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

HELEN DE SOUZA FRANCK

Avaliação de Atraso, Consumo e Proteção de Somadores Tolerantes a Falhas

Dissertação apresentada como requisito parcial para a obtenção do grau de Mestre em Microeletrônica

Prof. Dr. Ricardo Augusto da Luz Reis
Orientador

Prof. Dr. José Luís Almada Güntzel
Co-orientador

Porto Alegre, janeiro de 2011.

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Franck, Helen de Souza

Avaliação de Atraso, Consumo e Proteção de Somadores Tolerantes a Falhas / Helen de Souza Franck – Porto Alegre: Programa de Pós-Graduação em Microeletrônica, 2011.

165 f.:il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Microeletrônica. Porto Alegre, BR – RS, 2011. Orientador: Ricardo Augusto da Luz Reis; Co-orientador: José Luís Almada Güntzel.

1.Tolerância a Falhas 2. SET (*Single-Event Transient*) 3. Circuitos Somadores 4. Sistema de Numeração de Dígito Binário com Sinal. I. Reis, Ricardo Augusto da Luz. II. Güntzel, José Luís Almada. III. Título.

AGRADECIMENTOS

Agradeço a Deus, em primeiro lugar por ter me dado como pais duas pessoas maravilhosas que me ensinaram as lições que realmente importam na vida: lições de amor, sinceridade, amizade, hospitalidade, respeito, fé, caráter e principalmente, me ensinaram o sentido da vida. É por eles e para eles que estou concluindo mais essa etapa. Amo vocês e muito obrigada por tudo.

Também agradeço a Deus pela oportunidade de fazer o mestrado, embora eu ainda não compreenda bