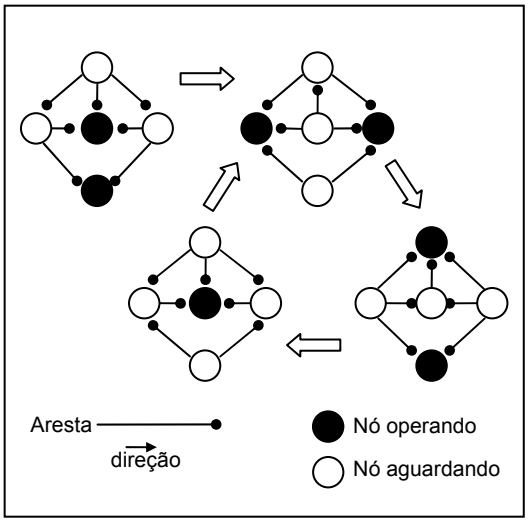


Figura 22: Diagrama de SER

2.8 ASERT

A metodologia ASERT [37, 38] - *Asynchronous Scheduling by Edge Reversal Timing* caracteriza um ambiente de trabalho que permite o compartilhamento de recursos comuns entre processos independentes. Nesta metodologia todos os processos do sistema agem de forma independente, de modo que ocorrem restrições apenas relacionadas à exclusão mútua com seus vizinhos e a sua própria temporização. Do ponto de vista arquitetural, não há restrições quanto à quantidade de processos ou à forma de comunicação entre eles.

A questão de geração de um sistema com tal metodologia pode ser caracterizada por três estágios distintos: a representação do sistema alvo, a construção da rede de temporização e a inicialização do sistema.

No primeiro estágio, como apresentado na Figura 23, o sistema alvo é associado a uma representação planar dos processos (blocos funcionais) e sua vizinhança. Esta é estabelecida a partir da ocorrência ou não da troca de mensagens entre pares de blocos funcionais. Para tal, considera-se que a simples propagação de um conjunto composto por um ou mais sinais elétricos constitui uma mensagem, independentemente da definição de um protocolo específico. A identificação dos blocos funcionais na Figura 23 é feita a partir da ordem de execução dos processos.

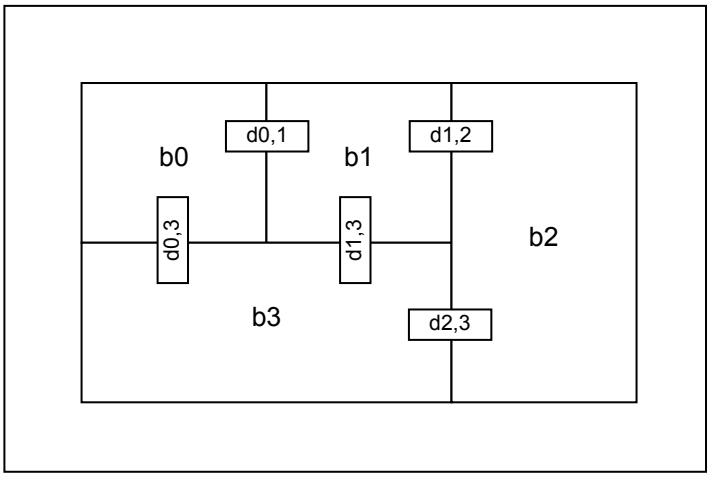


Figura 23:Processos ou blocos funcionais do sistema alvo

O grafo $C = (B, D)$ denota o sistema alvo, sendo B o conjunto de seus processos, ou blocos funcionais, e D o conjunto de arestas definido pela vizinhança de B, ou seja, para cada par de vizinhos b_u e $b_v \in B$ há uma aresta $d_{u,v}$. A Figura 24 representa o grafo C, cuja análise considera a mesma ordem da identificação inicial do sistema alvo, e também a mesma ordem de execução dos processos.

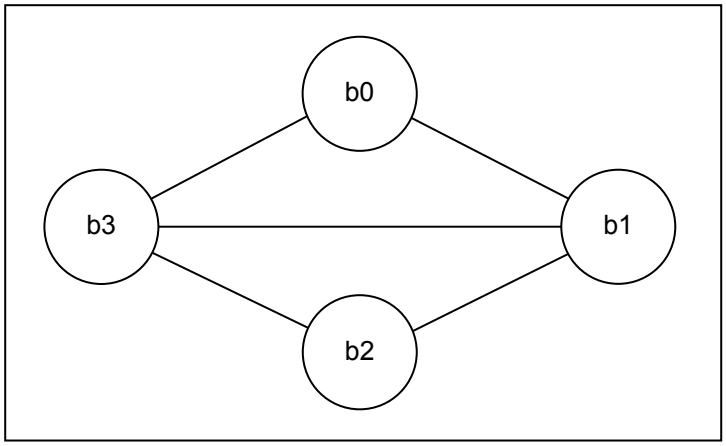


Figura 24: Grafo C = (B,D) gerado para o sistema alvo

A construção da rede de temporização G constitui o segundo estágio do problema de geração de um sistema baseado na metodologia ASERT. O grafo $G = (N,E)$ corresponde à rede auto-osciladora alvo, e sua construção é feita a partir de C. Cada bloco funcional de C corresponde a um nó n_i em G, e a cada aresta d_j de C

corresponde uma outra aresta e_j em G. Nesta etapa, o grafo G ainda não está completo, já que é preciso determinar a orientação das arestas.

O terceiro e último estágio do problema é a inicialização da rede G. Visto que o grafo C é do tipo não orientado, e que há uma correspondência unívoca entre os b_i e os n_i, é preciso promover uma orientação acíclica para G. Durante a construção de G, ao se manter a numeração dos nós conforme a ordem de execução, pode-se obter uma orientação acíclica apropriada através do seguinte critério: para cada aresta que interliga os nós vizinhos n_i e $n_j \in N$, deve-se orientar a aresta no sentido de i para j sempre que i for maior que j.

Para a implementação em *hardware* dos grafos nos quais a metodologia ASERT é aplicada, dois circuitos precisam ser elaborados: o controlador de aresta e o controlador de nó.

O controlador de aresta realiza a sinalização entre dois nós vizinhos. A direção para a qual a aresta aponta é representada pelo nível lógico alto, indicando que a ficha (*token*) está disponível para o nó apontado. Esse circuito pode ser implementado por uma porta lógica do tipo *XOR* e outra do tipo INVERSOR como mostra a Figura 25, tendo como entradas os sinais de fim de operação (*EOP 1* e *EOP 2*) de cada um dos nós conectados e como saídas os sinais indicando a direção da aresta (*Ficha* e $\overline{Ficha}$).

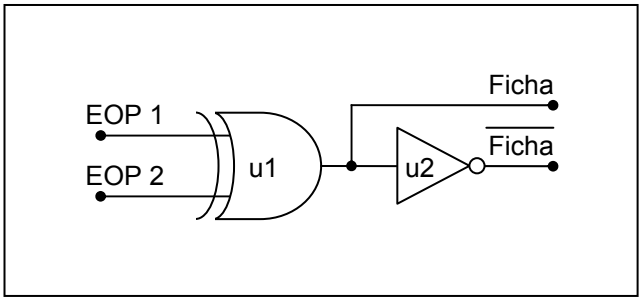


Figura 25: Implementação do controlador de aresta

O controlador de nó é o circuito utilizado para habilitar ou desabilitar a operação de um nó, que observa a presença de fichas, podendo ser implementado por uma porta *AND*, como mostra a Figura 26. Cada entrada desse circuito é conectada a uma das saídas de cada controlador de aresta vinda dos nós vizinhos, enquanto sua saída *opera* é usada para permitir o início da operação do circuito relativo ao nó. A lógica geradora do sinal de fim de operação *EOP* pode ser implementada de diversas maneiras e duas delas

serão analisadas na Seção 3.4. A entrada *run*, caso necessário, é usada para permitir o bloqueio de operação do nó.

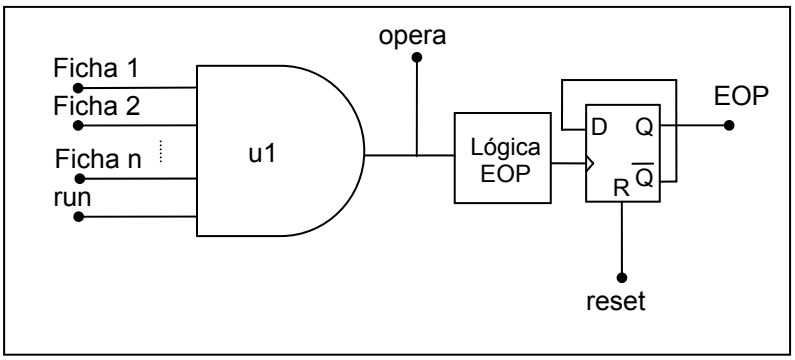


Figura 26: Implementação do controlador de nó

A Figura 27 apresenta uma implementação de como são interconectados um controlador de aresta a dois outros controladores de nós. Os blocos pontilhados delimitam cada um dos elementos.

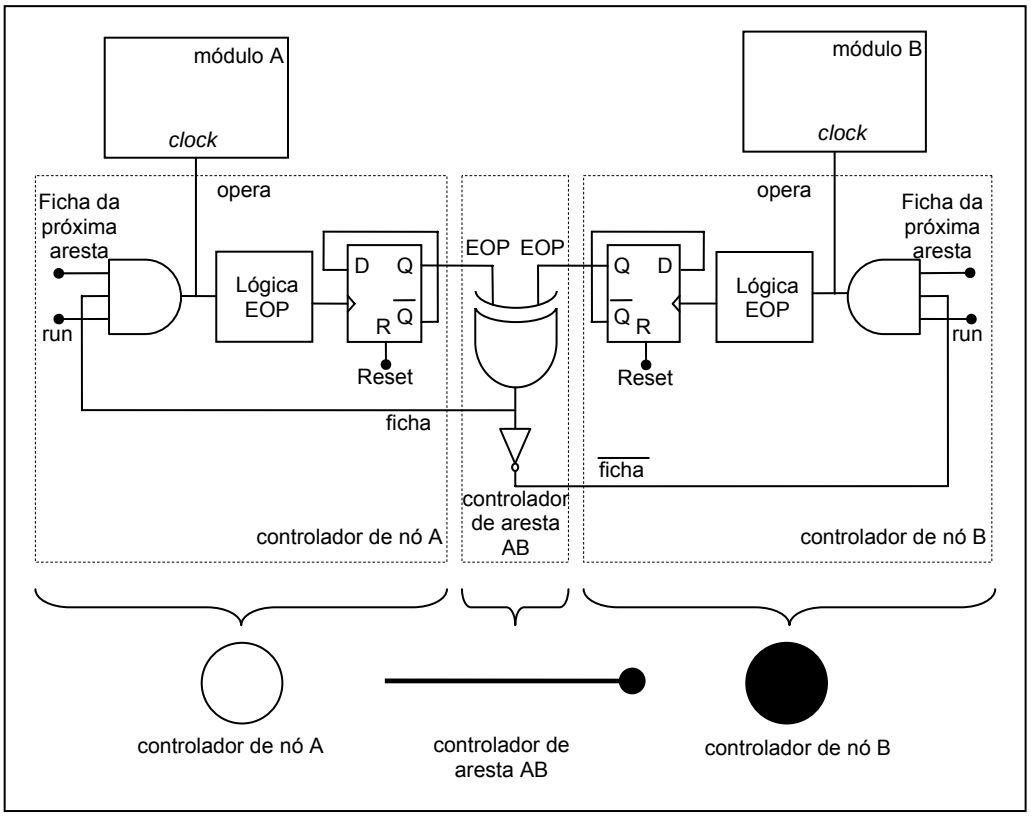


Figura 27: Exemplo de implementação de um controlador de aresta conectado a dois controladores de nós

3 Método de Conversão

A diferença fundamental entre um circuito síncrono e seu equivalente assíncrono se resume ao fato da inexistência de um sinal periódico global que sincroniza a troca de informações entre os diversos tipos de elementos de retenção de valores existentes em um circuito lógico comum, como registradores ou blocos de memória. No entanto, esses elementos estão igualmente presentes em circuitos assíncronos e portanto precisam ser devidamente coordenados para que os dados armazenados não venham a ser perdidos. Existe a necessidade do emprego de uma metodologia que não dependa de uma base temporal central, mas que seja capaz de manter a coerência do funcionamento normal do circuito projetado.

A conversão automatizada de um circuito síncrono para um circuito assíncrono equivalente deve ser capaz de analisar o circuito original e decidir como os diferentes elementos de memorização devem ser agrupados e sinalizados para que a operação original não seja afetada e, concomitantemente, adicionando o mínimo de lógica extra necessária para implementar a sinalização.

A granularidade também deve ser levada em consideração: caso seja muito fina, o *overhead* utilizado na sinalização pode ser muito grande, além da impossibilidade de algumas estruturas funcionarem coerentemente, como apresentado na Capítulo 1.2; e no caso oposto, o agrupamento de um grande número de elementos de memória em um único sinal de sincronismo fere o próprio conceito de assíncrono.

O trabalho desenvolvido é baseado no discernimento daquele que projetou o circuito síncrono e no modo como o mesmo subdividiu blocos funcionais básicos. Bons projetistas de circuitos lógicos levam vários aspectos em consideração no processo de implementação de um sistema, dentre outros, o particionamento de modularidade por tarefas, reutilização de blocos em outros sistemas e paralelismo de operações em diferentes estruturas. Seguindo essa linha, a metodologia de conversão baseia a concepção da granularidade da subdivisão diretamente provinda da organização modular na descrição em linguagem de hardware usada pelo projetista do circuito síncrono.

A metodologia apresentada para conversão de circuitos síncronos em assíncronos é fundamentada na associação de cada módulo da descrição do hardware a um nó do modelo ASERT. O processo de geração das arestas é baseado na informação extraída das interligações entre módulos, pois essas indicam dependências no fluxo de

sinais do circuito. As correlações entre módulos e nós ASERT, assim como entre conexões e arestas ASERT, serão introduzidas nas próximas secções.

3.1 Correlação entre módulos e nós ASERT

Como visto na Seção 2.8, a metodologia ASERT emprega nós e arestas como elementos básicos para a construção de uma topologia de sinalização assíncrona. Os nós são responsáveis por identificar o momento em que todos os recursos necessários para o correto funcionamento do bloco sob seu controle estão disponíveis, habilitá-lo a operar e comunicar aos outros nós quando sua operação foi terminada. Da mesma forma, um circuito pode ser expresso a partir de um conjunto de elementos compostos por blocos e interligações dos mesmos através de uma linguagem de descrição de *hardware*. Esses blocos, chamados *módulos* na linguagem *verilog* ou *entidades* na linguagem *VHDL*, são definidos por uma interface de sinais de entradas e saídas, e pela descrição de sua funcionalidade propriamente dita. Projetistas de circuitos empregam *módulos* a fim de encapsular arquiteturas bem definidas e até certo ponto autocontidas com o objetivo de diminuir a complexidade da estrutura como um todo e sua utilização se limita unicamente em referências em outros *módulos* que os englobam hierarquicamente. A visão global do planejador do sistema é aplicada diretamente na correlação do agrupamento em *módulos* com nós ASERT. Esses elementos dependem de recursos alheios como pré-requisito para operar assim como um *módulo* depende de entradas válidas para gerar saídas consistentes. A seguir serão apresentadas, através de exemplos, como são realizadas as associações entre módulos e nós.

A Figura 28 representa um circuito seqüencial *M* que possui 6 entradas *E[1-6]*, 6 portas lógicas combinacionais *u[1-6]*, uma porta lógica seqüencial e uma saída *S1*. Nesse caso, *S1* assume um valor exclusivamente dependente de um conjunto das entradas *E[1-6]*, disponibilizadas por outros módulos ou entradas primárias.

Supondo que, para cada uma dessas entradas, seja associado um sinal indicando que um novo valor está disponível, é possível discriminar o instante em que o módulo *M* é capaz de gerar uma saída *S1* coerente, produzindo, por sua vez, a mesma sinalização de fim de operação quando for realizada. A mérito de esclarecimento, o termo *novo valor* não significa necessariamente um valor diferente do anterior, mas sim o término de cálculo do bloco gerador do sinal em questão, que pode apresentar ou não

um diferente valor binário. A Figura 29 apresenta um diagrama de dependências das sinalizações do módulo *M*, e suas entradas e saída.

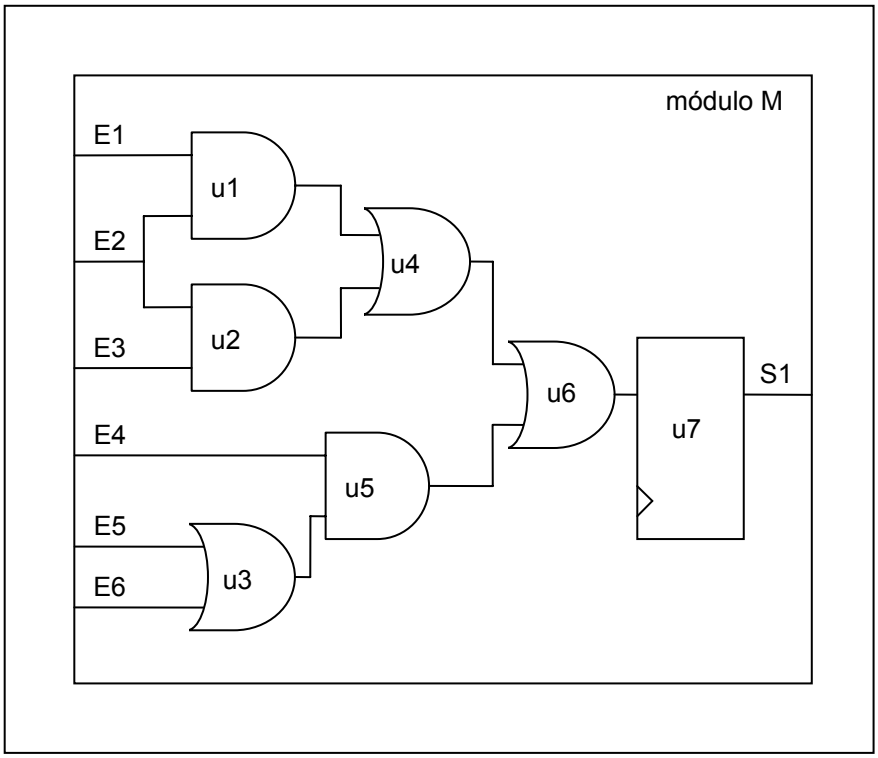


Figura 28: Exemplo de um circuito seqüencial

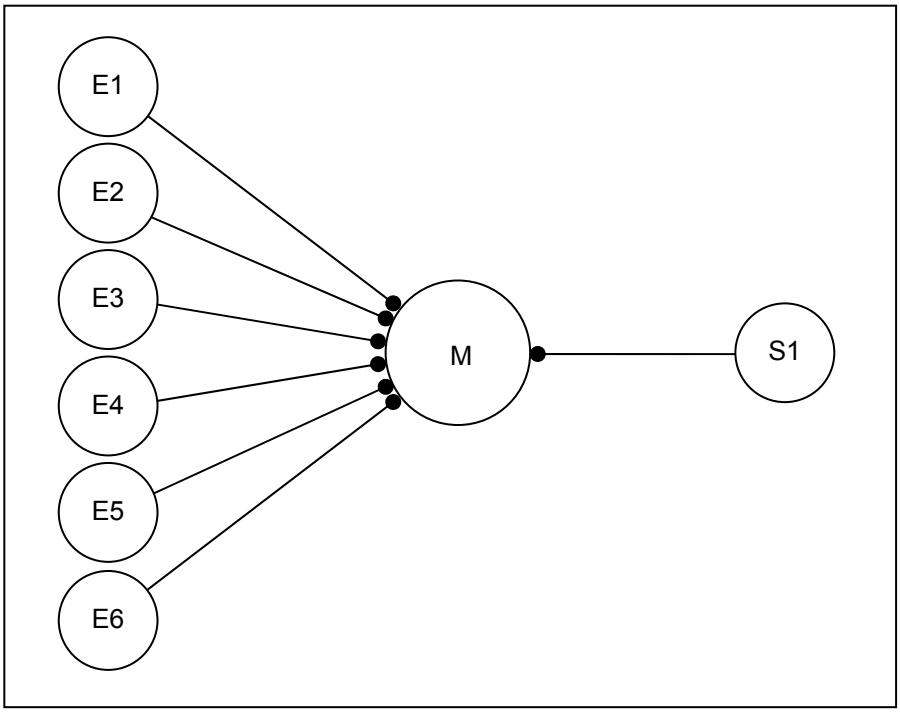


Figura 29: Diagrama de dependências de entradas e saídas do circuito combinacional

O círculo *M* representa a função lógica realizada pela descrição do funcionamento do módulo *M*; os círculos *E[1-6]* representam os blocos fontes que geram as entradas *E[1-6]* para o módulo *M*; e o círculo *S1* representa o bloco destino receptor da saída *S1* do módulo *M*.

Assim como os nós de um circuito controlado por ASERT, as entradas e saídas de um módulo precisam ter todas as suas dependências disponíveis para que possam operar adequadamente e gerar um resultado correto, que por sua vez será consumido pelo módulo conectado em sua saída. Extrapolando esse conceito, a topologia modular da descrição do hardware pode ser mapeada diretamente em uma topologia ASERT, onde cada módulo é convertido em um nó e cada interconexão é convertida em uma aresta.

As conexões entre módulos serão analisadas a seguir. A fim de facilitar a nomenclatura, a saída *Sx* de um módulo *Mx* será representada da forma *Mx/Sx*, assim como a entrada *Ex* do módulo *Mx* será simbolizada por *Mx/Ex*.

3.2 Correlação entre interconexões de módulos e arestas ASERT

Assim como os nós, as arestas realizam um papel vital no funcionamento da metodologia ASERT para circuitos assíncronos. Esses são os elementos responsáveis pela transferência da informação relativa ao estado operacional em que os nós interligados por eles se encontram, habilitando ou proibindo seu funcionamento. Da mesma forma que módulos da representação de circuitos em uma linguagem de descrição de *hardware* podem ser mapeados diretamente em nós de uma topologia equivalente no método ASERT de sinalização assíncrona, como apresentado na Seção 3.1, as interligações entre módulos através de sinais e barramentos podem ser utilizadas para extrair informações suficientes para a geração de arestas entre dois nós interconectados. A Figura 29 mostra o módulo *M*, suas entradas *E[1-6]* e sua saída *S1*. Em um circuito completamente conectado, como assumido nesse caso, cada uma das entradas do módulo *M* deve estar conectada a uma saída de um outro módulo ou de uma entrada primária do sistema. Essas interligações são realizadas através da declaração de fios individuais ou de um conjunto de fios que carregam uma informação codificada, os barramentos, em conjunto com a definição do módulo provedor do sinal e sua respectiva

saída a ele conectada, ou ainda da porta de entrada do sistema, e do módulo consumidor e sua respectiva entrada. A informação extraída desses dados indica exatamente como os nós transpostos dos módulos devem ser correlacionados e que, por conseguinte, as arestas podem ser derivadas diretamente das interligações por sinais entre dois módulos através de hierarquias da descrição do circuito alvo, onde cada aresta é única para cada conjunto único de pontos de origem e fim. Em situações em que a saída de um módulo ou uma entrada primária esteja associada a mais de uma entrada em módulo ou de uma saída primária, é necessária a criação do número de arestas correspondente à quantidade de destinos existentes para cada uma das fontes. Na condição em que existam mais que um conjunto de ligações entre dois módulos, no caso de vários bits de um mesmo barramento de dados, por exemplo, a representação é realizada por uma única aresta, como será apresentado na Seção 3.3. A seguir, é apresentado um exemplo da elaboração de arestas a partir da descrição de um circuito.

A Figura 30 apresenta um diagrama de blocos de um circuito descrito por 5 módulos interconectados, onde *M1* e *M2* são blocos geradores de dados e o módulo *M3* tanto consome os dados vindos dos 2 primeiros como fornece um resultado para *M4* e *M5*, que neste caso são simples consumidores.

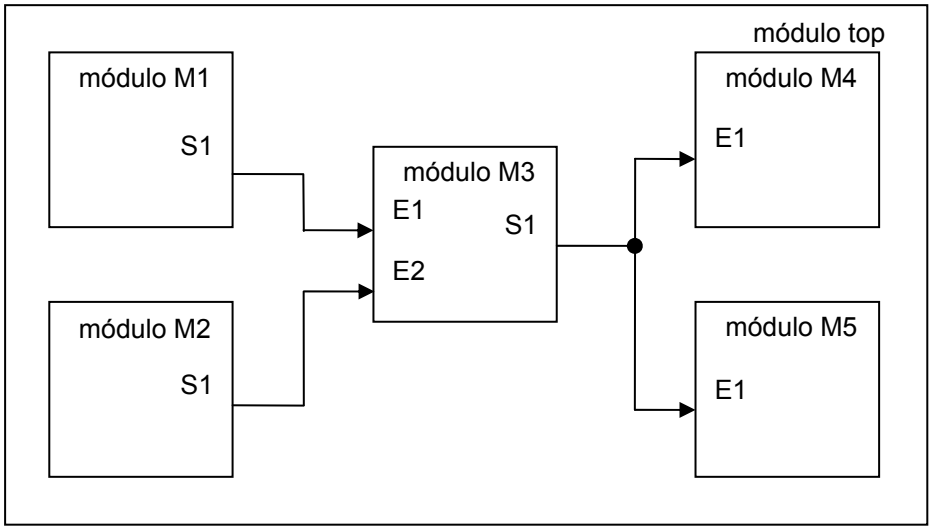


Figura 30: Circuito expresso em uma linguagem de descrição de hardware

M3/S1 terá um valor coerente quando ambos *M3/E1* e *M3/E2* possuírem entradas válidas provindas de *M1/S1* e *M2/S1* e respectivamente comunicadas. Ao mesmo tempo *M3* deve sinalizar para os módulos *M4* e *M5* que um novo *M3/S1* foi

calculado para que esses façam uso de *M4/E1* e *M5/E1* respectivamente. Esse mecanismo de sinalização é perfeitamente solucionado utilizando arestas ASERT, pois permitem que a informação de fim de operação de cada nó seja facilmente trocada entre dois módulos.

O código para a descrição desse exemplo na linguagem *Verilog* está exemplificado na Figura 31. Os fios *M1S1_M3E1*, *M2S1_M3E2* e *M3S1_M4E1M5E1* conectam respectivamente a saída *M1/S1* à entrada *M3/E1*, a saída *M2S1* à entrada *M3/E2* e a saída *M3/S1* às entradas *M4/S1* e *M5/S1*. O número total de declarações de fios é 3, mas o número de interconexões é 4, já que *M3/S1* tem mais de um destino, mesma quantidade de arestas necessárias para a correta sinalização do método ASERT.

A sinalização ASERT do circuito descrito na Figura 30 deve ser realizada como indica a Figura 32, contendo 5 nós mapeados diretamente dos 5 módulos existentes interconectados pelas 4 arestas transpostas dos sinais de interligações dos módulos.

```
module top ();

wire M1S1_M3E1;
wire M2S1_M3E2;
wire M3S1_M4E1M5E1;

M1 instancia_M1 (.S1 (M1S1_M3E1));
M2 instancia_M2 (.S1 (M2S1_M3E2));
M3 instancia_M3 (.E1 (M1S1_M3E1),
                 .E2 (M2S1_M3E2),
                 .S1 (M3S1_M4E1M5E1));
M4 instancia_M4 (.E1 (M3S1_M4E1M5E1));
M5 instancia_M5 (.E1 (M3S1_M4E1M5E1));

endmodule
```

Figura 31: Código de descrição em Verilog

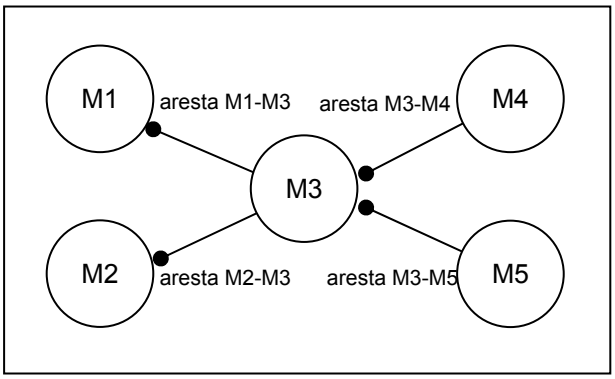


Figura 32: Aplicação de arestas ASERT entre módulos interdependentes

Na prática, a informação necessária para a criação das arestas a fim de realizar a conversão síncrono-assíncrono está disponível no próprio código que descreve o circuito. Uma lista contendo todos os módulos e suas respectivas saídas é extraída da descrição e percorrida por um algoritmo que identifica o conjunto de entradas de módulos às quais estão conectadas, permitindo que uma tabela de interdependências entre saídas e entradas de módulos seja anotada e utilizada para a geração das arestas ASERT.

3.3 Mapeamento de uma descrição de hardware na metodologia ASERT

A metodologia apresentada propõe uma técnica para conversão de um circuito síncrono em um circuito assíncrono equivalente, tornando dispensável a existência de um sinal global de sincronismo. É necessário, portanto, a garantia de que as características de um circuito síncrono sejam refletidas na versão assíncrona. Circuitos síncronos se baseiam, na sua forma mais simples, no fato de que informações temporárias são retidas em registradores, elementos de memória, constituindo um estado particular a cada ciclo de *clock*, como apresentado na Figura 33 e na Figura 34.

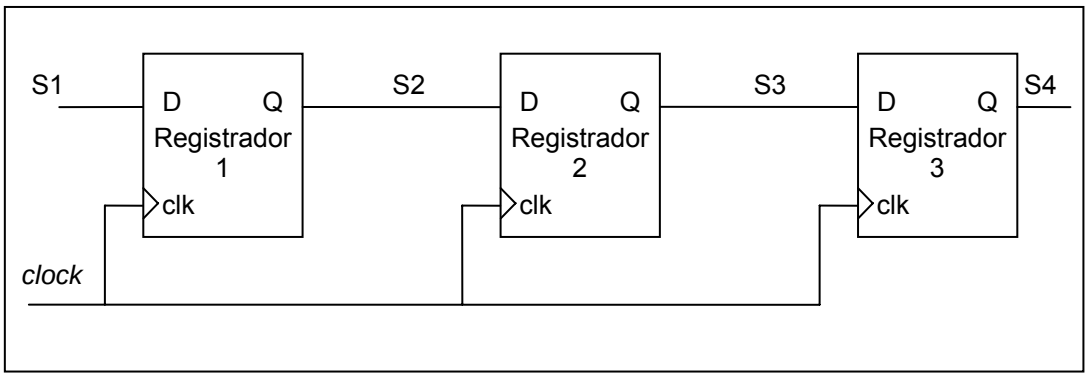


Figura 33: Circuito síncrono

À primeira borda de subida do *clock*, o *Registrador 1* retém o primeiro dado válido *Dado1* e a cada transição positiva subseqüente do *clock*, o registrador abaixo retém o valor retido pelo registrador acima no ciclo anterior.

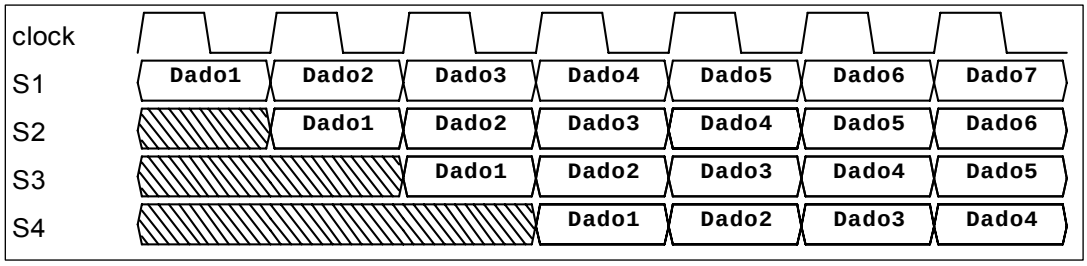


Figura 34: Forma de onda de funcionamento do circuito síncrono

Na versão assíncrona, a sinalização entre elementos de memória deve ser capaz de reproduzir o mesmo comportamento para que seja consistente com o funcionamento original síncrono. A Figura 35 exemplifica como a conversão de uma seqüência de registradores é realizada.

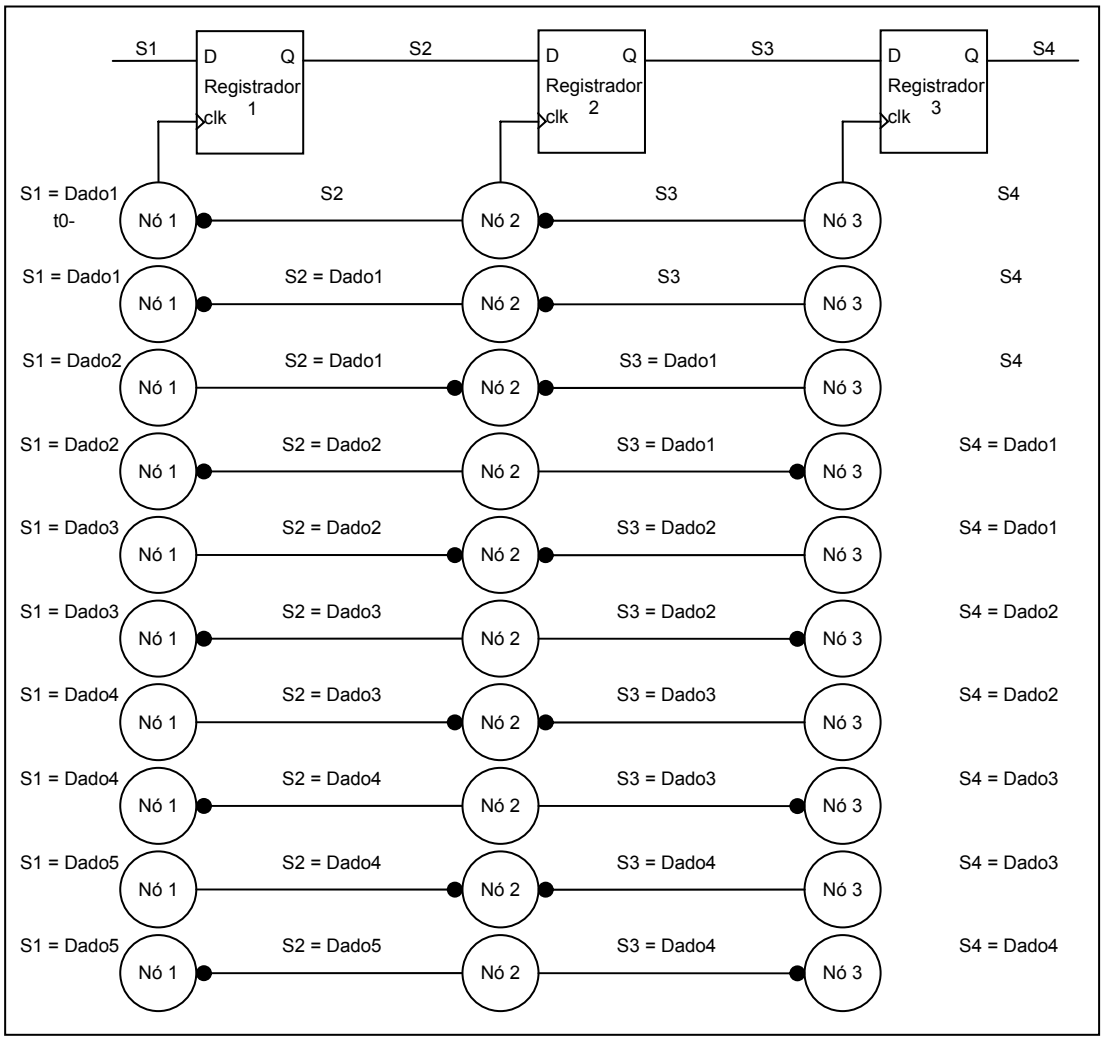


Figura 35: Sinalização assíncrona aplicada ao exemplo anterior

Nó 1, *Nó 2* e *Nó 3* são controladores de nós ASERT, implementados como apresentado na Seção 2.8. Cada controlador tem sua saída *opera* conectada à entrada de *clock* do seu respectivo registrador. Dessa forma, quando as arestas associadas a cada controlador estiverem direcionadas ao mesmo, o registrador é acionado e copia o valor presente na sua entrada de dados, exatamente como a contraparte síncrona. O método de conexão dos nós exposto no exemplo não é o mais otimizado, por obrigar os estágios vizinhos a se manterem estacionários enquanto um nó está em modo de operação. Métodos de otimização espaciais e temporais serão apresentados nas seções 3.3.1 e 3.3.2.

O exemplo da Figura 35 demonstra que a metodologia de conversão síncrona-assíncrona deve focalizar a atenção nos elementos de armazenamento, pois esses são os responsáveis por manter a integridade do funcionamento original. Conseqüentemente, os módulos mapeados em nós ASERT devem ser devidamente selecionados, limitando-se apenas aos que possuem caminhos que cruzem os blocos hierárquicos através de componentes de memorização. Dessa forma, as correlações entre módulos e interconexões a nós e arestas ASERT respectivamente, apresentadas nas seções 3.1 e 3.2, devem obedecer a esse critério.

O primeiro passo a ser realizado no processo de conversão é reconhecer quais os módulos que possuem elementos de memória entre suas entradas e saídas. Um exemplo é o módulo *M1* na Figura 36, circuito constituído das portas lógicas combinacionais *u1*, *u2* e *u3* e das portas seqüenciais *u4* e *u5*. *M1/S1* é uma saída que depende dos valores de *M1/E1* e *M1/E2*, enquanto *M1/S2* depende de *M1/E1*, *M1/E2* e *M1/E3*, ambas possuindo registradores em seus caminhos, tornando *M1* um candidato apto a se transformar em nó pelo método de conversão.

Outro fator a ser levado em consideração é o conjunto das combinações de entradas responsáveis por gerar saídas coerentes. Os cones de dependência (Seção 2.5) de um módulo são extraídos do circuito original, como exemplificado na Figura 37, que mostra o cone de dependências do módulo *M1*, com o objetivo de identificar esses pares de dados e promover as correlações bilaterais necessárias para as interconexões entre módulos.

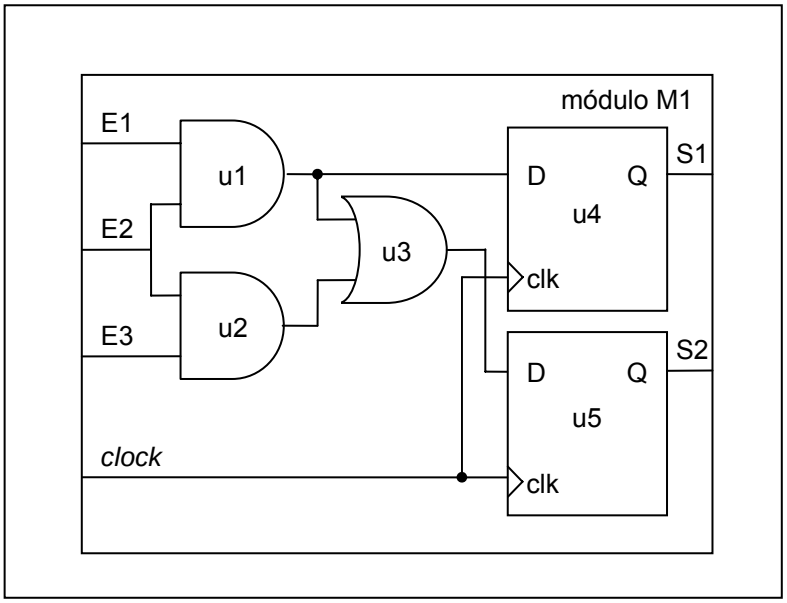


Figura 36: Módulo M1 com elementos combinacionais e seqüenciais

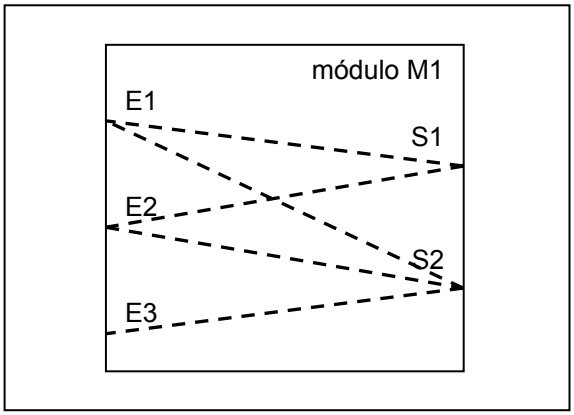


Figura 37: Diagrama do cone de dependências do módulo M1

A Figura 38 insere o módulo *M1* em um contexto com outros módulos representados por seus diagramas de cone de dependências, onde as linhas tracejadas em negrito identificam ramos dos cones de dependências que possuem registradores em seu caminho e as linhas normais indicam não possuir elementos de memória. Nessa composição, as dependências de entradas e saídas de um circuito podem ser usadas para interpretar como os módulos devem ser correlacionados visando a procedência e o destino de suas entradas e saídas. O valor de *M1/S1* é resultado da combinação dos valores das saídas *M2/S1* e *M0/S1*, pois ambas possuem ao menos um elemento de memorização no processo de geração em seus respectivos módulos. Já a saída *M1/S2* é

dependente dos valores de *M0/S1*, *M2/S1* e *EP4*, todavia não é considerada dependente de *M2/S2*, visto que no caminho interno ao módulo *M2* não há nenhuma forma de retenção de dados que conclua na saída *S2*. Portanto, o caminho que tem início na entrada primária *EP4* atravessa transparentemente o módulo *M2* e termina registrado por um determinado elemento pertencente ao módulo *M1*.

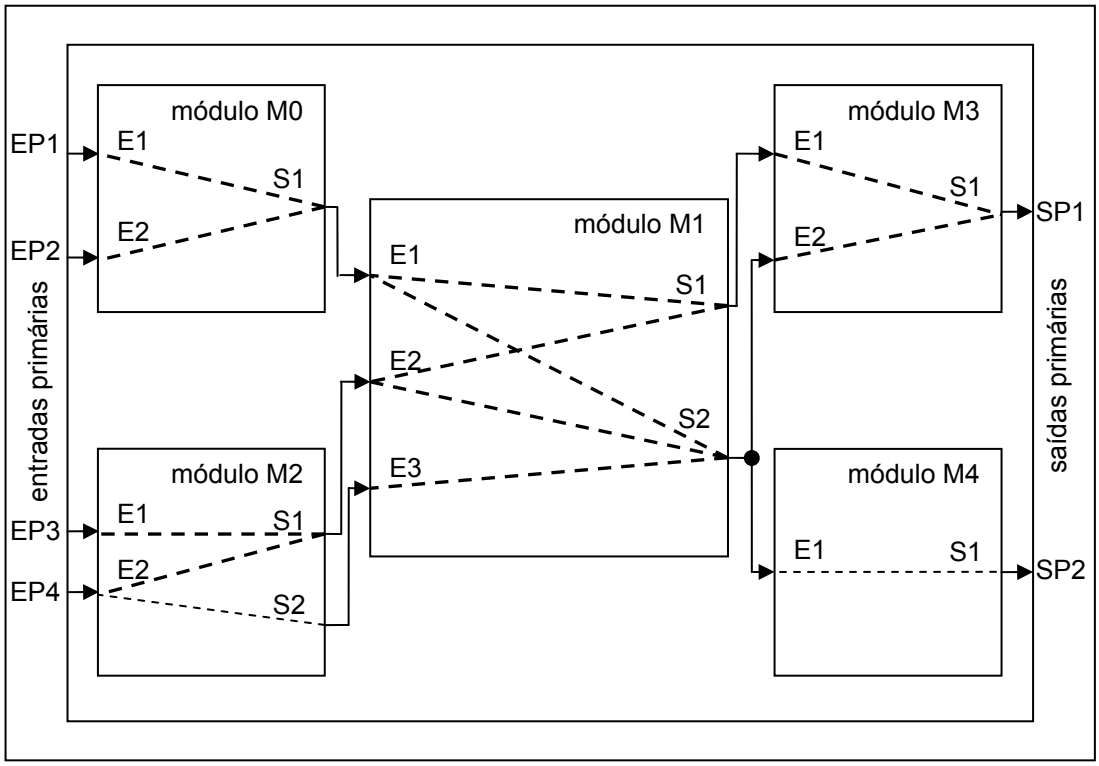


Figura 38: Módulo M1 inserido em um circuito exemplo

Da mesma forma, *M1/S1* causa uma variação na saída *M3/S1*, ao passo que *M1/S2* influencia no resultado de *M3/S1*, assim como no valor de *EP2*, pois o caminho formado pelo par *M4/E1* e *M4/S1* não é amostrado, não sendo considerado um candidato para conversão em um nó ASERT.

A fim de simplificar a análise, a Tabela 1 resume as dependências que existem entre módulos, entre entradas primárias e módulos, e entre módulos e saídas primárias.

Para que a conversão seja adequadamente realizada, é essencial que o cone de dependências entre entradas e saídas de cada um dos módulos do sistema seja extraído e analisado tanto individualmente, como entre os demais módulos. *M1*, por exemplo, não deve operar caso estejam disponíveis novos valores das entradas *M1/E1* e *M1/E3*, pois nenhuma das saídas *M1/S1* e *M1/S2* terá um valor coerente se a entrada *M1/E2* não

possuir um novo valor. Nesse caso, somente quando as 3 entradas *M1/E1*, *M1/E2* e *M1/E3* estiverem disponíveis, o módulo *M1* tem condições de operar apropriadamente.

Tabela 1: Dependências entre módulos, entradas e saídas primárias do circuito da Figura 38

Nó	Recursos Geradores	Recursos Consumidores
M0	EP1 EP2	M1
M1	M0 M2 EP4	M3 SP2
M2	EP3 EP4	M1
M3	M1	SP1
EP1	Externo	M0
EP2	Externo	M0
EP3	Externo	M2
EP4	Externo	M1 M2
SP1	M3	Externo
SP2	M1	Externo

O próximo passo a ser realizado no processo de conversão síncrono-assíncrono é a compilação das informações obtidas nos passos anteriores e concluir na associação de nós do método ASERT a cada um dos módulos candidatos assim como na correspondência entre arestas e as conexões extraídas da topologia de interconexões dos módulos, e entradas e saídas primárias.

Apesar do circuito nesse ponto estar operando em um ambiente isento de uma referência central de sincronismo, ainda há a necessidade de se efetuar uma interface que estabeleça a troca de informações com o meio externo. Nesse caso, é indispensável determinar um protocolo que possibilite que as fontes geradoras de entradas primárias possam indicar que um novo valor está disponível para ser consumido pelo sistema, assim como o próprio sistema tem que ser capaz de sinalizar quando novas saídas estão prontas para serem utilizadas pelos seus receptores. Como qualquer entrada primária pode ter inúmeros módulos ou saídas primárias como o destino e, como qualquer saída primária pode ter inúmeros módulos ou entradas primárias como fonte de geração de

dados, a metodologia de conversão associa cada entrada ou saída primária a um nó ASERT, como uma forma de canalização e isolamento da interação entre o ambiente em que o sistema se encontra e o próprio. Dessa forma, a aplicação do circuito convertido só é possível caso o ambiente externo seja capaz de fornecer um sinal informando que um novo dado está disponível para o consumo, assim como compreender quando o sistema sinaliza que um novo dado está preparado para ser usado. O diagrama ASERT resultante do circuito exemplo sugerido pode ser representado como na Figura 39.

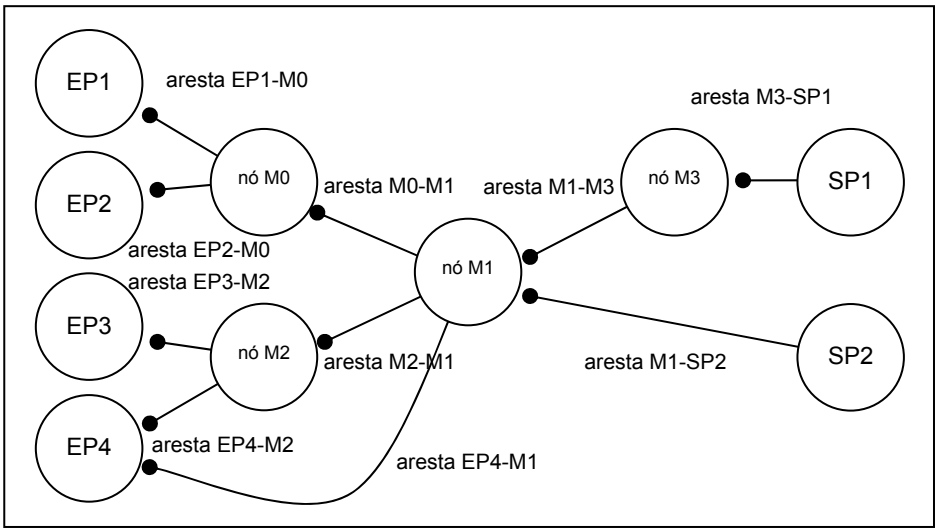


Figura 39: Diagrama ASERT do circuito da Figura 38

Do ponto de vista de um nó ASERT, as entradas de seu controlador refletem os estados em que se encontram os controladores de nós vizinhos, com informações que indicam a cada instante se estão em modo de operação ou estão prontos para receber ou prover o próximo dado. Essas informações são passadas pelos controladores de arestas e determinam o momento em que o nó deve operar, implicando fundamentalmente em habilitar a retenção de dados do elemento de memória associado ao mesmo controlador. No exemplo da Figura 39, os controladores de nós operam dinamicamente como apresentado no diagrama de tempo da Figura 40. Quando *M0* for sinalizado que as entradas primárias *EP1* e *EP2* estão disponíveis, o controlador do nó *M0* tem sua saída de início de operação acionada, habilitando *M0* a executar sua função. Após um intervalo pré-calculado $\Delta 0$, a saída *M0/S1* possui um valor estável e o controlador do nó *M0* emite um sinal de fim de operação. O módulo *M2* funciona basicamente da mesma forma que *M0*. Quando terminarem sua execução, ambos os módulos terão suas arestas apontadas para *M1*, que por sua vez entrará em operação. Desta forma cada bloco é

acionado somente quando for apropriado e os caminhos de dados e de controles são seguidos normalmente.

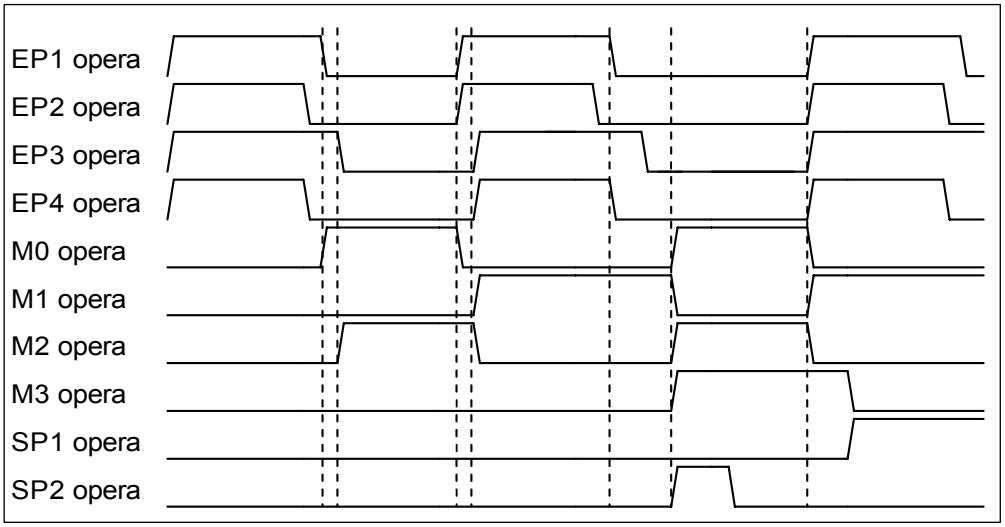


Figura 40: Diagrama de tempo da dinâmica de execução do circuito exemplo

O exemplo introduzido na Figura 38 foi concebido de forma a facilitar a visualização e compreensão da concepção da técnica de associação de módulos a nós da metodologia ASERT de sinalização assíncrona, caracterizado por uma estrutura praticamente plana, contendo apenas módulos auto-contidos dispostos em um sistema que descreve suas interconexões. A técnica, em última instância, detecta e seleciona módulos com saídas registradas em todos os níveis hierárquicos da descrição do sistema como candidatos adequados à correlação com controladores nós, não somente como fontes geradoras dos sinais, mas como destinos consumidores dos mesmos.

A Figura 41 propõe um algoritmo para implementação.

3.3.1 Otimização Espacial

Embora a metodologia desenvolvida seja capaz de transformar automaticamente um circuito genérico síncrono em um circuito assíncrono equivalente sinalizado pelo método ASERT, a implementação final apresenta rigidez de operação devido à granularidade grossa resultante da técnica de conversão que associa cada módulo retentor de dados a um nó ASERT. A redução da flexibilidade de funcionamento do circuito completo é fruto da aglutinação de caminhos até então tidos como independentes embutidos em um mesmo módulo, que os tornam atrelados por serem todos responsáveis por indicar que aquele módulo/nó deve operar, através de suas

interconexões com outros módulos ou entradas e saídas primárias e seus respectivos controladores de nó. O módulo *M1* da Figura 38, por exemplo, é habilitado a operar somente quando todas as arestas relativas às interconexões associadas às suas entradas *M1/E1*, *M1/E2* e *M1/E3* sinalizam ao controlador de nó responsável que valores novos estão disponíveis, possibilitando a geração de saídas coerentes em *M1/S1* e *M1/S2*. No entanto, do ponto de vista de cada uma das saídas do módulo, as entradas essenciais para que individualmente seja gerado um valor coerente não é necessariamente o conjunto completo das entradas disponíveis do bloco. A saída *M1/S1* possui todos os pré-requisitos que necessita para gerar um resultado consistente assim que novos valores estiverem disponíveis nas entradas *M1/E1* e *M1/E2*. No entanto, o método de conversão obriga que a entrada *M1/E3* também possua um novo valor para que o módulo *M1* por completo seja capaz de operar, o que limita a capacidade de operações paralelas enquanto forem contidas em um módulo regulado por um único controlador de nó.

```
loop módulo_fonte em <todos os módulos do sistema>
  se módulo_fonte possui <saída registrada>
    se módulo_fonte não associado nó_fonte
      cria_nó nó_fonte
    fim_se
    loop módulo_destino em <todos os módulos do sistema>
      se módulo_fonte conecta módulo_destino
        se módulo_destino não associado nó_destino
          cria_nó nó_destino
        fim_se
        cria_aresta nó_fonte nó_destino
      fim_se
    fim_loop
  fim_se
fim_loop
```

Figura 41: Descrição da metodologia em forma de algoritmo

Estendendo o conceito da relevância do ponto de vista das saídas, o método de conversão deve associar um controlador de nó para cada saída de todos os módulos cujos caminhos possuam elementos de retenção de dados, permitindo com isso que

saídas que dependam de entradas distintas possam ser calculadas assim que suas dependências forem disponibilizadas. Dessa forma, cada saída individual do módulo possui um conjunto específico de entradas provindas de saídas de outros módulos, de entradas primárias, de saídas primárias ou ainda de módulos que consumirão o resultado da saída em questão, que compõem o requisito mínimo para que ocorra a execução de sua operação.

Reanalisando o módulo *M1*, as seguintes situações podem vir a ocorrer:

- o caminho que produz a saída *M1/S1* pode operar assim que as saídas *M0/S1* e *M2/S1* tenham valores disponíveis e que a saída *M3/S1* esteja pronta para consumir simultaneamente o resultado *M1/S1;*
- o caminho responsável por gerar a saída *M1/S2* pode operar quando novos valores de *M0/S1*, *M2/S1* e *EP4* estão disponíveis e as saídas *M3/S1* e *SP2* estão prontas para consumir o resultado *M1/S2*; e
- ambos os caminhos podem operar assim que todos os recursos descritos nos itens anteriores estejam disponíveis.

A Tabela 2 resume os conjuntos de recursos que exprimem o novo modo de funcionamento do exemplo na Figura 38.

No circuito exemplo da Figura 38, o módulo *M1* é o único capaz de se beneficiar da otimização espacial introduzida nessa seção por possuir mais de uma saída independente registrada na sua descrição. O diagrama resultante da nova técnica de aplicação do método ASERT no circuito é apresentado na Figura 42.

A Figura 43 mostra o diagrama de tempo após a inclusão de mais um controlador de nó no módulo *M1*. Um exemplo simples dos efeitos da otimização é a geração da saída primária *SP2*. Como o caminho que gera a saída *M1/S2* é mais curto que o caminho gerador de *M1/S1* e a temporização de cada um dos controladores de nó age de forma independente, a saída *SP2* é disponibilizada ao meio externo assim que *M1/S2* estiver resolvido, ao contrário do método anterior que dependeria de ambos *M1/S2* e *M1/S1* para que isso acontecesse.

Ao associar um controlador de nó a cada saída registrada de um módulo, a granularidade dos nós é reduzida, possibilitando o aumento da dinâmica de operação assim como da capacidade de paralelismo de execução do bloco projetado, que eventualmente possa ter conglomerado um excessivo número de funções, entradas e saídas em um mesmo módulo.

Tabela 2: Dependências dos nós associados a cada saída do circuito da Figura 38

Nó	Recursos Geradores	Recursos Consumidores
M0/S1	EP1 EP2	M1/S1 M1/S2
M1/S1	M0/S1 M2/S1	M3/S1
M1/S2	M0/S1 M2/S1 EP4	M3/S1 SP2
M2/S1	EP3 EP4	M1/S1 M1/S2
M3/S1	M1/S1 M1/S2	SP1
EP1	Externo	M0/S1
EP2	Externo	M0/S1
EP3	Externo	M2/S1
EP4	Externo	M1/S2
SP1	M1/S1 M1/S2	Externo
SP2	M1/S2	Externo

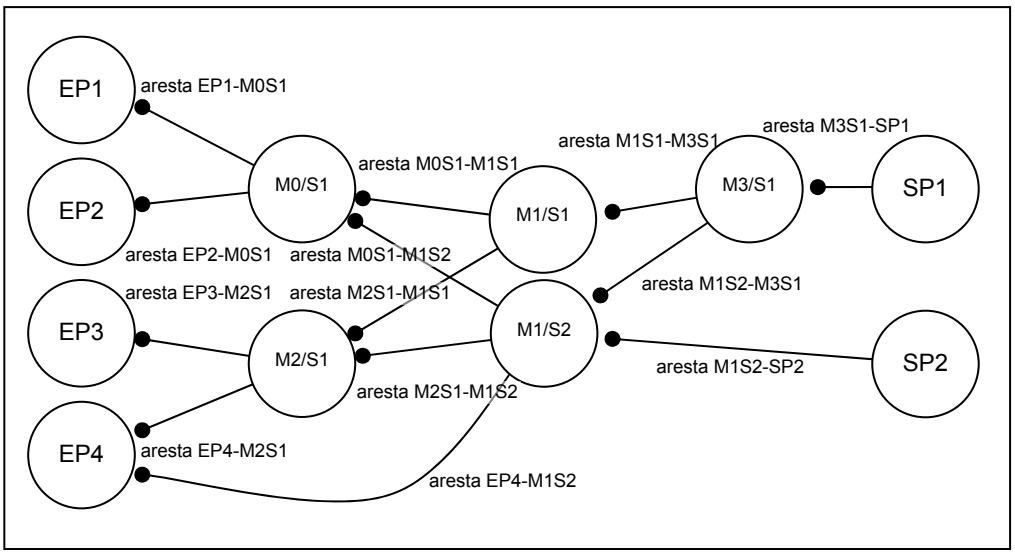


Figura 42: Controladores de nós associados a cada saída registrada do circuito da Figura 38

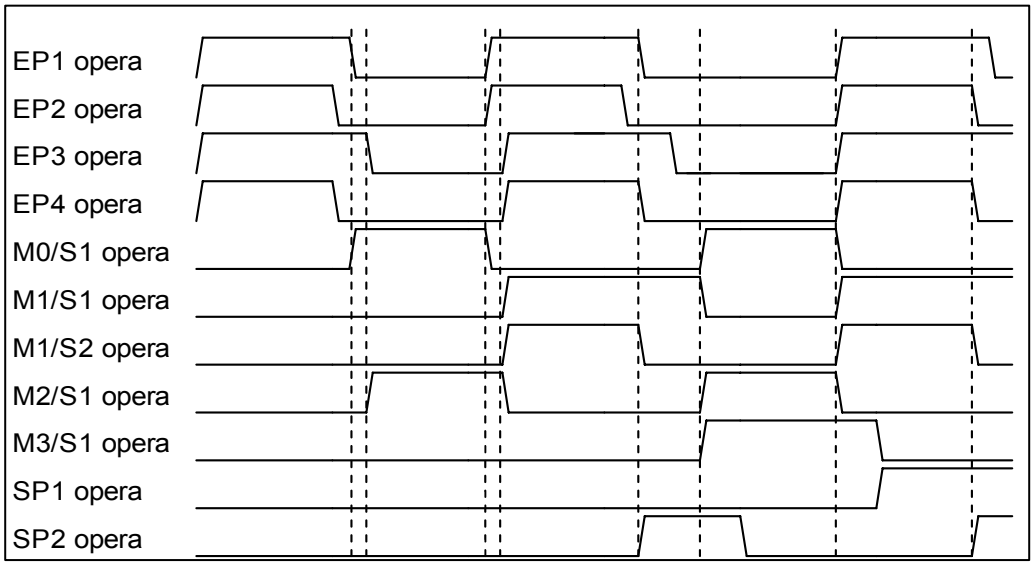


Figura 43:Diagrama de tempo com controladores de nós associados a saídas

O algoritmo de geração dos controladores de nós interconectados por arestas ASERT é corrigido como mostra a Figura 44.

```
loop módulo_fonte em <todos os módulos do sistema>
  loop saída_fonte em <todas as saídas do módulo_fonte>
    se saída_fonte registrada
      se saída_fonte não associado nó_fonte
        cria_nó nó_fonte
      fim_se
      loop módulo_destino em <todos os módulos do sistema>
        se módulo_fonte conecta módulo_destino
          loop saída_destino em <todas as saídas do módulo_destino>
            se saída_fonte conecta saída_destino
              se saída_destino não associado nó_destino
                cria_nó nó_destino
              fim_se
              cria_aresta nó_fonte nó_destino
            fim_se
          fim_loop
        fim_se
      fim_loop
    fim_se
  fim_loop
fim_loop
```

Figura 44: Algoritmo da metodologia com otimização espacial

3.3.2 Otimização Temporal

Uma interconexão pode ser interpretada como um recurso comum compartilhado entre dois blocos independentes definidos por elementos de retenção de dados: um bloco fonte, que utiliza a interconexão para propagar uma saída gerada por sua lógica embutida, e o bloco destino, que faz uso desse sinal como estímulo em uma ou mais entradas dos seus registradores. O intervalo em que o bloco fonte se aproveita da interconexão é determinado pelo tempo de propagação do caminho mais longo que o sinal percorre desde a fonte até o destino e, durante esse período, os elementos de memorização que irão receber esse sinal não devem ser habilitados a registrar pois o resultado seria instável e, por conseguinte, imprevisível. Fundamentalmente, esse é o conceito básico do funcionamento de um circuito síncrono e o motivo pelo qual um circuito regido por uma fonte única de temporização deve operar à freqüência relativa ao tempo de propagação mais longo entre dois registradores.

No entanto, a metodologia de conversão proposta gera um grande inconveniente: a interdependência serial da transmissão de dados em uma cadeia de registradores. A Figura 35 mostra passo a passo como o fluxo de dados é executado em uma seqüência de 3 registradores, sendo o nó 2 o de maior interesse para essa análise pois está inserido entre dois outros. A implementação da técnica ASERT, conectando cada registrador a um controlador de nó, obriga que dois nós vizinhos nunca possam consumir dados de seu predecessor simultaneamente, obrigando a transferência a ter sempre uma lacuna de operação presente na estrutura. No entanto, essa limitação pode ser evitada inserindo um controlador de nó de interconexão que possui um intervalo de operação equivalente ao tempo mínimo permitido pela tecnologia de células padrão em sua entrada de *clock*, mantendo os tempos de operação dos controladores de nós intocados. A Figura 45 mostra como a nova implementação do simples exemplo de uma cadeia de registradores se comporta quando os controladores de nó de interconexão *Nó X1* e *Nó X2* são incluídos. Apesar de possuir um número de transições maior do que o apresentado na Figura 35, o tempo de execução necessário para que o *dado4* atinja *S4* é praticamente 2⁄3 do valor original, supondo que o circuito opere à freqüência de 100MHz numa tecnologia CMOS de 0,13µm, onde o tempo de *setup* de um registrador é aproximadamente 200ps:

Δ_N = 10ns, tempo de execução de um nó;

Δ_R = 0,2ns, tempo de *setup* do registrador;

$\Delta_S = 9 \times \Delta_N$ = 90ns, tempo total no modo original serial

$\Delta_P = 5 \times \Delta_N + 6 \times \Delta_R$ = 61,2ns, tempo total no modo paralelo

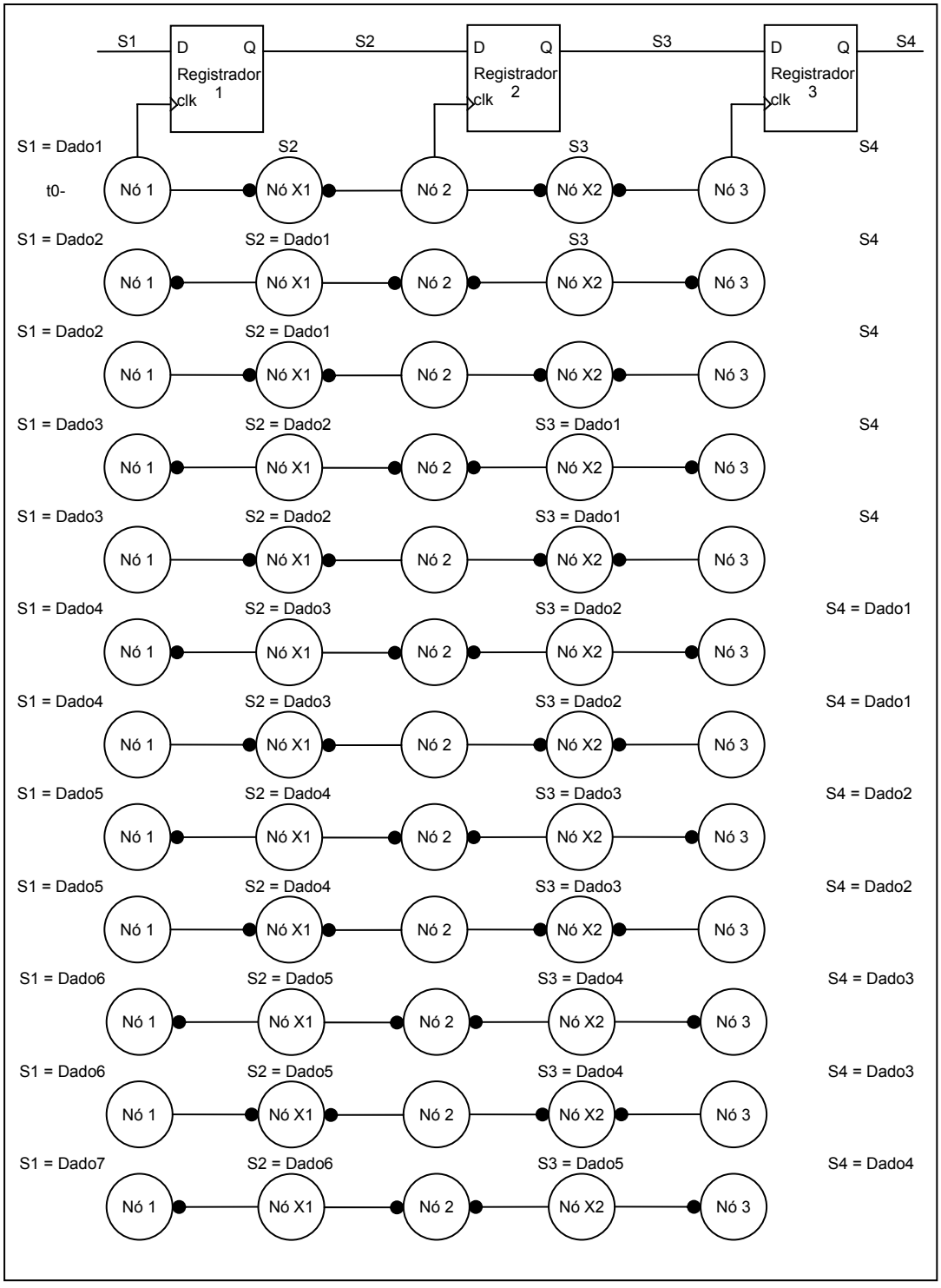


Figura 45: Inserção de um controlador de nó para interconexão

Ao inserir os controladores de nós para interconexão, o diagrama ASERT do circuito da Figura 38 é alterado como mostra a Figura 46, assim como o algoritmo original é transformado como apresentado na Figura 47.

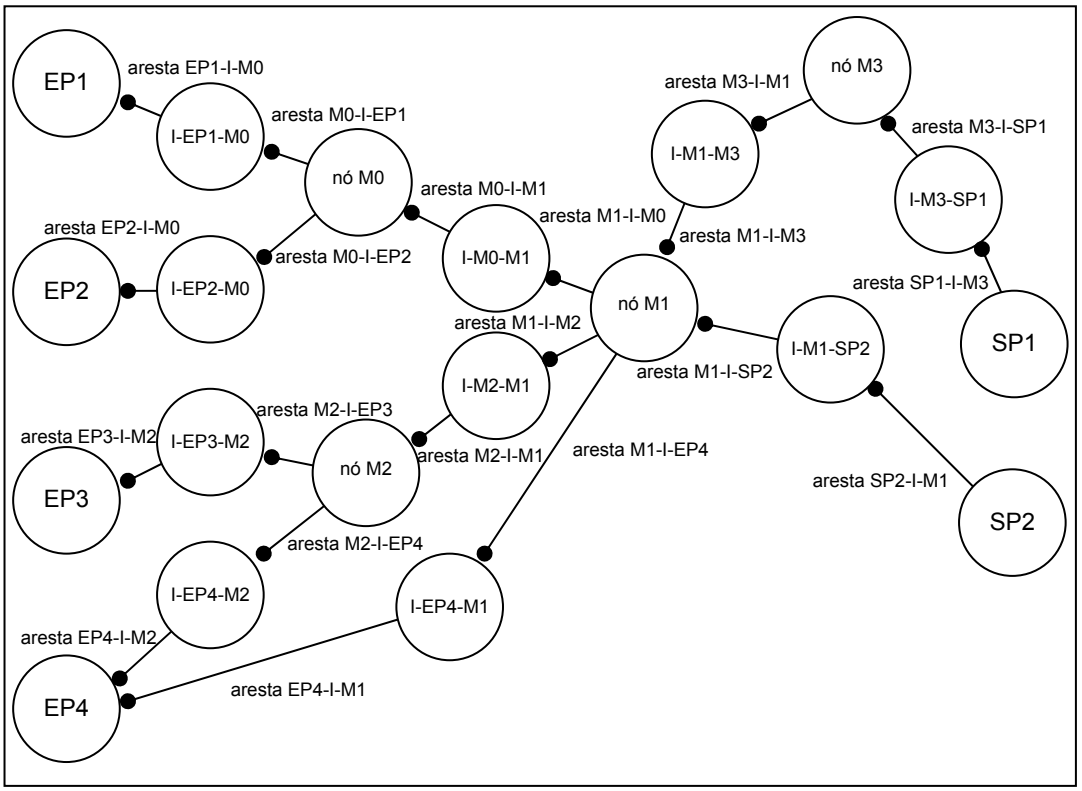


Figura 46: Diagrama ASERT incluindo controladores de nós de interconexão da Figura 38

A técnica de inserção de controladores de nós em interconexões aumenta a vazão de dados na propagação de cadeias de registradores, visto que cada estágio dos elementos de memória está sempre carregado de informação gerado no estágio anterior, sem a necessidade da intercalação de bolhas devido à sinalização assíncrona de recursos mutuamente exclusivos.

Apesar da otimização apresentar o ganho demonstrado anteriormente, existe a possibilidade de uma situação particular causar a perda de dados na transferência entre dois registradores caso a metodologia seja aplicada. O exemplo da Figura 45 apresentou uma seqüência de registradores ligados serialmente, cujos atrasos entre suas interconexões foram considerados idênticos, o que não é uma visão realista. Os atrasos não possuem nenhuma correlação entre si, podendo ser tanto maiores quanto menores que as interconexões vizinhas. Na circunstância em que os atrasos de duas interconexões adjacentes forem decrescentes em relação ao fluxo do caminho de dados,

a metodologia de otimização pode ser aplicada sem que ocorra nenhum risco à integridade dos dados. Entretanto, caso os atrasos dos nós vizinhos se apresentarem de forma crescente em relação ao fluxo de dados, é possível que ocorra uma perda de informação na transmissão dos mesmos.

```
loop módulo_fonte em <todos os módulos do sistema>
  se módulo_fonte possui <saída registrada>
    se módulo_fonte não associado nó_fonte
      cria_nó nó_fonte
    fim_se
    loop módulo_destino em <todos os módulos do sistema>
      se módulo_fonte conecta módulo_destino
        se módulo_destino não associado nó_destino
          cria_nó nó_destino
        fim_se
        se não existe nó_inter_destino
          cria_nó nó_inter_destino
          cria_aresta nó_inter_destino nó_destino
        fim_se
        cria_aresta nó_fonte nó_inter_destino
      fim_se
    fim_loop
  fim_se
fim_loop
```

Figura 47: Algoritmo para geração de nós e arestas original com controladores de nós para interconexão

A questão levantada é semelhante à situação do emprego de um circuito síncrono que contém um caminho crítico que não é respeitado pela freqüência do *clock* aplicado. Assim que a combinação de controle do caminho de dados habilitar o fluxo através do pior caso, o registrador, que iria memorizar o valor provindo desse caminho, o realiza antes que o valor correto esteja presente na entrada do mesmo, ocasionando a retenção errônea de dado e a perda do verdadeiro.

Um programa foi desenvolvido com a finalidade de facilitar a análise de variações de controladores e seus respectivos tempos de operação, e sua saída é mostrada no Apêndice 7.1. A Figura 48 e a Figura 49 apresentam a evolução temporal

de um simples diagrama ASERT cujo fluxo de dados acompanha nós com tempos de operação crescentes e decrescentes ao longo da cadeia de registradores. Todos os dados inseridos em *Reg A* são transportados corretamente até *D*. A Figura 50 representa o diagrama resultante da aplicação da otimização temporal por completo, cujos nós de interconexões, denominados *I.<nome do nó>*, geram o fenômeno similar ao caso do circuito síncrono com violações no caminho crítico. O fluxo de dados é normal de *T=0ns* e *T=1ns*, quando *Reg A* controlado pelo nó *A* recebe o sinal de habilitação, retendo o dado *azul* enquanto um novo dado *verde* já está disponível na entrada de *Reg A*. No entanto, como o nó *B* tem seu tempo de propagação igual a 6ns, contra 4ns do nó *A*, e a otimização temporal permite a operação simultânea de 2 nós adjacentes, no instante *T=5ns* a entrada de *Reg A* é habilitada a receber um novo valor antes que o nó *B* fosse capaz de reter o dado *azul* anterior. Nesse instante o registrador controlado pelo nó *A* retém o dado a*marelo* presente em sua entrada, descartando o dado *azul* antes mesmo que *Reg B* tenha a chance de armazená-lo, perdendo-o definitivamente.

A solução para esse inconveniente requer que a metodologia averigüe todas as possíveis combinações entre nós destinos do fluxo de dados e os nós geradores de dados a fim de identificar os controladores candidatos à falhas de transferência de dados, e insira um controlador de aresta como atalho entre os dois nós problemáticos. O princípio de execução mutuamente exclusiva, característico de uma aresta SER, se encaixa com precisão nessa situação, obrigando os dois nós em questão a serem ativados alternadamente, assegurando que tal situação não tenha a possibilidade de ocorrer. Outra possível solução seria remover por completo o controlador de nó de interconexão, aplicando uma técnica híbrida, onde a otimização temporal seria aplicada somente entre pares de nós com atrasos decrescentes. No entanto, os exemplos não revelam detalhes de situações reais, onde um nó destino pode manter relações com inúmeros nós fonte, e a implementação híbrida descartaria a possibilidade de paralelismo de execuções entre os nós que não apresentam atrasos crescentes em seu caminho de dados.

Ao executar o procedimento de seleção, o diagrama original da Figura 48 deve ter sua topologia de nós e arestas otimizada conforme a apresentada na Figura 52 e na Figura 53. Ao inserir a aresta curva entre os nós *A* e *B*, as escritas em seus respectivos registradores passam a ser coordenadas e nenhum dos dados fornecidos na entrada primária é perdido. O algoritmo que descreve a inserção de nós é alterado como mostra a Figura 51.

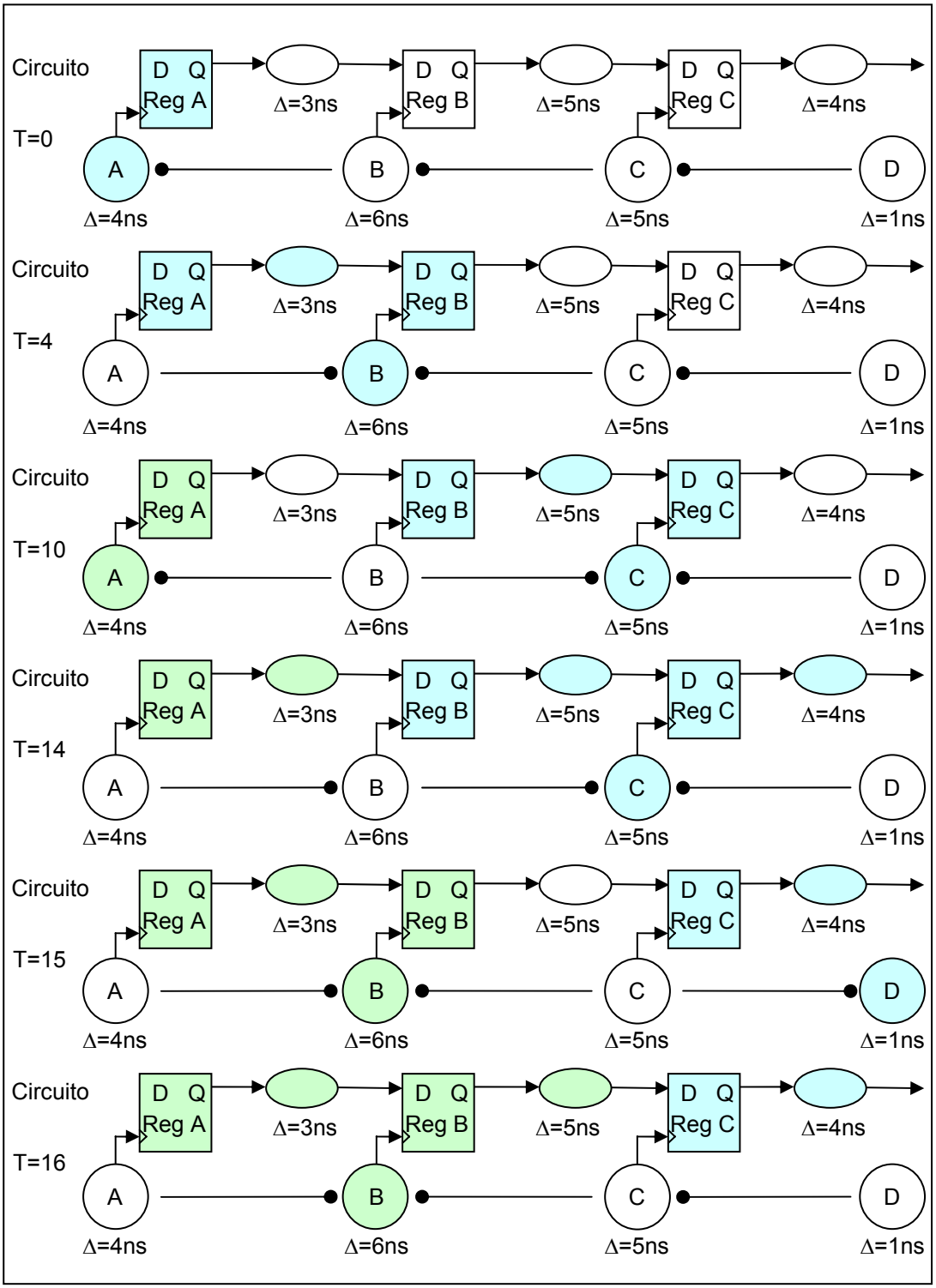


Figura 48: Diagrama ASERT com tempos de operações crescentes e decrescentes (1ª parte)

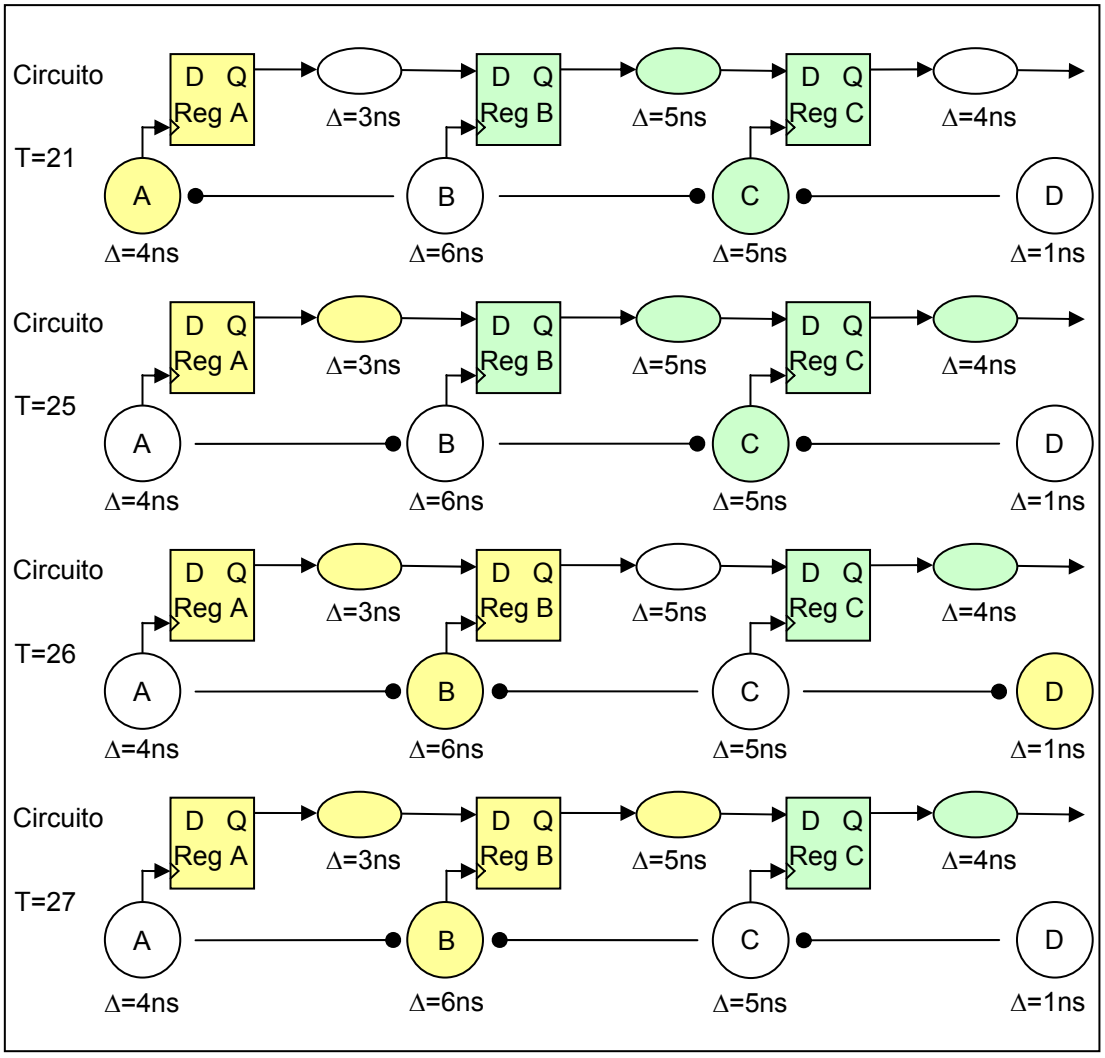


Figura 49: Diagrama ASERT com tempos de operações crescentes e decrescentes (2ª parte)

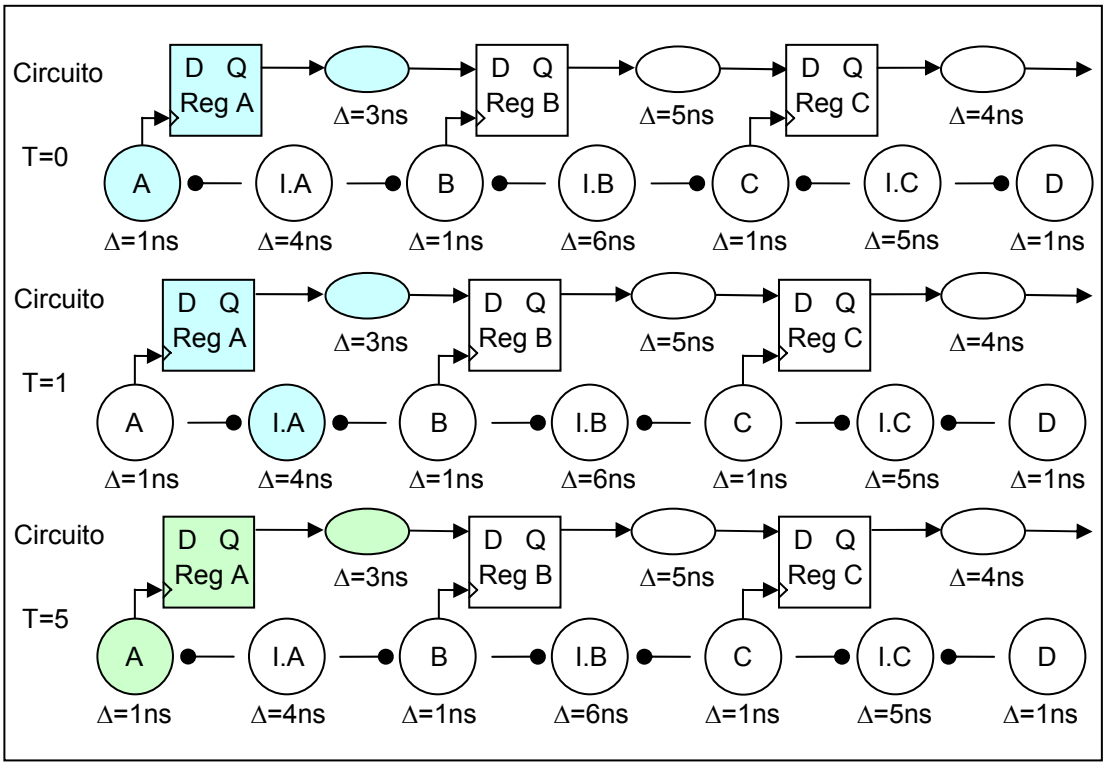


Figura 50: Otimização temporal aplicada ao exemplo anterior

```
loop módulo_fonte em <todos os módulos do sistema>
  se módulo_fonte possui <saída registrada>
    se módulo_fonte não associado nó_fonte
      cria_nó nó_fonte
    fim_se
    loop módulo_destino em <todos os módulos do sistema>
      se módulo_fonte conecta módulo_destino
        se módulo_destino não associado nó_destino
          cria_nó nó_destino
        fim_se
        se não existe nó_inter_destino
          cria_nó nó_inter_destino
          cria_aresta nó_inter_destino nó_destino
        fim_se
        se tempo_destino > tempo_fonte
          cria_aresta nó_fonte nó_destino
        fim_se
        cria_aresta nó_fonte nó_inter_destino
      fim_se
    fim_loop
  fim_se
fim_loop
```

Figura 51: Algoritmo para geração de nós e arestas original com controladores de nós para interconexão e arestas em atrasos crescentes

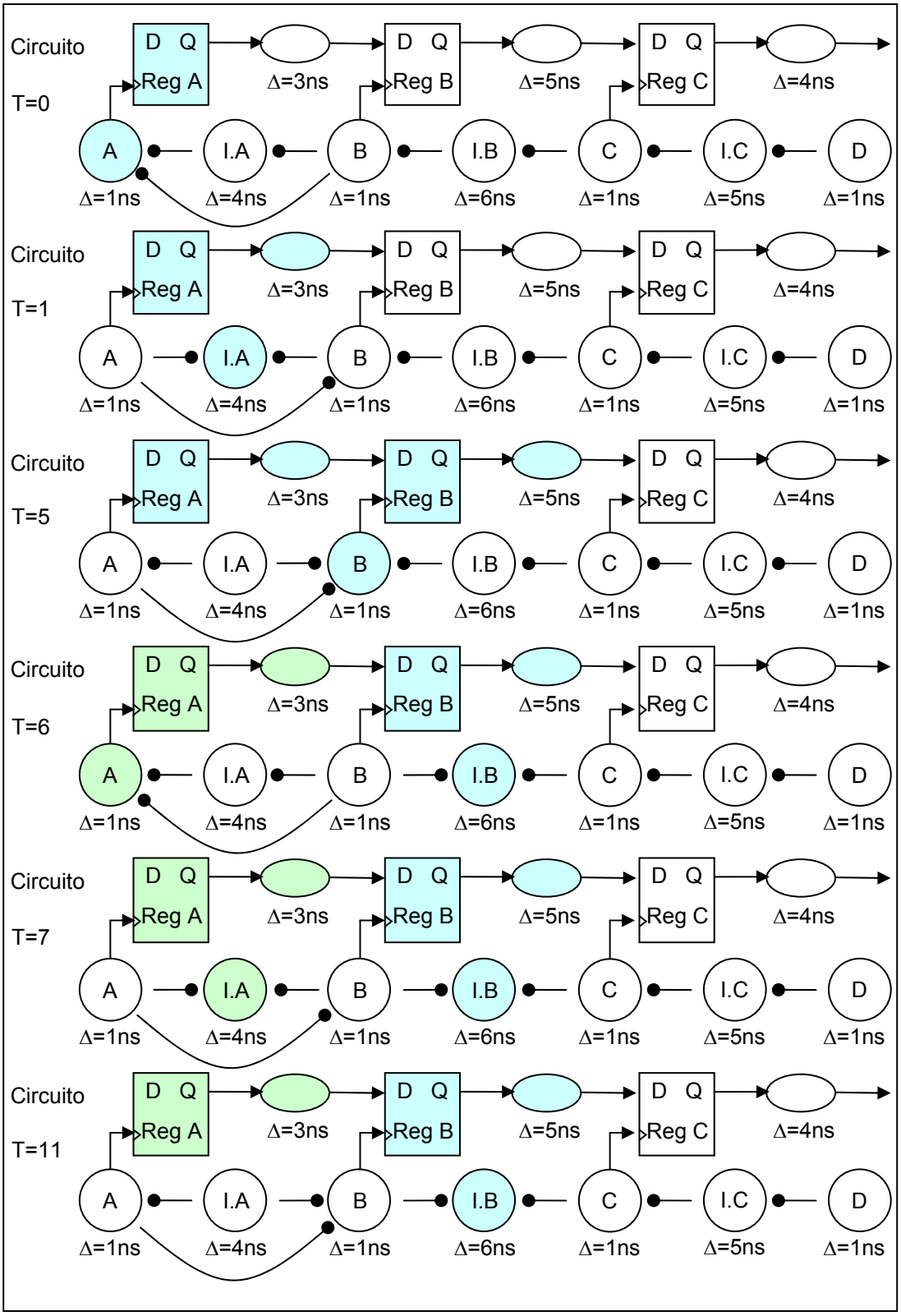


Figura 52: Diagrama ASERT com tempos de operações crescentes e decrescentes (1ª parte)

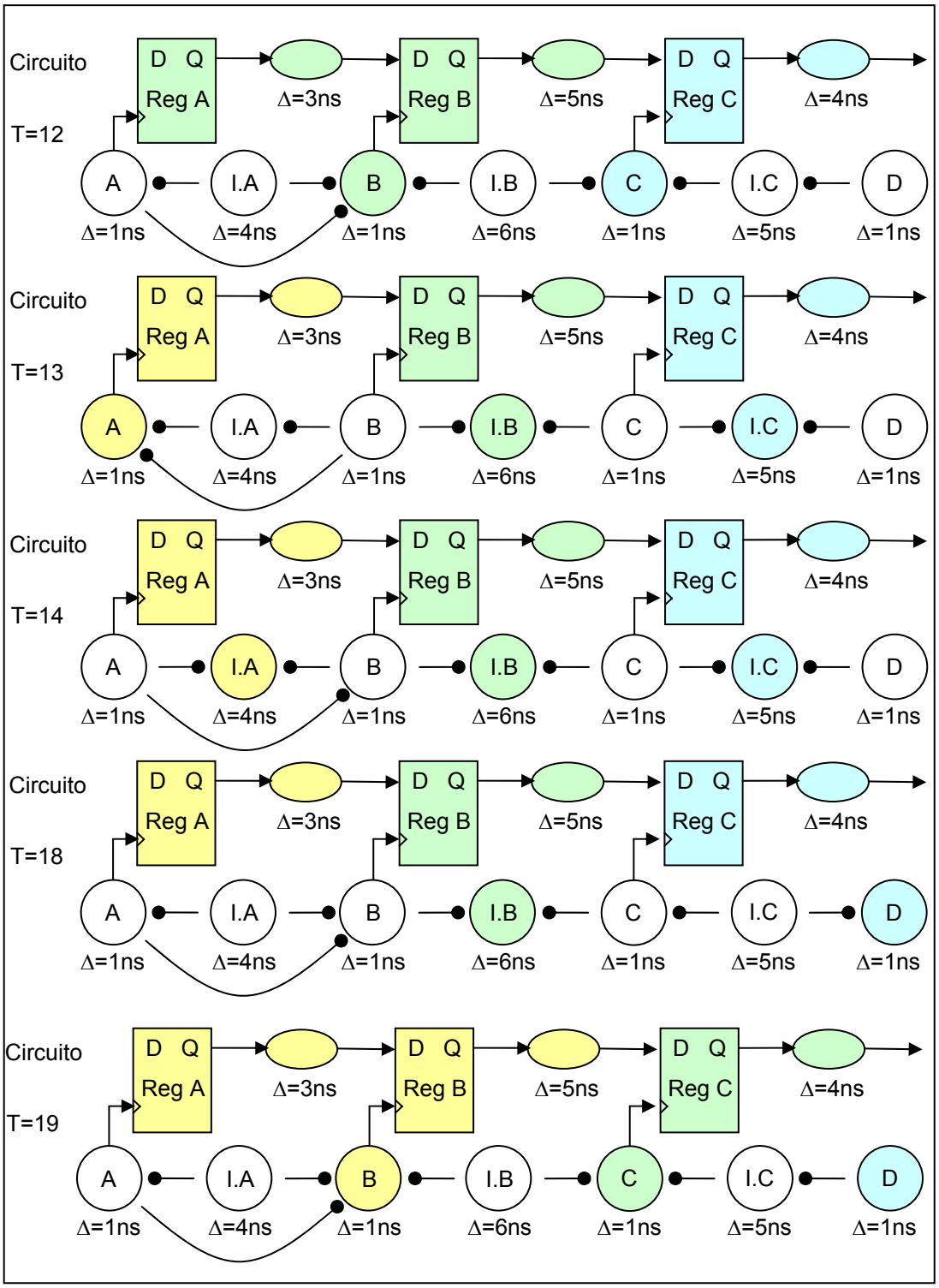


Figura 53: Otimização temporal evitando nós de interconexão entre nós com tempos de operações crescentes (2ª parte)

O afunilamento do fluxo de dados causado pelo módulo *M1* sem a aplicação da otimização espacial, presente na Figura 46, é resolvido particionando o controle da sinalização assíncrona das respectivas saídas, ao combinar a aplicação dos métodos de otimização espacial e de otimização temporal, culminando no diagrama ASERT

completo do circuito da Figura 38 apresentado na Figura 54 e no algoritmo da metodologia revisado na Figura 55.

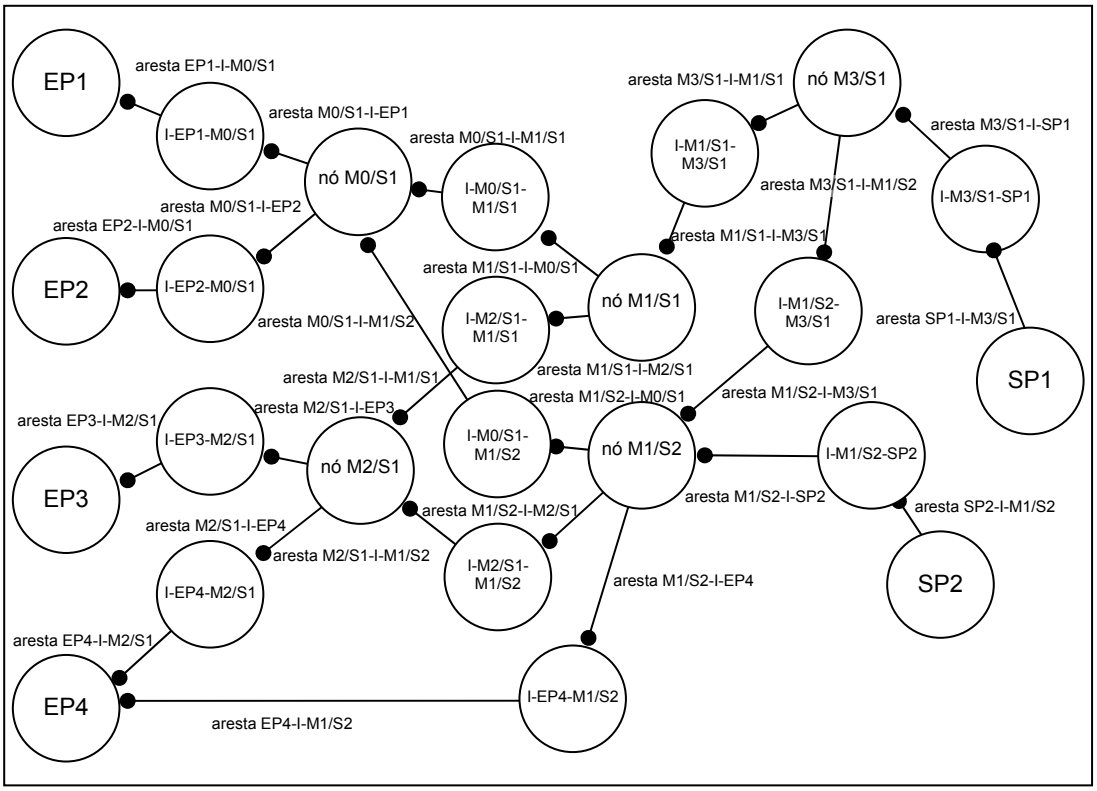


Figura 54: Aplicação de otimizações espacial e temporal no exemplo da Figura 38

```
loop módulo_fonte em <todos os módulos do sistema>
 loop saída_fonte em <todas as saídas do módulo_fonte>
  se saída_fonte registrada
   se saída_fonte não associado nó_fonte
    cria_nó nó_fonte
   fim_se
   loop módulo_destino em <todos os módulos do sistema>
    se módulo_fonte conecta módulo_destino
     loop saída_destino em <todas as saídas do módulo_destino>
      se saída_fonte conecta saída_destino
       se saída_destino não associado nó_destino
        cria_nó nó_destino
       fim_se
       se não existe nó_inter_destino
        cria_nó nó_inter_destino
        cria_aresta nó_inter_destino nó_destino
       fim_se
       se tempo_destino > tempo_fonte
        cria_aresta nó_fonte nó_destino
       fim_se
       cria_aresta nó_fonte nó_inter_destino
       cria_aresta nó_fonte nó_destino
      fim_se
     fim_loop
    fim_se
   fim_loop
```

Figura 55: Algoritmo da metodologia de geração de nós e arestas aplicando otimizações espacial e temporal

3.4 Geração do sinal de fim de operação

A viabilidade do emprego de qualquer método de sinalização assíncrona se baseia substancialmente na disponibilidade da comunicação do término de operação das unidades funcionais a serem sincronizadas, que podem ser compostas, desde a forma mais simples, através de uma única porta lógica ou até por inúmeras delas, constituindo complexos circuitos inseridos em sistemas maiores ainda. No entanto, as ferramentas de CAD e bibliotecas de células básicas para desenvolvimento de circuitos lógicos digitais

usualmente disponíveis no mercado são voltadas para o projeto de circuitos síncronos e por isso não incorporam a funcionalidade de indicação de fim de operação em nenhuma de suas estruturas, sendo portanto necessário conceber um modo secundário de sinalização junto à metodologia de conversão.

A Seção 3.3 apresentou as técnicas desenvolvidas de particionamento, otimizações e associações entre o circuito original e controladores de nós e arestas utilizadas na metodologia de conversão proposta. A Seção 3.3.2 definiu o controlador de nó de interconexão o responsável por gerenciar o tempo necessário para a propagação do caminho mais longo entre os dois nós que o mesmo interconecta. O tempo de operação associado ao controlador de nó de interconexão deve ser maior que o atraso consumido, através do caminho mais longo, entre o último elemento de retenção de dados antes da saída associada ao controlador de nó gerador de dados e o primeiro elemento de memória habilitado pelo controlador de nó receptor. Esse procedimento é realizado com o auxílio de programas de CAD por possuírem algoritmos especialmente otimizados para esse fim, além das caracterizações necessárias das células padrão extraídas de modelos fornecidos pelos projetistas das bibliotecas em conjunto com os manufaturadores das pastilhas do circuito integrado.

A Figura 56 exemplifica o texto mostrando que:

$\Delta_{interAB} = \Delta_{AS1\text{-}3} + \Delta_{AB} + \Delta_{BS1\text{-}1}$, onde

$\Delta_{interAB}$ é período de operação do controlador de nó da interconexão *MA/S1* e *MB/S1*,

$\Delta_{AS1\text{-}3}$ é o tempo necessário para percorrer a lógica combinacional entre a saída *MA/Registrador2/Q* e a saída *MA/S1*,

Δ_{AB} é o tempo necessário para percorrer a lógica combinacional entre a saída *MA/S1* e a entrada *MB/E1* e

$\Delta_{BS1\text{-}1}$ é o tempo necessário para percorrer a lógica combinacional entre a entrada *MB/E1* e a entrada *MB/Registrador1/D*.

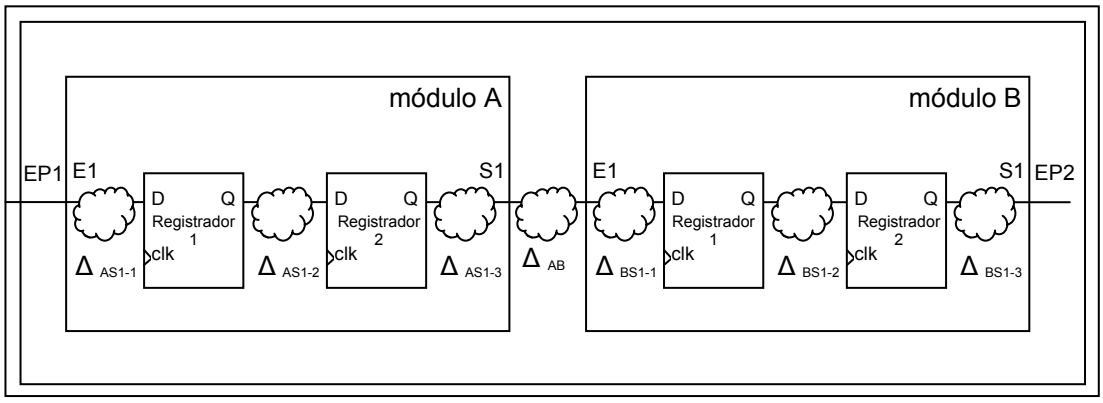


Figura 56: Método de extração do tempo de operação do controlador de nó de interconexão

Nessa seção serão introduzidas duas técnicas distintas para implementação dos atrasos requeridos para o funcionamento da sinalização assíncrona.

3.4.1 Atraso Casado

Atraso casado (*matched delay*) é uma técnica do tipo de atraso fixo que consiste em construir uma estrutura paralela inócua com um atraso igual ou maior que o caminho crítico a ter o seu fim de operação sinalizado. A estrutura paralela é responsável por atrasar um sinal aplicado a sua entrada, o de início de operação, de forma que o sinal de fim de operação seja ativado adequadamente após a saída do circuito principal estar estável. Na prática, como mencionado, ferramentas de CAD auxiliam na tarefa de identificar o tempo de propagação do caminho crítico entre os dois registradores apontados pelos algoritmos de particionamento e otimizações da metodologia de conversão, para cada um dos controladores de nó de interconexão. A estrutura de atraso casado é constituída por uma cadeia de células de retardo (*delay cell*) conectadas serialmente, cujo número de elementos é determinado pela relação do tempo de propagação intrínseco às células, baseado nas referências que descrevem a tecnologia utilizada, e o caminho crítico do circuito alvo:

$$N_{c\text{帻}ulas} \geq \frac{\Delta_{cr\text{癤}ico}}{\Delta_{c\text{帻}ula}} \text{ onde,}$$

$N_{células}$ é o número de células que compõem a cadeia de retardo,

$\Delta_{crítico}$ é o atraso do caminho crítico do circuito alvo e

$\Delta_{célula}$ é tempo de propagação intrínseco de uma célula de retardo.

A Figura 57 mostra um exemplo de implementação da técnica de atraso casado enquanto a Tabela 3 resume o número e categorias de portas lógicas que existem entre cada entrada *M/E[1-6]* e a saída *M/S1*, onde:

Δ_{AND} é o atraso de propagação de uma porta lógica *AND***,**

Δ_{OR} é o atraso de uma porta lógica *OU* e

Δ_{RET} é o atraso o de uma célula de retardo.

Tabela 3: Resumo do número de portas entre as entradas e a saída do exemplo da Figura 57

Entrada	Número de portas *AND*	Número de portas *OR*
E1	2	1
E2	2	1
E3	1	1
E4	0	1
E5	0	3
E6	0	3

Como $\Delta_{OR} > \Delta_{AND}$, o caminho crítico do módulo *M* será resultante da entrada de dados pelas entradas *M/E5* e/ou *M/E6*, percorrendo através das portas lógicas *u2*, *u4* e *u5*, e terminando na saída *M/S1*, totalizando uma propagação de $3\times \Delta_{OR}$. Uma vez que $\Delta_{OR} = 5/3\ \Delta_{RET}$, o número mínimo de células de retardo necessário para que o sinal de *fim de operação* seja coerente com o tempo de propagação do caminho crítico do circuito é $3\times 5/3\ \Delta_{RET} = 5 \times \Delta_{RET}$, como representado na Figura 57.

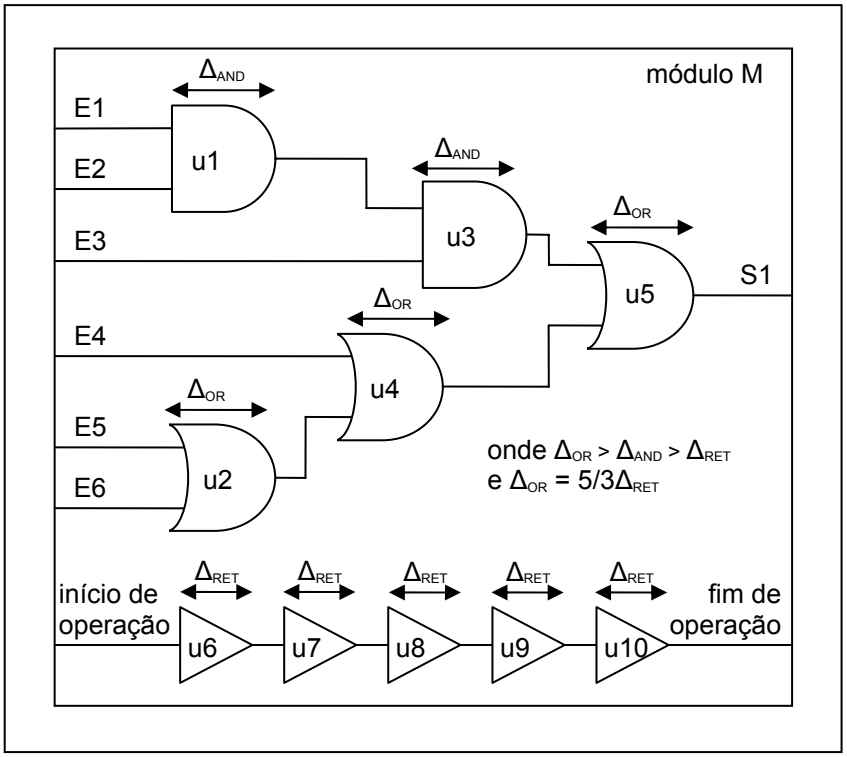


Figura 57: Sinal de fim de operação gerado pela técnica de atraso casado

A introdução da cadeia de atraso casado possibilita o módulo M receber o sinal de início de operação provindo de seu controlador de nó, assim que um novo conjunto de entradas esteja disponível, e a comunicar aos seus nós vizinhos o fim de operação. Neste momento, o dado gerado está válido e pronto para ser consumido da sua saída *S1*, ao mesmo tempo em que suas entradas estão disponíveis para receber um novo conjunto de dados para processamento.

3.4.2 *Diet Clock*

A grande desvantagem da utilização no método de atraso casado para a geração do sinal de fim de operação de um bloco numa implementação assíncrona é a possibilidade de variação das características do comportamento físico real. As características estimadas, fornecidas através de documentos e modelos pelo manufaturador de circuitos integrados, nem sempre são corretamente refletidas no produto final. As diferenças podem ocorrer devido a desvios no processo de manufatura ou até no próprio projeto das máscaras do circuito a ser implementado. Os efeitos dessa possível variação podem influenciar o caminho crítico do circuito original, usado como base para a implementação da cadeia de células de retardo geradora do sinal de fim de operação, de

forma a ter um tempo de propagação diferente em relação ao previsto na fase de projeto. Caso o caminho crítico seja alterado provocando um intervalo de propagação menor que o esperado, o sinal de fim de operação ainda será mais longo que o mesmo, mantendo ainda a execução coerente da sua função original. No entanto, se o caminho crítico for alterado tornando-se mais longo que a cadeia de retardo, então o circuito inteiro está comprometido visto que não será possível garantir que a saída calculada pelo circuito original estará estável no instante em for sinalizado o fim de operação deste mesmo bloco. No caso dos circuitos síncronos, esse problema é quase sempre contornável já que a freqüência do sinal sincronizador pode ser ajustada até o ponto em que o circuito inteiro consiga se propagar da forma esperada.

Outro fator que denigre o emprego do método de atraso casado é o fato de se basear em células de retardo para compor o tempo necessário para ultrapassar o intervalo entre o instante do início da operação e o final da mesma. Levando em consideração que um circuito típico utilizando tecnologia CMOS de 0,13μm tem um *clock* com um período de operação em torno de 10ns, e o atraso intrínseco de uma célula de atraso (DLY4X1) é de aproximadamente 0,36ns, seriam necessárias 28 células de retardo para atingir o intervalo requerido para tal caminho, totalizando uma área de $13{,}6 \times 28 = 380\mu m^2$. Como referência, um registrador com entrada de *reset* (`DFFRX1`) possui uma área de $32{,}3\mu m^2$.

O *diet clock* é um procedimento alternativo de geração do sinal de fim de operação proposto como substituto do método de atraso casado. Nele é empregado um sinal temporal de referência, o *clock*, aplicado a contadores de valores fixos que produzem intervalos pré-programados correspondentes aos tempos de duração de cada nó de interconexão estimados no momento da aplicação dos algoritmos de conversão síncrona-assíncrona. Cada contador associado a um controlador de nó possui um sinal de habilitação da contagem diretamente conectado ao sinal de início de operação do nó. Dessa forma, o contador se encontra em estado estacionário até que seja requisitado o começo da temporização.

O conjunto de portas lógicas que constituem os circuitos contadores responsáveis pela geração dos sinais de fim de operação deve ser contido em uma região física devidamente concentrada para que a árvore de distribuição de *clock* seja praticamente irrelevante se comparada ao restante das células utilizadas pelo método.

A Figura 58 apresenta um exemplo de aplicação do *diet clock* em um sistema com 4 módulos conectados serialmente, aplicando o método de associação de nós de

interconexão como regulador do tempo de propagação do caminho entre o último elemento de memória de um módulo e o primeiro do módulo seguinte. A Figura 59 mostra os atrasos necessários para cada dado enviado através da sua respectiva interconexão e quantas unidades do período do *clock* são necessárias de forma a gerar o sinal de fim de operação apropriado.

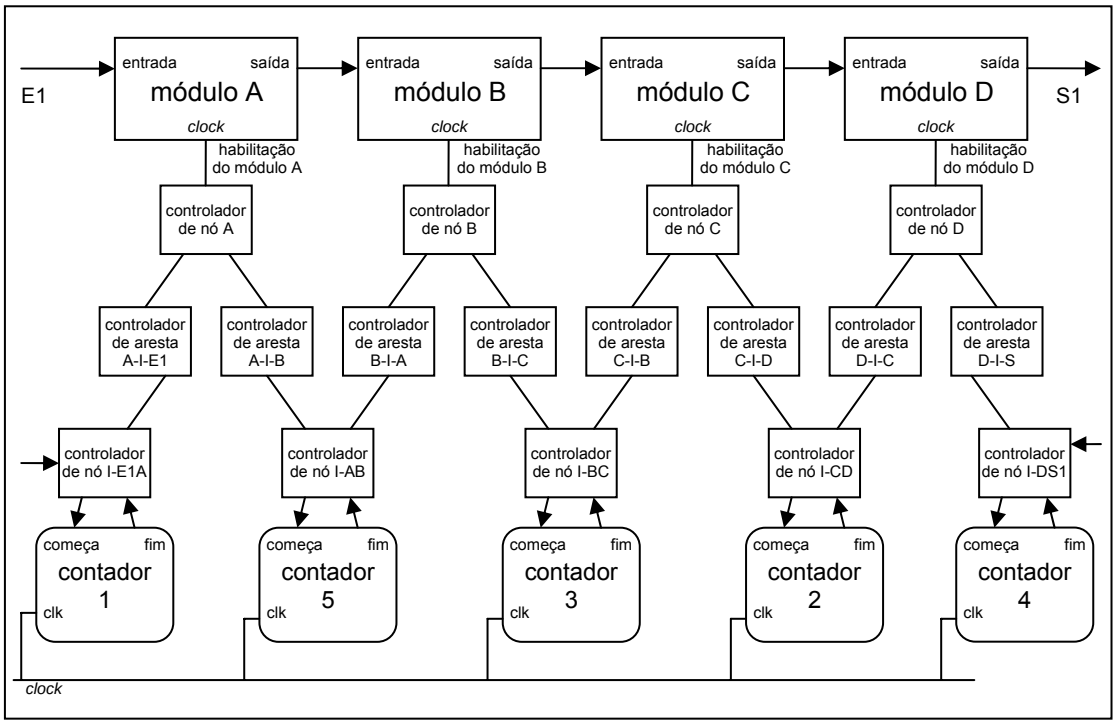


Figura 58: Aplicação do diet *clock* como temporizador de controladores de nós

Ao utilizar o método do *diet clock*, garante-se que mesmo em caso de variação do processo de manufatura da pastilha do circuito integrado há a possibilidade de ajustar o *clock* de referência para que os atrasos relativos às interconexões sejam devidamente calibrados.

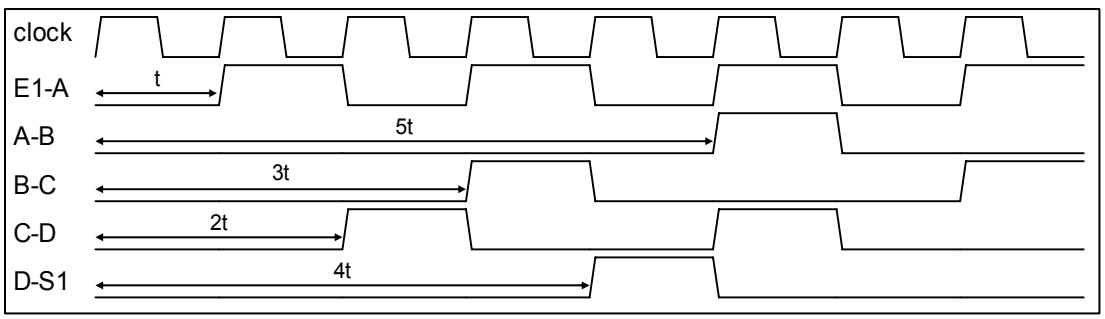


Figura 59: Características dos controladores de nós de interconexão

O método de atraso casado e o método de *diet clock* são comparados em área e dissipação de potência com um tempo de operação programado para 8ns, utilizando uma biblioteca de células básicas da tecnologia CMOS genérica 0,13µm.

A metodologia de comparação aplicada, cujo fluxograma está representado na Figura 60, teve início na criação de uma descrição em verilog para cada um dos métodos apresentados. A técnica do *diet clock* foi baseada em um incrementador síncrono de 3 bits com período de operação de 1ns. Já a descrição do circuito de atraso casado foi baseada na instanciação do devido número de células de retardo manualmente, nesse caso 22 células, uma vez que atrasos temporais em descrições comportamentais de circuitos não são estruturas capazes de ser utilizadas por ferramentas sintetizadoras de circuitos em portas lógicas.

Ambos os códigos foram fornecidos a uma ferramenta sintetizadora para que fossem processados de forma a produzir uma descrição equivalente em portas lógicas, assim como a extrair informações de atraso inseridas pelo modelo de carga de fios e os próprios atrasos intrínsecos às portas lógicas. O circuito resultante do processamento para o método de *diet clock* está apresentado na Figura 61, enquanto a Figura 62 mostra o resultado do atraso casado.

O arquivo com a descrição em verilog do circuito em portas lógicas combinado ao arquivo contendo as informações de tempo de propagação através de interconexões e intrínsecos no formato *sdf* (*standard delay file*) foram compilados juntamente com um ambiente para teste em um simulador lógico. Esse ambiente gera um sinal de início de operação a cada 8ns e recebe um sinal de fim de operação aproximadamente 8ns após. A Figura 61 apresenta as formas de onda da simulação de implementação do *diet clock*. O circuito incrementador é habilitado assim que o sinal de início de operação é comunicado, ativando o sinal de fim de operação ao atingir o final da contagem, 8,4ns após o início, como indicado através das duas barras verticais.

A Figura 62 mostra as formas de onda da simulação em portas lógicas com o método de atraso casado. Diferentemente do circuito implementado pelo método de *diet clock*, esse circuito não é realmente habilitado a operar; o sinal na entrada é simplesmente atrasado em relação ao momento que ele é aplicado a uma célula de retardo até o momento em que a deixa. Os sinais *S0* a *S20* são nós intermediários que interligam as células de retardo serialmente, provocando o efeito de onda de propagação

do mesmo sinal aplicado à primeira das células, e sinalizando o fim de operação após 8,3ns.

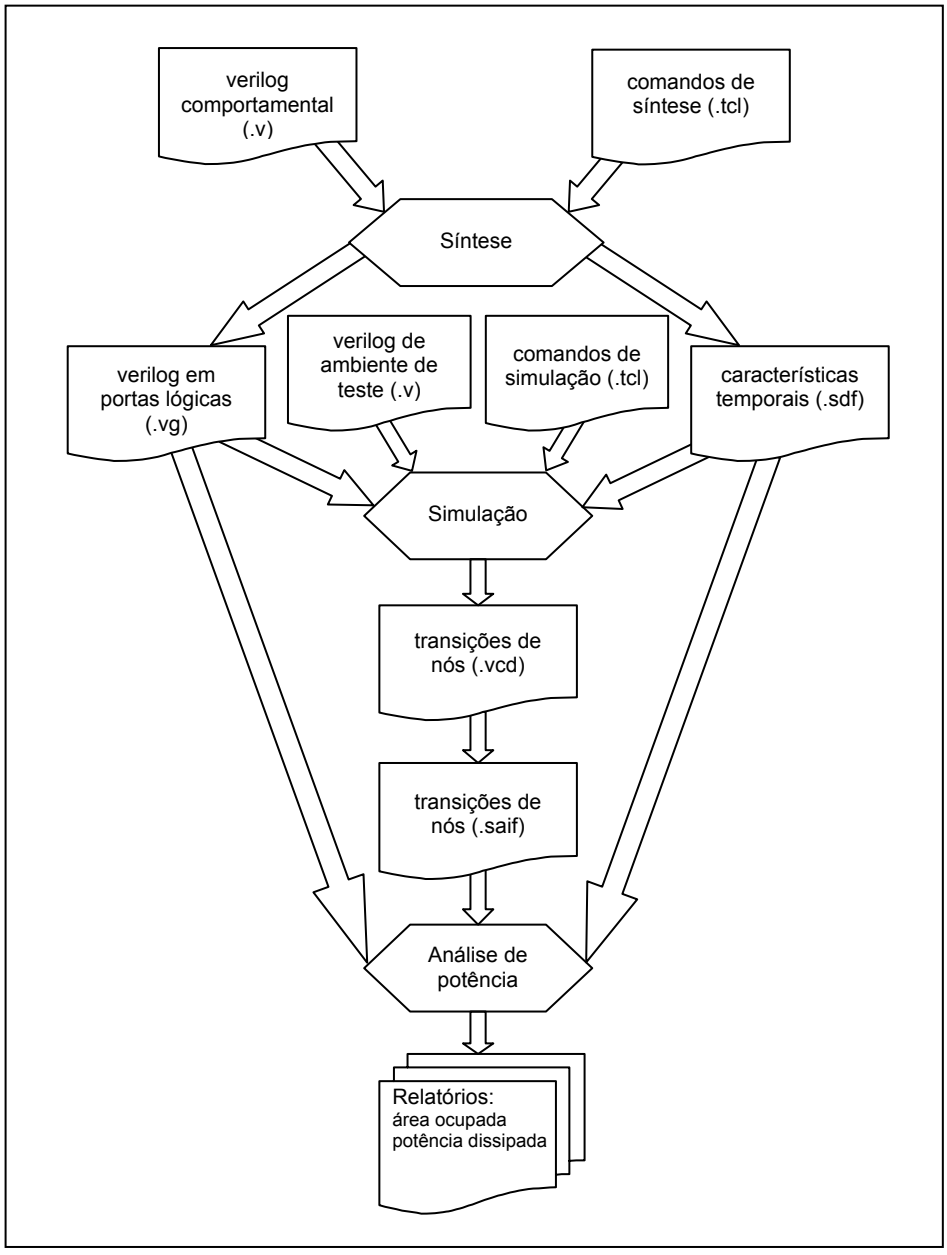


Figura 60: Fluxograma da metodologia de comparação entre os métodos de atraso casado e diet *clock*

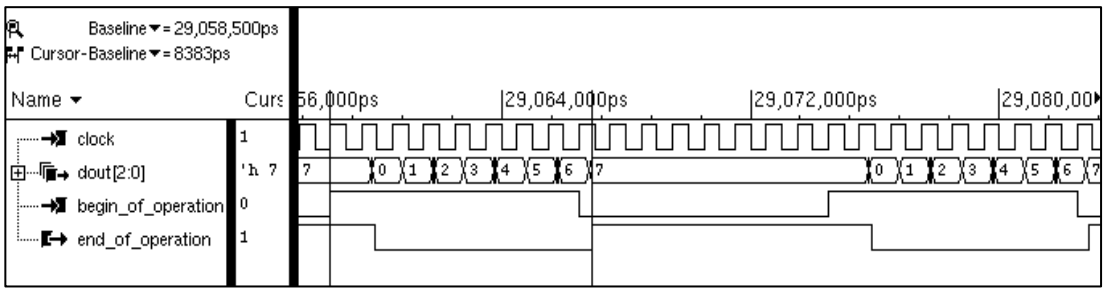


Figura 61: Forma de onda da simulação por portas lógicas do diet *clock*

Durante a execução da simulação, a ferramenta foi instruída a registrar as transições de todos os fios existentes no circuito em um arquivo no formato *vcd* (*value change dump*) a fim de possibilitar pós-processamento e análise dos resultados. Os dados foram reaplicados à ferramenta de síntese, juntamente com a descrição em portas lógicas do circuito combinada aos valores de atraso de propagação do mesmo que possibilitaram o cálculo da área consumida e da potência dissipada por cada um dos métodos.

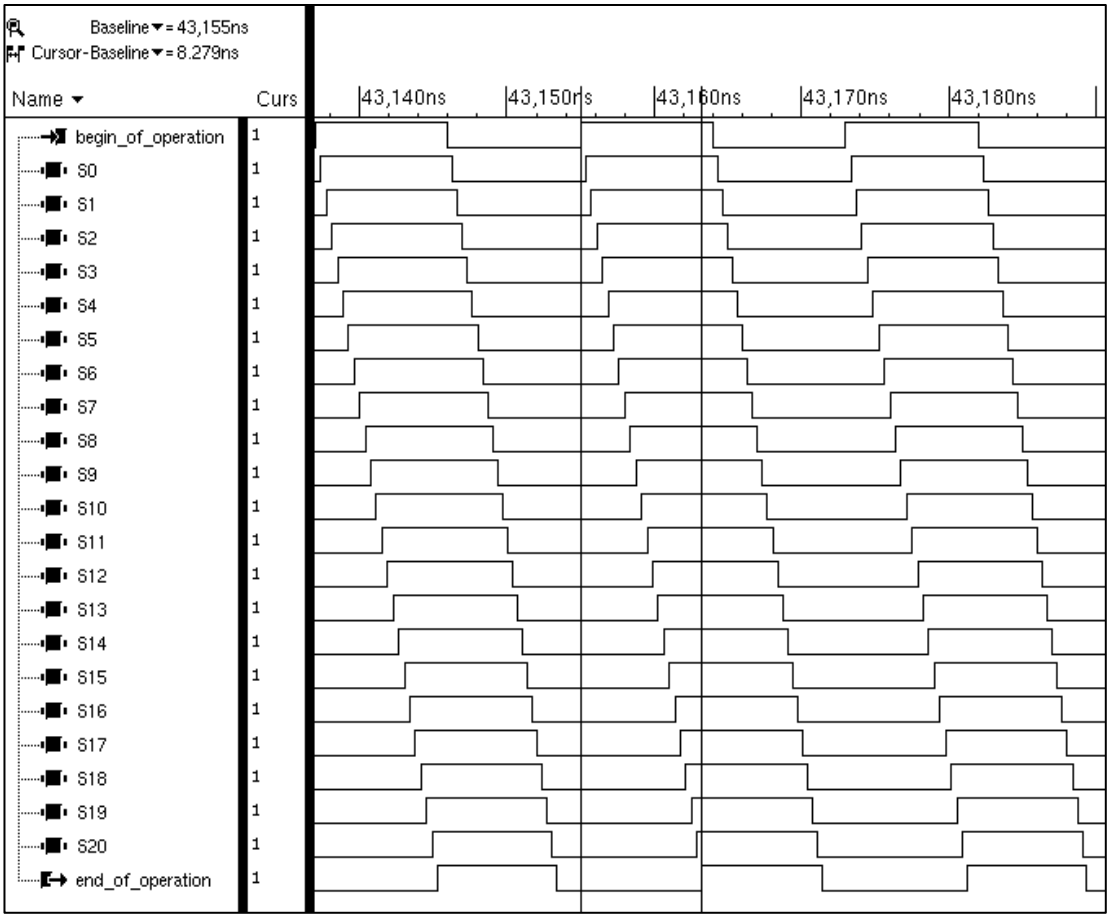


Figura 62: Forma de onda da simulação em portas lógicas da implementação de atraso casado

A Tabela 4 resume a distribuição da área ocupada reportada pela última etapa da comparação. As células de retardo são células de grande área, comparadas por exemplo com a área de um registrador do tipo que o sintetizador empregou na implementação do incrementador. Apesar do método de atraso casado ter área não combinacional nula, a área ocupada por suas 22 células combinacionais é praticamente o dobro do total consumido pelo método do *diet clock*.

Tabela 4: Áreas consumidas por ambos os métodos estudados

	Quantidade				**Área**		
Método	Portas lógicas	Nós	Células	Referências	Combinacional (μm^2)	Não combinacional (μm^2)	Total (μm^2)
Atraso casado	2	23	22	1	298,7	0	298,7
Diet clock	7	17	11	6	71,3	96,8	168.1

As informações das transições de cada nó do circuito geradas através da simulação compensada com atrasos de propagação permitem que a ferramenta de síntese seja capaz de analisar e extrair valores das potências dissipadas não somente estaticamente, mas também dinamicamente. A comparação detalhada de ambos os métodos estudados está apresentada na Tabela 5. A célula de retardo é uma célula que, como o próprio nome identifica, retarda a passagem do valor aplicado entra a sua entrada e sua a saída. Essa propriedade é conseguida inserindo uma grande capacitância em um dos estágios internos isolados, o que, inerentemente, causa a grande elevação de potência intrínseca dessa célula. Registradores por sua vez dissipam mais potência que células combinacionais quando suas entradas de *clock* têm seu valor alterado por ativarem múltiplos *gates* de transistores ao mesmo instante. Por isso, a dissipação relativa a chaveamento de nós é maior nesses elementos se comparados aos elementos combinacionais.

A potência estática do método de atraso casado é 14% menor que a do *diet clock*, mas sua ordem de grandeza é 1000 vezes menor que da potência dinâmica, o que constitui em uma pequena variação negativa do método do diet *clock*.

A somatória dos dois tipos de potência dinâmica favorece o método do *diet clock*, concluindo em uma melhor solução na geração de sinais temporizadores de operações para sinalizações assíncronas.

Tabela 5: Potência dissipada por ambos os métodos estudados

Método	**Potência dissipada**			
	Intrínseca à célula (µW)	**Chaveamento de nós (µW)**	**Dinâmica total (µW)**	**Estática (nW)**
Atraso casado	52,7	5,8	58,5	41,6
Diet clock	24,7	9,7	34,7	47,7

4 Implementação

A metodologia de conversão síncrono-assíncrono teve seu funcionamento comprovado através da implementação da técnica utilizando diversas linguagens e ferramentas, como esboçado no diagrama de fluxo da Figura 63.

4.1 Automatização da Conversão

O processo de conversão requer como entrada de dados os itens abaixo:

- descrição em portas lógicas do circuito original síncrono em verilog;
- nome do nível mais alto na hierarquia de módulos;
- nome associado à entrada primária do sinal de sincronismo (*clock*);
- nome associado à entrada primária do sinal responsável por inicializar os elementos de memória (*reset*);
- nomes das entradas primárias que terão comportamento estático durante o tempo de execução como pinos de configuração de modos de operação, por exemplo, determinados previamente à liberação do sinal de inicialização (*reset*).

O estudo da técnica de conversão requereu a elaboração de vários circuitos de teste capazes de exercitar cada etapa do processo, além de focalizar em situações e configurações peculiares em diversos aspectos de um circuito lógico genérico. Entretanto, a fim de comprovar o funcionamento em condições reais, circuitos que implementam funções conhecidas foram colhidos na Internet ou emprestadas de empresas para serem empregados como casos para análise, relacionados abaixo:

- FIR-3.8.8: filtro FIR de ordem 3, precisão de entrada de 8 bits, precisão dos coeficientes de 8 bits [66];
- RS Decoder: decodificador de *Reed-Solomon* [67];
- FIR-12.16.10: filtro FIR de ordem 12, precisão de entrada de 16 bits, precisão dos coeficientes de 10 bits [66];
- MDCT: transformada discreta de co-seno modificada [68];
- RSA Cipher: criptografador com algoritmo RSA [69];

- AES: criptografador com algoritmo AES (*Advanced Encryption Standard*) de 128 bits de largura para chave e dados [70].

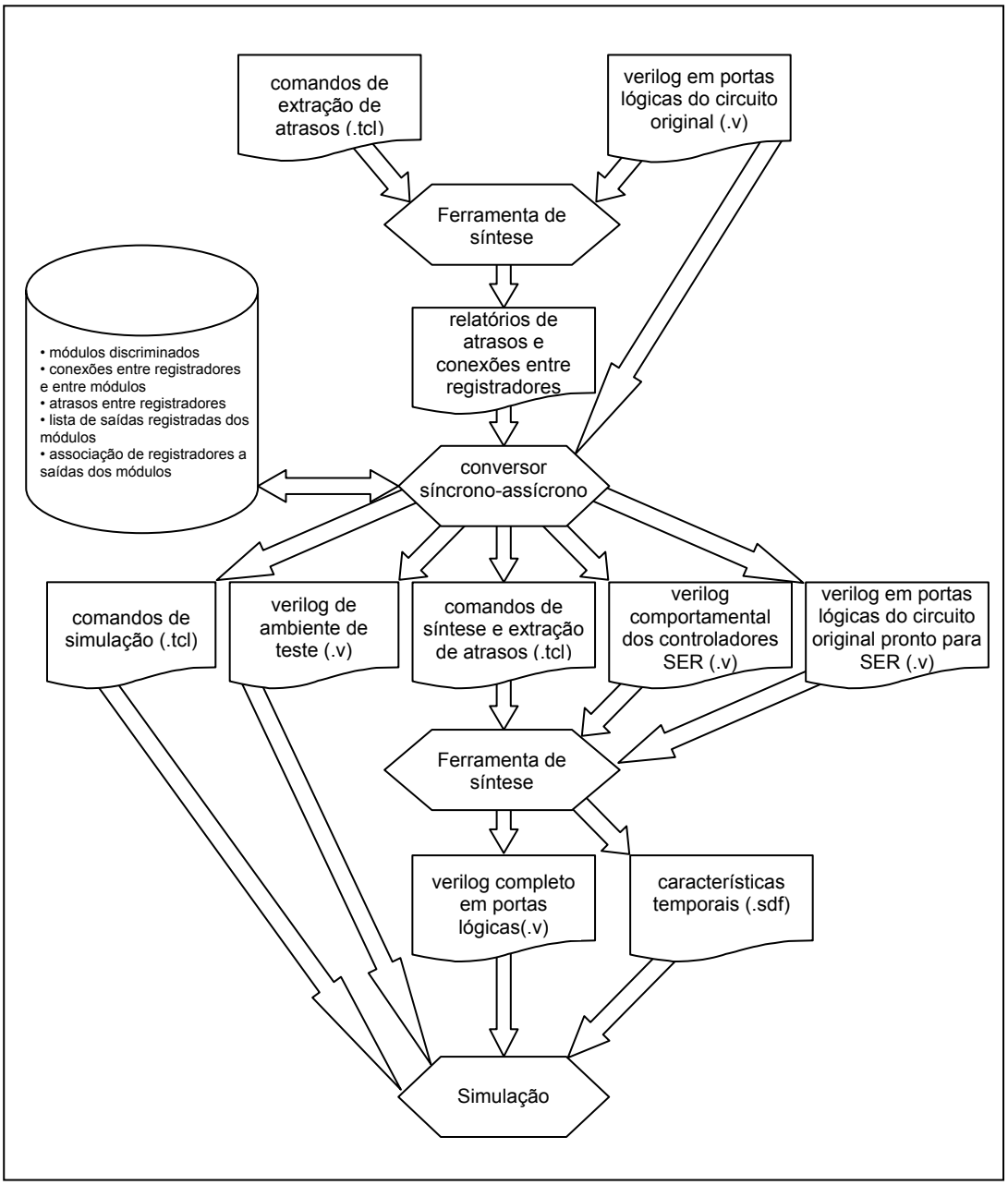


Figura 63: Diagrama de fluxo da implementação do conversor síncrono-assíncrono

A ferramenta de síntese utilizada para a implementação é o *Design Compiler* provido pela empresa americana *Synopsys*, sediada na Califórnia. As informações enumeradas anteriormente são disponibilizadas à ferramenta de síntese através de *scripts* escritos na linguagem de programação *tcl*, forma de entrada mais poderosa para o ambiente de síntese não interativo da ferramenta, cujo nome é *dc_shell*. A análise

prática foi baseada em uma biblioteca de células padrão projetada para tecnologia CMOS genérica de 0,13µm.

Nesse estágio são realizados levantamentos de todos os caminhos entre elementos de memorização do circuito, armazenando em listas o conjunto dos nomes dos elementos de partida e chegada associados ao tempo de propagação entre os mesmos. O tempo de execução desse passo é o mais longo a ser realizado durante o processo de conversão pois a complexidade não somente aumenta com o número de registradores, assim como com a quantidade de diferentes caminhos existentes entre dois elementos de retenção agregado à complexidade desses caminhos, por exemplo o número de níveis de hierarquia, profundidade dos caminhos e tipo de lógica no seu fluxo. Esse fator pode ser confirmado na Tabela 6, que mostra informações sobre os diferentes exemplos convertidos e os respectivos tempos de execução.

Tabela 6: Comparação de tempo de execução dos diferentes circuitos exemplos

Circuito aplicado	Número de portas equivalentes			Caminhos entre registradores	Tempo de execução (hh:mm:ss)
	Células combinacionais	Células seqüenciais	Total de células		
FIR-3.8.8	2.711	991	3.702	1.546	0:35
RS Decoder	7.460	4.355	11.816	10.242	7:57
FIR-12.16.10	21.099	6.158	27257	14.398	13:08
MDCT	17.049	13.233	30.283	37.491	15:40
RSA Cipher	3.800	4.238	8.038	41.133	17:38
AES	36.008	6.749	42.754	96.044	4:39:17

Os *scripts* foram aperfeiçoados função por função de forma a diminuir o tempo de execução total do estágio em mais de 5 vezes, desde a primeira versão até a atual. O exemplo mais proeminente é o código do circuito criptografador AES, que na primeira versão levou mais de 26 horas para completar a tarefa e na versão atual toma 4:40 horas de processamento dedicado em uma plataforma *CISC AMD*, à freqüência de 2GHz e um total de 2GB de RAM, sob o sistema operacional *Linux*.

Os resultados da análise realizada pela ferramenta de síntese são então fornecidos como entrada para o programa que realiza a conversão em si. Esse programa

foi baseado nas linguagens *gawk* (*GNU awk*) e *sh* (*Bourne shell*), ambas distribuídas juntamente com o sistema operacional utilizado.

O primeiro passo do conversor é ler o arquivo com o a descrição em nível de portas lógicas do circuito a ser trabalhado e extrair as seções de código que definem os módulos do sistema e redirecioná-los a arquivos distintos, ao mesmo tempo em que as entradas e saídas de cada um deles são analisadas, necessárias em passos adiante. O próximo estágio é gerar a indexação dos relatórios com conexões e tempos de propagação emitidos pela ferramenta de síntese a fim de aumentar a performance da execução dos próximos passos e armazená-los em um único arquivo.

Nesse instante o programa deixa o ambiente de *sh* e dá início à seqüência principal de em um programa escrito na linguagem *awk*. A partir do arquivo de indexação é formada uma estrutura de dados na memória dinâmica, que, para cada caminho traçado e anotado pela ferramenta de síntese são extraídos os seguintes dados:

- Rótulo de início do caminho anotado, na forma *módulo@instância*;
- Rótulo de fim do caminho anotado, na mesma forma do início;
- Nome do arquivo que contém o caminho completo, incluindo os tempos de propagação através de cada uma das portas;
- Módulo do início do caminho analisado;
- Módulo do fim do caminho analisado;
- Pino de saída da porta lógica de início ou nome da entrada primária e
- Pino de entrada da porta lógica do fim ou nome da saída primária.

Uma vez que a estrutura básica é montada, os arquivos contendo os tempos de propagação são interpretados, para que sejam extraídos e associados à sua respectiva base de dados: o tempo total de propagação do pior caso relacionado ao conjunto formado pelo ponto inicial e final do caminho em questão, qual saída do módulo que contém o início do caminho é usada, e qual registrador é responsável pela geração de dados do mesmo caminho.

O programa entra na fase da geração do código verilog responsável por descrever os elementos de sinalização inseridos automaticamente, substituir os sinais de *clock* por sinais de habilitação, e conectar ambos os blocos envolvendo-os em uma casca pronta para ser utilizada, como mostra a Figura 64, começando pelos módulos a serem instanciados como controladores de arestas e controladores de nós. Cada conexão

entre dois registradores resulta na criação de duas arestas: a que conecta o nó fonte ao nó de interconexão e a que conecta o nó de interconexão ao nó destino. Ao mesmo tempo os intervalos de operação de ambos os nós são comparados e uma aresta extra conectando o nó fonte ao nó destino é criada caso apresentem atrasos crescentes em relação ao fluxo de dados. Os códigos descritos em verilog para cada um dos controladores de arestas são armazenados em um arquivo para que seja integrado ao sistema completo ao finalizar o processo de conversão. A função que produz cada controlador de aresta é responsável também por invocar a função que gera os controladores de nós cuja aresta os interconecta, construindo simultaneamente os códigos verilog relativos a esses blocos. A função que cria os controladores de nós, por sua vez, aciona a edição do verilog em nível de portas do circuito original substituindo o sinal de *clock* do circuito pelos sinais de habilitação responsáveis por comandar o funcionamento dos registradores regidos pelos próprios controladores de nós que começaram o processo, contidos no módulo *original* da Figura 64.

O próximo passo do programa é a elaboração do módulo onde todos os blocos controladores de nós e os blocos controladores de arestas são instanciados e interconectados por fios. A interface desse módulo possui tanto saídas de habilitação que serão conectados aos seus respectivos registradores, assim como portas de sinalização com o meio exterior do sistema convertido. Dessa forma, o circuito responsável por toda a implementação da metodologia ASERT fica autocontido em um único módulo, sem se confundir com o código do circuito original, como o módulo *SER* da Figura 64.

O nível hierárquico mais alto do sistema é finalmente criado com o objetivo de realizar a integração do módulo que contém o circuito funcional e do módulo composto de todos os elementos de sinalização ASERT, representado pelo módulo *topo* na Figura 64. A interface desse bloco é composta somente pelas portas dos dados que saem do e entram no circuito funcional além das usadas na sinalização da disponibilidade de dados.

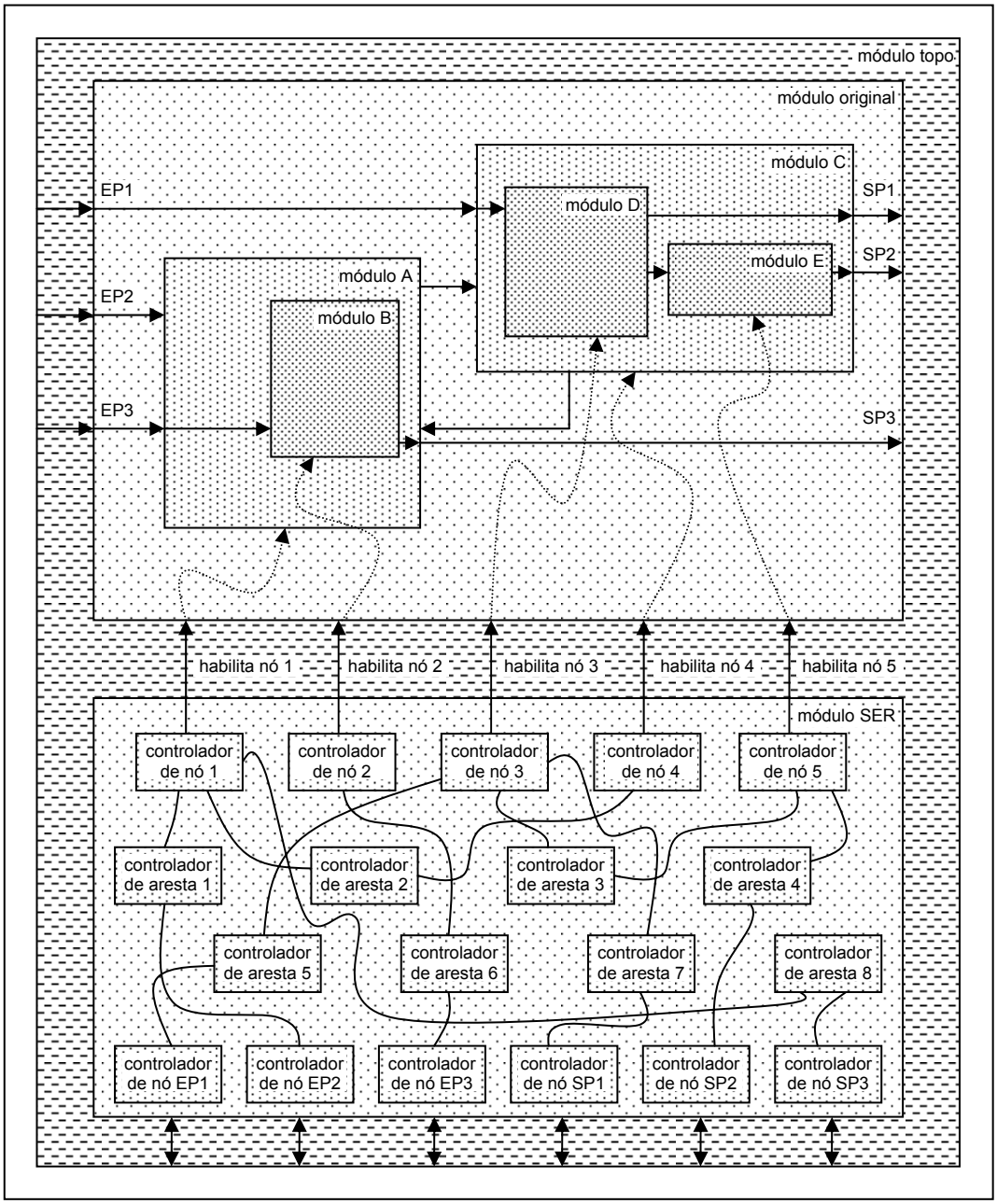


Figura 64: Organização do sistema completo convertido

Terminada a etapa de geração de módulos do sistema, tem início a construção de um ambiente de teste em Verilog, onde as entradas de dados do circuito são associadas a tarefas produtoras de estímulos compatíveis com as larguras das respectivas portas. Essas tarefas realizam o procedimento de sinalização que novos dados estão prontos para serem consumidos pelas entradas primárias e que os dados produzidos pelo circuito já foram armazenados das saídas. *Scripts* de compilação para ferramenta de simulação *ncsim* da *Cadence Design Systems*, sediada em San José, Califórnia, são criados automaticamente.

4.2 Resultados

A metodologia de conversão de circuitos síncronos em circuitos assíncronos funcionalmente equivalentes apresentada nesse trabalho empregou 6 núcleos de sistemas digitais, introduzidos na Seção 4, como plataformas de experimentação e desenvolvimento do algoritmo implementado. A alta complexidade dos núcleos utilizados como teste comprovam a capacidade do algoritmo de produzir resultados coerentes com caraterísticas arquiteturais como, por exemplo, máquinas de estado, fluxo de dados com realimentações, acesso a bancos de registradores ou memória e unidades lógicas e aritméticas.

A aplicação do procedimento de conversão apresentado na Seção 4 foi realizada a cada um dos 6 circuitos síncronos, resultando nas respectivas descrições em portas lógicas de sua versão assíncrona equivalente. A realização dos experimentos contou ainda com a análise das versões originais síncronas como base de comparação dos resultados quantitativos e qualitativos a serem esperados da versão assíncrona. Após a síntese dos códigos síncronos ao nível de portas lógicas, o procedimento para o levantamento das caraterísticas temporais, dissipação de potência e área ocupada é aplicado analogamente em ambas *netlists* síncronas e assíncronas, como apresentado no diagrama de fluxo da Figura 65.

A fim de realizar a extração dos valores da dissipação de potência não somente estática, mas dinâmica também, são necessários que vetores de exitação sejam aplicados coerentemente às entradas primárias do circuito analisado, de forma a exercitá-lo como em modo de operação normal. Os ambientes de teste para a versão assíncrona convertida, gerados automaticamente através do procedimento de conversão, são modificados para que sejam equivalentes aos da versão síncrona. Arquivos com comandos de controle de simulação e extração de informação das transições de todos os nós do circuito são aplicados juntamente com o código em portas lógicas do circuito analisado, agregado às suas características temporais calculadas pela ferramenta de síntese, resultando em uma base de dados que indicará à ferramenta de análise de potência como é o comportamento dinâmico do respectivo circuito.

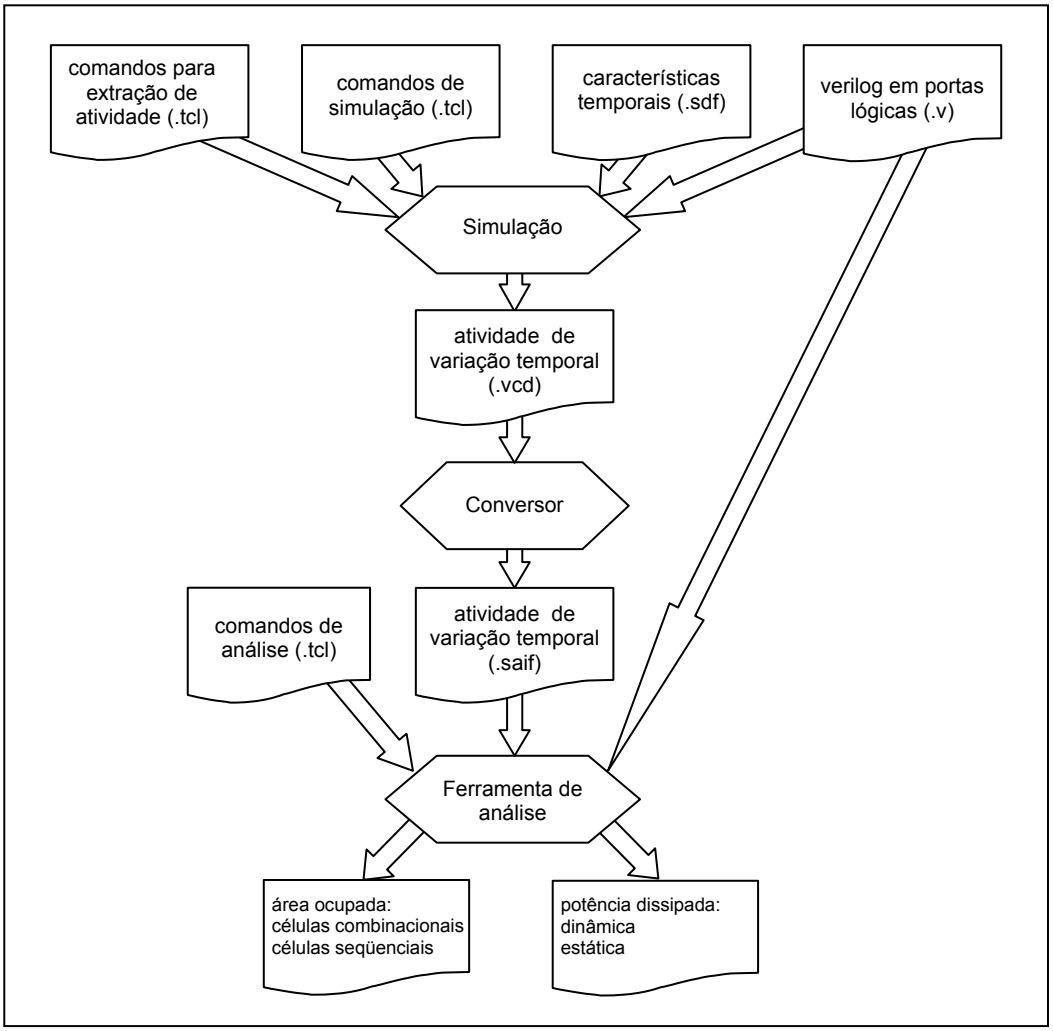


Figura 65: Fluxo aplicado para extração de dados sobre potência e área

Aplicando a técnica de análise aos 6 circuitos selecionados para teste, os fatores observados comparativamente foram:

- áreas ocupadas por células combinacionais e seqüenciais,
- duração da execução da função específica e
- potências estática e dinâmica dissipadas durante a execução da função específica do circuito.

A Tabela 7 apresenta características comuns das versões síncrona e assíncrona de cada um dos sistemas estudados.

Tabela 7: Características comuns entre versões síncrona e assíncrona

Circuito aplicado	Ciclos de operação	Caminho crítico (ns)	Largura de dados (bits)
FIR-3.8.8	23	9,61	8
RS Decoder	35	9,79	5
FIR-12.16.10	31	15,66	16
MDCT	88	18,68	8
RSA Cipher	655	21,81	32
AES	12	25,51	128

A coluna de *ciclos de operação* indica o número de ciclos de *clock* necessário para finalizar a tarefa completamente. Na versão assíncrona esse valor pode ser observado como o número de vezes que as entradas e saídas sinalizam a capacidade de respectivamente receber e prover dados novos.

A metodologia de conversão apresentada nesse trabalho não é invasiva, ou seja, nenhuma das portas originais do circuito é alterada assim como nenhum caminho de dados é perturbado, com exceção das conexões aos pinos de *clock* de células seqüenciais. Portanto, o caminho mais longo da versão assíncrona continua sendo o mesmo caminho crítico da versão síncrona e está disposto na coluna *caminho crítico*.

Os elementos de sinalização gerados pelo algoritmo conversor, como os controladores de nós e os controladores de arestas, estão relacionados na Tabela 8.

Tabela 8: Números dos controladores de nós e arestas dos circuitos estudados

Circuito aplicado	Controladores de nós	Controladores de nós de interconexão	Total de controladores de nós	Controladores de arestas
FIR-3.8.8	9	7	16	26
RS Decoder	130	123	253	2563
FIR-12.16.10	37	35	72	121
MDCT	100	95	195	2139
RSA Cipher	7	4	11	18
AES	22	19	41	307

Existe, como esperado, uma significativa correlação entre número de elementos criados pelo conversor e a estrutura arquitetural utilizada na versão síncrona. O circuito *MDCT*, por exemplo, possui somente 2 níveis hierárquicos. No entanto, existem 40 módulos de memória com dados de 8bits de largura que se conectam a 2 outros módulos que as utilizam e, por conseguinte, as controlam com as 40 respectivas saídas de endereço, além de sinais habilitações, todos gerados por máquinas de estados. Um cálculo aproximado demonstra que o número de controladores de nós pode rapidamente crescer em uma estrutura desse tipo:

$N_N \approx (N_m + (N_C + N_m)) \times 2$,

onde:
N_N = número de nós,
N_m = número de memórias e linhas independentes de endereços,
N_C = número de linhas de controle,
2 = controlador de nó + controlador de nó de interconexão.

Fazendo $N_m = 40$ e $N_C = 5$,
$N_N \approx (40 + 40 + 5) \times 2 = 170$.

Por outro lado, o filtro FIR de 3ª ordem, que possui 4 níveis hierárquicos, requer somente 16 controladores de nós, já que o nível mais baixo possui 6 saídas registradas e o nível imediatamente superior possui 1 saída registrada, e todos os outros módulos tem comportamento combinacional. Os 7 controladores de nós internos inferem em 7 outros controladores de nós de interconexão, que somados a 2 controladores dedicados para controle da entrada e saída primária, totalizam os 16 controladores de nós necessários para a implementação da sinalização assíncrona do circuito convertido.

O impacto de aumento de área ocupada resultante do emprego da conversão síncrona-assíncrona está refletido na comparação realizada em termos de portas equivalentes na Tabela 9.

Tabela 9: Comparação do número de portas equivalentes nas versões síncrona e assíncrona

Circuito aplicado	Portas equivalentes na versão síncrona			Portas equivalentes na versão assíncrona			Variação do total (%)
	Comb	Seq	Total	Comb	Seq	Total	
FIR-3.8.8	2.711	991	3.702	2.830	1.094	3.925	6,0
RS Decoder	7.460	4.355	11.816	18.580	5.987	24.567	108,0
FIR-12.16.10	21.099	6.158	27.257	21.658	6.622	28.281	3,53
MDCT	17.049	13.233	30.283	26.484	14.491	40.974	35,3
RSA Cipher	3.800	4.238	8.038	3.884	4.309	8.193	1,9
AES	36.008	6.749	42.754	37.363	7.013	44.373	3,8

Onde *Comb* significa combinacionais e *Seq* significa seqüenciais.
A última coluna representa a diferença percentual do total de portas equivalentes na versão assíncrona em relação a sua versão síncrona.

Controladores de nós e controladores de arestas são implementados em circuitos lógicos conforme apresentado na Seção 2.8. Um controlador de aresta é composto de uma porta lógica do tipo *ou exclusivo* de duas entradas e outra do tipo *inversor*. O controlador de nó, por sua vez, é constituído por portas do tipo *e*, cujo número de entradas depende diretamente do número de arestas que estejam conectadas ao nó, além de uma porta registradora do estado de fim de operação. Portanto a relação entre controladores de nós e controladores de arestas de áreas ocupadas não é fixa, mas dependente da complexidade das interconexões do circuito síncrono original. Como base de cálculo, a Tabela 10 relaciona as áreas ocupadas por cada um dos tipos de portas lógicas acima mencionados, supondo que o controlador de nó esteja sendo empregado em sua configuração mínima de duas arestas somado ao sinal de *run* (Figura 26).

O controlador de nó ocupa 80% a mais que um controlador de aresta, caso haja somente 2 outros controladores de nós sinalizando diretamente ao mesmo, o que na prática significa que controladores de nós têm seu peso dobrado em relação à ocupação de área se comparados aos controladores de arestas.

Um módulo que possua 1000 registradores cujos caminhos de dados passem todos através da mesma saída do módulo será sinalizado por um único controlador de nó, ao passo que se cada um desses registradores dispuser de uma respectiva saída independente, 1000 controladores de nós seriam necessários para sincronizá-los.

Tabela 10: Controladores de nós e arestas, seus elementos e respectivas áreas

Tipo de porta lógica	Célula da biblioteca	Área (μm^2)	Portas equivalentes
AND	AND3X2	8.49	1.70
Registrador	DFFRX2	37.34	7.47
XOR	XOR2X2	20.37	4.07
Inversor	INVX2	5.09	1.00
Controlador de aresta		25.46	5.07
Controlador de nó		45.83	9.17

Portanto, o aumento de área creditado à inserção dos elementos de sinalização assíncrona será dependente dos seguintes fatores:

- número de saídas registradas de cada módulo – afeta o número de controladores de nós,
- número de instâncias de cada módulo utilizado no sistema – afeta o número de controladores de nós,
- complexidade de interconexões entre módulos – afeta a área ocupada por cada controlador de nó.

O próximo item de comparação entre as versões síncrona e assíncrona do mesmo circuito é a variação da duração de execução dos testes aplicados. O método da conversão apresentado na Seção 4 mostra que o programa conversor se baseia fundamentalmente em relatórios de propagação de atrasos de caminhos específicos gerados pela ferramenta de síntese, como indicado no diagrama da Figura 63. Esses relatórios são usados não somente como informação para a construção dos controladores de nós, mas também como um meio de realizar o levantamento do caminho crítico do circuito original e fornecer esse valor ao ambiente de verificação da versão síncrona para análise dinâmica comparativa. Os dados de entrada são aplicados exatamente com a mesma temporização em ambos os casos; baseados em transições do sinal de *clock* na versão síncrona e nas sinalizações emitidas na versão assíncrona. As medidas de tempo de execução foram tomadas relativas a instantes que padrões idênticos foram aplicados ou gerados em ambas as versões do sistema. A Tabela 11

indica como a duração de execução dos testes foi afetado devido à conversão assíncrona de cada um dos circuitos estudados.

Tabela 11: Duração das execuções síncronas e assíncronas da mesma função

Circuito aplicado	Síncrono (ns)	Assíncrono (ns)	Variação (%)
FIR-3.8.8	223,6	227,9	1,9
RS Decoder	352,4	338,9	-3,8
FIR-12.16.10	488,6	494,8	1,3
MDCT	1.643	1.647	0,2
RSA Cipher	14.347	14.005	-2,4
AES	282,4	276,6	-2,1

A duração da execução dos testes aplicados a cada sistema se comportou relativamente de forma estável: 3 circuitos apresentaram uma pequena queda no tempo de execução, 2 outros apresentaram aumentos numericamente ainda menores e um permaneceu praticamente intacto. Na média houve uma queda de 0,81% no tempo de execução entre as versões síncronas e assícronas dos sistemas analisados. Todos os dados são resultados de simulações em nível de portas lógicas com anotação de atrasos, antes de *layout*.

Como mencionado na Seção 3.4, circuitos síncronos têm sua freqüência de operação determinada pelo o tempo de propagação mais longo entre 2 registradores ou entre registradores e entrada ou saídas primárias existentes em sua estrutura, enquanto circuitos assíncronos operam à freqüência relativa a média dos tempos de propagação dos mesmos caminhos. Portanto, variações negativas da duração de execução podem ser atingidas somente em casos onde haja caminhos com tempos de propagação diferentes entre si. Quanto maior o número de caminhos mais rápidos que o caminho crítico, maior a possibilidade de ganho de tempo na versão assíncrona. No entanto, as ferramentas de síntese voltadas para projeto de circuitos síncronos tendem a não otimizar caminhos que sejam mais rápidos que o caminho crítico, com o objetivo de não aumentar a ocupação de área e dissipação de potência desnecessariamente. No fundo, um dos passos no complicado processo de síntese é o de resgate de recursos dispensáveis após verificada a impossibilidade de aperfeiçoamento do caminho crítico, enxugando o *netlist* e fazendo

com que caminhos mais rápidos sejam traduzidos em economia de área e potência ao preço de aumento no tempo de propagação dos mesmos.

A Seção 3.3.2 apresentou o método de otimização temporal, seu incoveniente de perda de dados em situações específicas e a solução para esses casos, que causa a serialização de transmissão de dados no ponto afetado, impossibilitando o acionamento paralelo de nós interligados que possuam tempos de operação com valores crescentes em relação ao fluxo de dados. Sistemas que possuam essa característica têm seu tempo médio de propagação afetado, retardando o fluxo de dados e denegrindo a capacidade de processamento do mesmo.

Os fatores que influenciam no ganho de tempo de operação dos circuitos convertidos com a técnica apresentada são:

- metodologia de síntese; a síntese deve ser realizada de forma a otimizar todos os caminhos existentes no circuitos, incluindo aqueles que têm seus tempos de propagação menores que o caminho crítico e
- estágios separados por barreiras temporais devem ser organizados de forma a apresentar tempos de operação decrescentes em relação ao fluxo de dados do circuito, evitando o acréscimo da aresta entre nós com atraso crescente.

O terceiro e último ponto analisado na comparação entre as versões síncronas e assíncronas do mesmo sistema foi a quantidade de potência dissipada ao serem postos a executar a mesma tarefa. Novamente, o experimento contou com a aplicação de vetores idênticos em ambas as implementações e seu período de execução foi estabelecido a partir de dados de entrada e saída específicos, comprovando que os circuitos realizaram o mesmo trabalho.

Como nenhuma porta lógica existente na versão síncrona é alterada no processo de conversão, a potência estática dissipada pela versão assíncrona deve ser invariavelmente maior do que a versão síncrona, por ser diretamente proporcional à quantidade de transistores existentes no circuito. A Tabela 12 apresenta a variação das potências estática e dinâmica das duas implementações em situações de funcionamento contínuo.

Os valores das variações de potência estática apresentados são praticamente idênticos aos dados da variação de portas equivalentes da Tabela 9, já que quanto maior é a quantidade de portas lógicas adicionadas ao sistema, mais transistores são

incorporados ao circuito, resultando no aumento direto de dissipação de potência estática.

Tabela 12: Dissipação de potências estáticas e dinâmicas nos diversos sistemas

Circuito aplicado	Potência dissipada na versão síncrona		Potência dissipada na versão assíncrona		Variação da potência dissipada	
	Estática (µW)	Dinâmica (mW)	Estática (µW)	Dinâmica (mW)	Estática (%)	Dinâmica (%)
FIR-3.8.8	5,11	0,65	5,42	0,61	6,1	-6,2
RS Decoder	15,91	3,01	32,35	11,48	103,3	281,4
FIR-12.16.10	38,37	2,02	39,77	1,93	3,6	-0,45
MDCT	36,06	2,97	50,06	12,70	37,2	327,6
RSA Cipher	10,47	1,23	10,69	1,22	2,1	-0,8
AES	55,66	5,21	57,80	5,15	3,8	-1,2

Enquanto as variações da dissipação de potência dinâmica da maioria dos circuitos estudados parecem ser condizentes com os fundamentos de economia de potência pregado pela teoria de circuitos assíncronos, o circuito *MDCT* e o circuito decodificador *Reed-Solomon* chamam atenção na Tabela 12 pelos seus aumentos em relação a sua contra-parte síncrona. Para esses dois casos, o procedimento da extração da potência dissipada foi alterado de forma a distinguir a parte referente somente aos controladores ASERT da parte do circuito aplicado, com o objetivo de averiguar se a conversão para assíncrono alterou o comportamento do circuito original ao ponto de causar esse enorme aumento, e os resultados estão dispostos na Tabela 13.

Tabela 13: Subdivisão de potências dissipadas na versão assíncrona

Circuito aplicado	Potência dissipada no circuito estudado		Potência dissipada nos controladores ASERT	
	Estática (µW)	Dinâmica (mW)	Estática (µW)	Dinâmica (mW)
RS Decoder	15,91	1,68	16,44	9,80
MDCT	36,06	6,85	14,00	5,85

Do ponto de vista da potência dissipada no circuito estudado, a estática é exatamente igual a da versão síncrona. Em relação à potência dinâmica, o circuito decodificador *Reed-Solomon* apresentou uma queda de 44,2%, mas em contra-partida o circuito *MDCT* apresentou um aumento de 130%. O motivo principal deste aumento vem, novamente, do grande número de memórias conectadas ao mesmo bloco que as acessa. No caso síncrono, em que todas as memórias possuem um sinal centralizado de sincronismo para acesso, as diversas memórias têm seus valores lidos e disponibilizados simultaneamente, fazendo com o que o caminho de dados transicione até que se estabilize após o caminho mais longo ter sido percorrido. No caso assíncrono, como não pode ser garantido que as memórias terão seus sinais de sincronismo acionados concomitantemente, esse tempo de propagação se extende até que o caminho crítico da última memória ativada seja percorrido, ocasionando o aumento da duração de transições através da lógica calculatória do algoritmo assim como a respectiva dissipação de potência dinâmica, observado na diferença da Tabela 12 e da Tabela 13.

Ambos os casos apresentaram uma grande adição de potência dissipada vindo dos controladores de nós e arestas, fator explicado pelo excepcional número desses elementos inseridos na conversão, devido ao método da descrição do *hardware* empregado pelo projetista da versão síncrona. Esse não é um fato isolado caraterístico do processo da conversão apresentada. Estilos de descrição de *hardware* são comumente responsáveis pela qualidade dos resultados atingidos por ferramentas de síntese de circuitos síncronos e, apesar de muitos conseguirem implementar sistemas lógicos dos mais variados tipos, existem os bons e os maus projetistas, e esse fator pode influenciar tanto no desenvolvimento de circuitos síncronos assim como nos assícronos.

Durante o desenvolvimento do trabalho apresentado, o processo de evolução do algoritmo e do programa, incluindo a etapa de testes e verificação de resultados, teve o circuito criptografador AES de 128 bits como o circuito aplicado de maior complexidade de caminho de dados e máquinas de controle, além de conglomerar o maior número de portas lógicas dos circuitos estudados. A Figura 66 mostra o funcionamento síncrono original do circuito, com as entradas *Data_in*, *Key* e *Start*; as saídas *Data_out* e *Valid_Data*, e sinais internos como *SubBytes_out*, um dos sinais com maior intensidade de transições por ser a saída de uma memória vetorial conjugada a várias outras operações estabelecidas pelo algoritmo de criptografia.

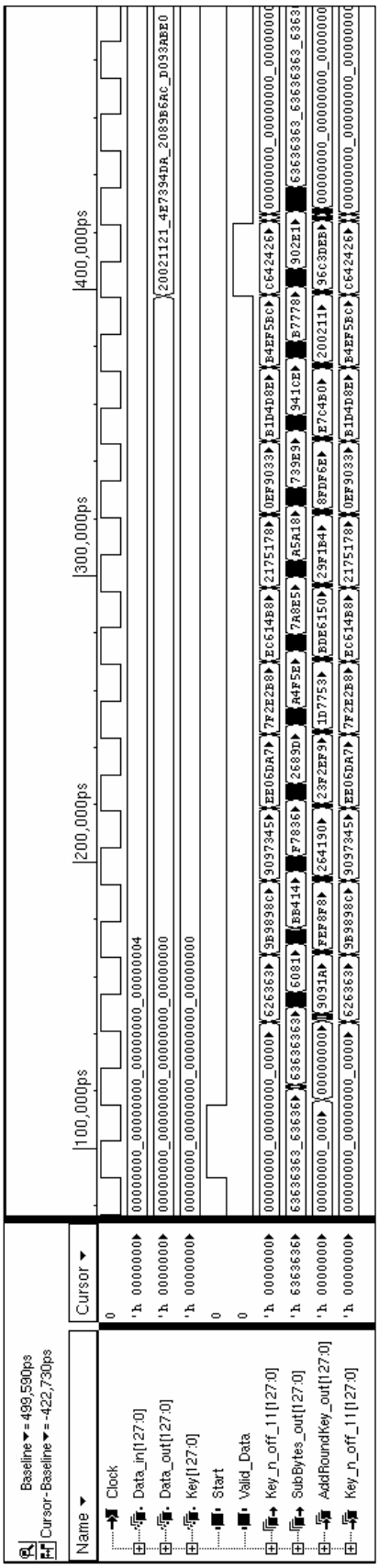


Figura 66: Formas de onda da operação do circuito AES em sua versão síncrona original

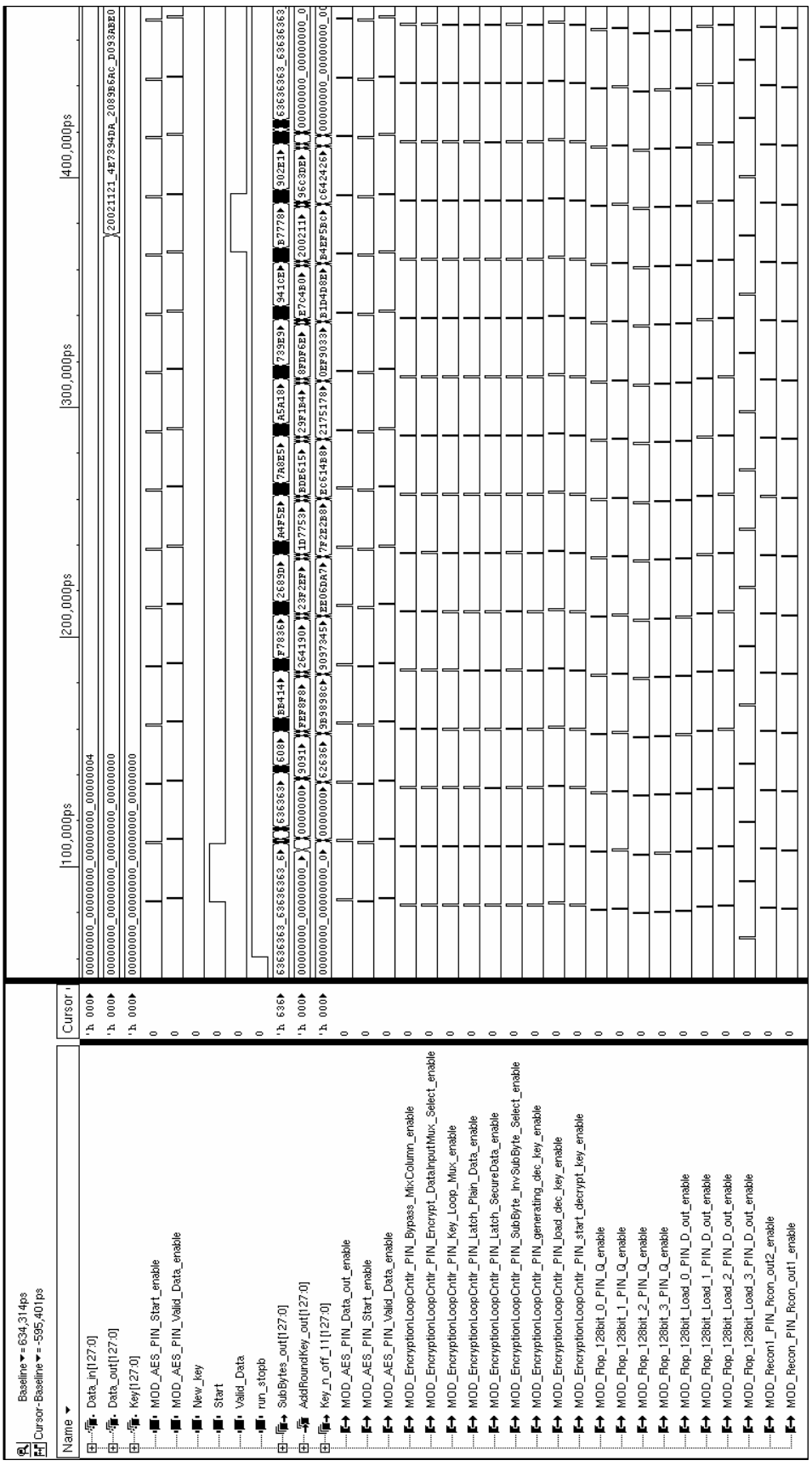


Figura 67: Formas de onda da operação do circuito AES em sua versão assíncrona

A Figura 67 apresenta o circuito AES após a aplicação da metodologia de conversão síncrono-assíncrono. Estão representados os sinais de entrada e saída primários, além dos sinais de habilitação gerados por cada um dos 22 controladores de nós inseridos no processo de conversão.

Apesar da complexidade do referido circuito, a conversão síncrono-assíncrono foi realizada com sucesso, resultando em ganhos tanto em potência dissipada como em tempo de execução da tarefa predestinada em relação à versão síncrona, mantendo, em última instância, inalterado o comportamento funcional do sistema original.

5 Conclusão

A metodologia para conversão de circuitos síncronos em circuitos assíncronos equivalentes proposta nesse trabalho possibilita uma simples forma de implementação de circuitos assíncronos utilizando ferramentas e bibliotecas de células básicas disponíveis no mercado, essencialmente aplicando o método ASERT de sinalização assíncrona de maneira automática. A técnica se mostrou capaz de converter diversos tipos de circuitos síncronos, cujas variações em arquitetura, dimensão e complexidade confirmaram a flexibilidade do algoritmo em tratar de estruturas como máquinas de estados, *pipelines*, realimentações, tabelas indexadas estáticas e dinâmicas, dentre outras.

O meio empregado para particionamento de sub-blocos passíveis de serem transformados em unidades assíncronas é extraído da divisão hierárquica do sistema original. Como observado durante a apresentação dos resultados na Seção 4.2, a conversão nem sempre traz os benefícios esperados devido à estrita dependência da arquitetura original, apesar de em nenhum dos casos se mostrar inatingível. No entanto, caso o projetista almeje o circuito assíncrono através da metodologia de conversão proposta, a estrutura de divisão modular e hierárquica do circuito síncrono em desenvolvimento pode ser direcionada a contribuir para uma conversão mais bem otimizada, através do modo como o agrupamento funcional em módulos do sistema é realizado. O circuito pode ser completamente projetado, implementado, testado e aperfeiçoado no domínio síncrono antes de ser processado pela ferramenta produzida pelo trabalho, além de permitir o uso de todo o suporte provido ao desenvolvimento de circuitos síncronos, como sintetizadores, simuladores e depuradores.

O processo de conversão foi implementado por programas escritos nas linguagens *bourne shell*, responsável por controlar a execução de todos os passos do método; *tcl*, forma de entrada de comandos e controle das ferramentas de CAD, e *awk*, responsável por criar, instanciar e interconectar os devidos controladores de nós e arestas necessários para a conversão. A implementação alcançou um alto grau de automatização, restando ao usuário fornecer poucas informações além do *netlist* síncrono:

- nome do nível mais alto na hierarquia de módulos;
- nome associado à entrada primária do sinal de sincronismo (*clock*);

- nome associado à entrada primária do sinal responsável por inicializar os elementos de memória (*reset*) e
- nomes das entradas primárias que terão comportamento estático durante o tempo de execução.

Circuitos podem ser desenvolvidos, testados, convertidos e armazenados em uma biblioteca para utilização posterior. Como cada entrada ou saída não estáticos dos blocos convertidos possui um controlador de nó associado, diversos circuitos com diferentes características temporais podem ser facilmente interconectados pelo simples intermédio de um controlador de aresta externo entre ambos os blocos. Dessa forma, o tempo gasto para implementação de um sistema é reduzido, simplificando também o processo de casamento entre blocos com diferentes domínios de freqüência.

Os resultados gerados pela aplicação da metodologia foram satisfatórios, com ganhos em dissipação de potência na maioria dos casos e ganhos em performance em metade do número de exemplos. Os trabalhos correlatos não apresentam informações suficientes para realizar uma comparação à altura. O trabalho apresentado por GINOSAR et al. [30] se baseia em alguns exemplos fictícios e outros não muito bem definidos, com o filtro FIR sem determinação de sua ordem, precisão da entrada ou dos coeficientes. Seu exemplo de maior dimensão, o filtro FIR com 1128 portas equivalentes, é 3 vezes menor do que o menor dos exemplos apresentados nesse trabalho, o filtro FIR de ordem 3, com 8 bits de precisão de entrada e coeficientes. O único resultado quantitativo significativo divulgado é o aumento de área de 26% no filtro FIR, 6 vezes maior do que o filtro FIR mais simples resultante da metodologia apresentada.

Na técnica de conversão apresentada por LAVAGNO et al. [31], bastante similar à acima comentada, os exemplos também deixam a desejar. No mais impressionante deles, a conversão de um processador DLX, foram empregados somente 4 controladores de sinalização assíncrona, um para cada estágio inteiro do *pipeline* do processador, o que minimiza a importância da aplicação de um método automático de conversão e não revela a real capacidade da técnica apresentada. O outro circuito exemplo usado para comprovação de resultados foi um núcleo criptografador DES, criado pelos autores. Esse circuito é fundamentalmente um *pipeline* de 16 estágios com deslocamentos e permutações de bits, sem máquinas de estados ou outros elementos mais complexos. Seu ganho é expresso em 12% em relação à potência dissipada, enquanto sua latência

sofre um aumento de 3% e sua área de 22%. O elemento comparativo mais próximo na versão apresentada nesse trabalho é o AES, somente pelo fato de servirem ao propósito de criptografia, pois o AES é extremamente mais complexo e seguro. Os resultados mostram uma melhora tanto em sua latência, de 2,1%, quanto uma redução de 1,2% em sua potência dinâmica dissipada e um aumento de área de 3,8%.

A metodologia apresentada se difere ainda das duas outras mencionadas por não ser invasiva, ao manter inalterarados os elementos do circuito original. A substituição de um único sinal de *clock* pelos respectivos sinais habilitadores gerados nos controladores de nós é a mínima alteração possível durante um processo de conversão síncrono-assíncrono. A metodologia apresenta flexibilidade de atuação em diversas estruturas comumente encontradas em circuitos síncronos, aliada à capacidade do método ASERT de sinalização assíncrona ser independente do tipo de geração do sinal de fim de operação, como atraso casado, o *diet clock*, codificação *bundled rail* ou codificação *dual rail*. O processo de conversão introduzido nesse trabalho é uma ferramenta poderosa, porém simples e direta para elaboração de circuitos assíncronos a partir de um fluxo trivial de desenvolvimento de circuitos síncronos.

Os trabalhos sugeridos a derivarem desse estudo incluem:

- procedimento automático de lógica para geração de fim de operação,
- sugestões de estilo para descrição de *hardware* para que o circuito síncrono seja convertido de maneira mais eficaz,
- análise de como se comportam as conversões de sistemas que possuam mais que um domínio de *clock* e
- adaptação de controladores de nós que possuam estado de inativo além dos atuais aguardando ou operando.

6 Referências Bibliográficas

1 HUNT, M., ROWSON, J. A., "Blocking in a System on a Chip", *Spectrum*, IEEE, v. 33, n. 11, pp. 35-41, November 1996.

2 GEPPERT, L., "Solid State [trends/developments]", *Spectrum*, IEEE, v. 33, n. 1, pp. 51-55, January 1996.

3 CHANDRAMOULI, R., PATERAS, S., "Testing Systems on a Chip", *Spectrum*, IEEE, v.33, n. 11, pp. 42-47, November 1996.

4 RINCON, A. M., CHERICHETTI, G., MONZEL, J. A. et al., "Core Design and System on a Chip Integration", *Design & Test of Computers*, IEEE, v. 14, n. 4, pp. 26-35, December 1997.

5 BRICAUD, P. J., "IP Reuse Creation for System on a Chip Design*"*, *Proceedings of the IEEE 1999 Custom Integrated Circuits*, pp. 395-401, May 1999.

6 SAVAGE, W., CHILTON, J., CAMPOSANO, R., "IP Reuse in the System on a Chip Era", In: *Proceedings of the 13th International Symposium on System Synthesis*, pp. 2-7, September 2000.

7 SARKAR, S., CHANCLAR, G. S., SHINDE, S., "Effective IP Reuse for High Quality SOC Design", In: *Proceedings of the International SOC Conference*, pp. 217-224, Tampere, Finland, September 2005.

8 MARTIN, A. J., "Remarks on Low-Power Advantages of Asynchronous Circuits", In: *ESSCIRC: Low-Power Systems on a Chip*, September 1998.

9 VAN BERKEL, K. et al., "Asynchronous Circuits for Low Power: A DCC Error Corrector", *Design & Test of Computers*, IEEE, v. 11, n. 2, pp. 22-32, 1994.

10 PIGUET, C., "Low-Power and Low-Voltage CMOS Digital Design", *Microelectronic Engineering*, v. 39, n. 1, pp. 179-208, 1997.

11 NIELSEN, L. et al., "Low-Power Operation Using Self-timed Circuits and Adaptive Scaling of the Supply Voltage", *IEEE Transactions on VLSI*, v. 2, n. 4, pp. 391-397, 1994.

12 DONNO, M., MACII, E., MAZZONI, L., "Power-Aware *Clock* Tree Planning", In: *Proceedings of ISPD'04*, pp. 138-147, Phoenix, USA, April 2004.

13 PANYASAK, D., SICARD, G., RENAUDIN, M., "From RTL Code Description to Low EMI Asynchronous Circuits", *Third Asynchronous Circuit Design Workshop,* Heraklion, Greece, January 2003.

14 FOURNIER, J. et al., "Security Evaluation of Asynchronous Circuits", *CHES Conference*, v. LNCS 2779, pp. 125-136, Springer-Verlag, Germany, September 2003.

15 HAUCK, "Asynchronous Design Methodologies", *Proceedings of the IEEE*, v. 83, n. 1, pp. 69-93, January 1995.

16 NOWICK et al., "Asynchronous Circuits and Systems", *Proceedings of the IEEE*, v. 87, pp. 219-222, 1999.

17 MARTIN, A. J., NYSTROM, M., "Asynchronous Techniques for System-on-Chip Design", *Proceedings of the IEEE*, v. 94, n. 6, pp. 1089-1120, June 2006.

18 KARAKI, N., "Asynchronous Design: An Enabler for Flexible Microelectronics", *Invited talk at the 12th IEEE International Symposium on Asynchronous Circuits and Systems*, Grenoble, France, March 2006.

19 SCHOELLKOPF, J.-P, “ATRS: An Alternative Roadmap for Semiconductors, Technology Evolution and Impacts on System Architecture”, *Invited talk at the 12th IEEE International Symposium on Asynchronous Circuits and Systems*, Grenoble, France, March 2006.

20 OBRIDKO, I., GINOSAR, R., “Minimal Energy Asynchronous Dynamic Adders”, *IEEE Transactions on VLSI*, v. 14, n. 9, pp. 1043-1047, September 2006.

21 TEIFEL, J., “Asynchronous Cryptographic Hardware Design”, In: *Proceedings of the 40th Annual IEEE International Carnahan Conferences Security Technology*, pp. 221-227, Lexington, KY, USA, October 2006.

22 YING, Y., LEI, Z., HAO, M., “Design and VLSI Implementation of an Asynchronous Low Power Microcontroller”, In: *Proceedings of the 4th International Conference on ASIC*, pp. 797-799, Shanghai, China, October 2001.

23 ALLIER, E. GOULIER, J. SICARD, G. et al., “A 120nm Low Power Asynchronous ADC”, In: *Proceedings of the 2005 International Symposium on Low Power Electronics and Design*, pp. 60-65, San Diego, CA, USA, August 2005.

24 AUSTIN, T., BERTACCO, V., BLAAUW, D. et al., “Opportunities and Challenges for Better Than Worst-Case Design”, In: *Proceedings of the ASP-DAC 2005 Asia and South Pacific Design Automation Conference*, v. 1, pp. 2-7, Shanghai, China, January 2005.

25 KARTHIK, S., DE SOUZA, I., RAHMEH, J. T. et al., “Interlock Schemes for Micropipelines: Application to a Self-Timed Rebound Sorter”, In: *Proceedings of the International Conference Computer Design (ICCD) 1991,* pp. 393-396, Cambridge, MA, USA, 1991.

26 LIEBCHEN, A., GOPALAKRISHNAN, G., “Dynamic Reordering of High Latency Transactions Using a Modified Micropipeline”, In: *Proceedings of the International Conference Computer Design (ICCD) 1992*, pp. 336-340, Cambridge, MA, USA, 1992.

27 MARTIN, “Programming in VLSI: From Communicating Processes to Delay-Insensitive Circuits”, In: C. A. Hoare, *Developments in Concurrency and Communication*, Boston, MA, USA, Addison-Wesley, 1990.

28 RENAUDIN, M. et al., “A Design Framework for Asynchronous/Synchronous Circuits Based on CHP to HDL Translation”, In: *Proceedings of the International Symposium on Advanced Research in Asynchronous Circuits And Systems*, pp. 135-144, Barcelona, Spain, April 1999.

29 VAN BERKEL, K., KESSELS, J., RONCKEN, M. et al., “The VLSI-Programming Language Tangram and Its Translation into Handshake Circuits”, In: *Proceedings of the European Conference on Design Automation*, Amsterdam, Netherlands, February 1991.

30 BRANOVER, A., KOL, R., GINOSAR, R., “Asynchronous Design by Conversion: Converting Synchronous Circuits into Asynchronous Ones” In: *Proceedings of the DATE*, v. 2, pp. 870-875, Paris, France, February 2004.

31 CORTADELLA, J., YAKOVLEV, A., LAVAGNO, L. et al., “Designing Asynchronous Circuits from Behavioral Specifications with Internal Conflicts”, In: *Proceedings of the International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pp. 106-115, Salt Lake City, USA, 1994.

32 YUN, K. Y., DILL, D. L., “Automatic Synthesis of 3D Asynchronous State Machines”, In: *Proceedings of the International Conference Computer Design (ICCD)*, pp. 576-580, Cambridge, MA, USA, 1992.

33 FUHRER, R. M., NOWICK, S. M., THEOBALD M. et al., *Minimalist: An Environment for Synthesis, Verification and Testability of Burst-Mode Asynchronous Machines*, CUCS-020-99, Columbia University, Computer Science Department, New York, NY, USA, 1999.

34 OETIKER et al., “Design Flow for a 3-Million Transistor GALS Test Chip”, *Third Asynchronous Circuit Design Workshop*, ACiD 2003, Heraklion, Greece, January 2003.

35 CORTADELLA, J., KONDRATYEV, A., LAVAGNO, L. et al., “Desynchronization: Synthesis of Asynchronous Circuits From Synchronous Specifications”, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, v. 25, n. 10, pp. 1904-1921, Oct. 2006.

36 KOL, R., GINOSAR, R., “A Doubly-Latched Asynchronous Pipeline”, In: *Proceedings of the International Conference Computer Design (ICCD),* pp. 706-711, Austin, TX, USA, 1997.

37 FRANÇA, F., ALVES, V., GRANJA, E., “Edge Reversal-Based Asynchronous Timing Synthesis”, *Proceedings of the IEEE International Synposium on Circuits and Systems*, Monterey, USA, June 1998.

38 FRANÇA, F., ALVES, V., GRANJA, E., “Processo de Síntese e Aparelho para Temporização Assíncrona de Circuitos e Sistemas Digitais Mutlifásicos”, *Patente nº 9703819-9*, Julho de 1997.

39 VAN BERKEL, C., JOSEPHS, M., NOWICK, S., “Scanning the Technology: Applications of Asynchronous Circuits”, *Proceedings of the IEEE*, v. 87, n. 2, pp. 223-233, February 1999.

40 Furber, S.B.; Edwards, D.A.; Garside, J.D., “AMULET3: a 100 MIPS asynchronous embedded processor”, *International Conference on Computer Design*, pp. 329-334, Austin, TX, USA, 2000.

41 LIU, Y., LI, Z., CHEN, P. et al., “Power-Efficient Asynchronous Design”, Jointly In: *6th International Conference on Embedded Systems, 20th International Conference on VLSI Design*, pp. 451-458, New Delhi, India, January 2007.

42 Tanabe, S. Nagano, N. Itoh, T. Murato, Y. Mizukawa, S. Sato, K., “Design method of a metallic enclosure considering EMI using a 3D-FEM”, *Electrical Performance of Electronic Packaging*, pp. 64–66, October 1995.

43 Chappel, J.F., Zaky, S.G., “EMI effects and timing design for increased reliability in digital systems”, *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, v. 44, pp. 130-142, February 1997.

44 BINK, A., YORK R., DE CLERCQ, M.,”ARM996HS: The First Licensable, *Clock*less 32-bit Processor Core*”, Hotchips*, August 2006.

45 HAUCK, O., SAUERWEIN, H., HUSS, S. A., “Asynchronous VLSI Architectures for Huffman Codecs”, *Proceedings of the 1998 IEEE International Symposium on Circuits and Systems*, v. 5, pp. 542-545, San Diego, CA, USA, June 1998.

46 RENAUDIN, M., *Architectures VLSI Asynchrones,* TELECOM Bretagne, Grenoble, France, 1996.

47 MARTIN, A., “The Limitation of Delay-Insensitivity in Asynchronous Circuits*”, Advanced Research In VLSI: Proceedings Of The Sixth MIT Conference,* MIT Press, Cambridge, MA, USA, April 1990.

48 BURNS, S., *Performance Analysis and Optimization of Asynchronous Circuits*, , Ph.D. dissertation, California Institute of Technology, Technical Report CS-TR-91-01, Pasadena, CA, USA, 1991.

49 KONDRATYEV, A., KISHINEVSKY M., LIN, B. et al., “Basic Gate Implementation of Speed-Independent Circuits”*,* In: *Proceedings of the 31st ACM/IEEE Design Automation Conference,* San Diego, CA, USA, June 1994.

50 UNGER, S., *Asynchronous Sequential Switching Circuits*, New York, NY, Wiley-Interscience, 1969.

51 KONDRATYEV, A., KISHINEVSKY M., YAKOVLEV, A., “Hazard-Free Implementation of Speed-Independent Circuits*”, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, v. 17, n. 9, September 1998.

52 DAVIS, A., NOWICK, S., *An Introduction to Asynchronous Circuit Design,* Technical Report UUCS-97-013, University of Utah, USA, September 1997.

53 NOWICK, S., *Automatic Synthesis of Burst-Mode Asynchronous Controllers*, Ph.D. dissertation, Stanford University, Technical Report CSL-TR-95-686, Stanford, CA, USA, December 1995.

54 T. Murata, "Petri Nets: Properties, Analysis and Applications", *Proceedings of the IEEE*, v. 77, n. 4, pp. 541-580, 1989.

55 PATIL, S., *An Asynchronous Logic Array,* Technical Report Memorandum 62, Massachusetts Institute of Technology, Project MAC, Cambridge, MA, USA, 1975.

56 CHU, T., "Automatic Synthesis and Verification of Hazard-free Control Circuits from Asynchronous FSM Specifications", In: *Proceedings of the IEEE International Conference on Computer Design*, IEEE Computer Society Press, Los Alamitos, CA, USA, 1992.

57 PURI, R., GU, J., "A Modular Partitioning Approach for Asynchronous Circuit Synthesis", In: *Proceedings of the 31st ACM/IEEE Design Automation Conference*, ACM, San Diego, CA, USA, June 1994.

58 LAVAGNO, L., MOON, C., BRAYTON, R. et al., "Solving the State Assignment Problem for Signal Transition Graphs", In: *Proceedings of the 29th IEEE/ACM Design Automation Conference,* IEEE Computer Society Press, Anaheim, CA, USA, June 1992.

59 ENDECOTT, P., "Parallel Structures for Asynchronous Microprocessors", *Computer Architecture Newsletter,* IEEE Computer Society Technical Committee on Computer Architecture, October 1995.

60 SUTHERLAND, I. E., "Micropipelines", *Communications of the ACM*, v. 32, n. 6, pp. 720-738, June 1989.

61 HAUCK, S., "Asynchronous Design Methodologies: An Overview", *Proceedings of the IEEE*, v. 83, n. 1, pp. 69-93, January 1995.

62 FURBER, S., "Asynchronous Design for Low Power", *1st UK Low Power Forum*, Sheffield, UK, 1998.

63 AKERS, S. B., "On the Use of Linear Sums in Exhaustive Testing", In: *Proceedings of the 15th Symposium on Fault-Tolerant Computing*, pp. 148-153, Niagara-on-the-Lake, Ontario, Canada, October 1985.

64 BARBOSA, V., GAFNI, E., "Concurrency in Heavily loaded Neighborhood-Constrained Systems", *ACM Transactions on Programming Language and Systems*, v. 11, n. 4, Oct. 1989.

65 BARBOSA, V., *An Introduction to Distributed Algorithms*, MIT Press, Cambridge, MA, USA, 1996.

66 HAWKINS, T., *CF FIR Filter*, http://www.opencores.org/projects.cgi/web/cf_fir, March, 2003.

67 PUTRA, R., *Reed-Solomon Decoder (31, 19, 6),*
http://www.opencores.org/projects.cgi/web/rs_decoder_31_19_6, February 2006.

68 KREPA, M., *Discrete Cosine Transform Core*, http://www.opencores.org/projects.cgi/web/mdct, April 2006.

69 MCQUEEN, S., *Basic RSA Encryption Engine*, http://www.opencores.org/projects.cgi/web/basicrsa, October 2003.

70 *Advanced Encryption Standard*, Federal Information Processing Standards Publication 197, November 2001.

7 Apêndices

7.1 Programa emulador de ASERT

Saída do programa emulador de ASERT desenvolvido para auxiliar na depuração de problemas ocorridos durante a elaboração do processo de conversão síncrono-assíncrono.

```
At: 0ns
Data:     1          0          0                      0                    0
Node:  ( EP ) o-- ( No1 ) o-- ( No2 ) o-- ( I.No3 ) o-- ( No3 ) o-- ( I.SP ) o-- ( SP )
Delay:    1ns        4ns        5ns          3ns        1ns          1ns        1ns

At: 1ns
Data:     1          1          0                      0                    0
Node:  ( EP ) --o ( No1 ) o-- ( No2 ) o-- ( I.No3 ) o-- ( No3 ) o-- ( I.SP ) o-- ( SP )
Delay:    1ns        4ns        5ns          3ns        1ns          1ns        1ns

At: 5ns
Data:     2          1          1                      0                    0
Node:  ( EP ) o-- ( No1 ) --o ( No2 ) o-- ( I.No3 ) o-- ( No3 ) o-- ( I.SP ) o-- ( SP )
Delay:    1ns        4ns        5ns          3ns        1ns          1ns        1ns

At: 6ns
Data:     2          1          1                      0                    0
Node:  ( EP ) --o ( No1 ) --o ( No2 ) o-- ( I.No3 ) o-- ( No3 ) o-- ( I.SP ) o-- ( SP )
Delay:    1ns        4ns        5ns          3ns        1ns          1ns        1ns

At: 10ns
Data:     2          2          1                      0                    0
Node:  ( EP ) --o ( No1 ) o-- ( No2 ) --o ( I.No3 ) o-- ( No3 ) o-- ( I.SP ) o-- ( SP )
Delay:    1ns        4ns        5ns          3ns        1ns          1ns        1ns

At: 13ns
Data:     2          2          1                      1                    0
Node:  ( EP ) --o ( No1 ) o-- ( No2 ) o-- ( I.No3 ) --o ( No3 ) o-- ( I.SP ) o-- ( SP )
Delay:    1ns        4ns        5ns          3ns        1ns          1ns        1ns

At: 14ns
Data:     3          2          2                      1                    0
Node:  ( EP ) o-- ( No1 ) --o ( No2 ) o-- ( I.No3 ) o-- ( No3 ) --o ( I.SP ) o-- ( SP )
Delay:    1ns        4ns        5ns          3ns        1ns          1ns        1ns

At: 15ns
Data:     3          2          2                      1                    1
Node:  ( EP ) --o ( No1 ) --o ( No2 ) o-- ( I.No3 ) o-- ( No3 ) o-- ( I.SP ) --o ( SP )
Delay:    1ns        4ns        5ns          3ns        1ns          1ns        1ns

At: 16ns
Data:     3          2          2                      1                    1
Node:  ( EP ) --o ( No1 ) --o ( No2 ) o-- ( I.No3 ) o-- ( No3 ) o-- ( I.SP ) o-- ( SP )
Delay:    1ns        4ns        5ns          3ns        1ns          1ns        1ns

At: 19ns
Data:     3          3          2                      1                    1
Node:  ( EP ) --o ( No1 ) o-- ( No2 ) --o ( I.No3 ) o-- ( No3 ) o-- ( I.SP ) o-- ( SP )
Delay:    1ns        4ns        5ns          3ns        1ns          1ns        1ns

At: 22ns
Data:     3          3          2                      2                    1
Node:  ( EP ) --o ( No1 ) o-- ( No2 ) o-- ( I.No3 ) --o ( No3 ) o-- ( I.SP ) o-- ( SP )
Delay:    1ns        4ns        5ns          3ns        1ns          1ns        1ns

At: 23ns
Data:     4          3          3                      2                    1
Node:  ( EP ) o-- ( No1 ) --o ( No2 ) o-- ( I.No3 ) o-- ( No3 ) --o ( I.SP ) o-- ( SP )
Delay:    1ns        4ns        5ns          3ns        1ns          1ns        1ns

At: 24ns
Data:     4          3          3                      2                    2
Node:  ( EP ) --o ( No1 ) --o ( No2 ) o-- ( I.No3 ) o-- ( No3 ) o-- ( I.SP ) --o ( SP )
Delay:    1ns        4ns        5ns          3ns        1ns          1ns        1ns
```

onde:
`Data` é o dado armazenado pelo registrador controlado pelo nó abaixo,
`( XX )` representa o nó XX,
`--o` representa uma aresta apontando para o nó à direita,
`o--` representa uma aresta apontando para o nó à esquerda e
`Delay` é o tempo de operação do nó acima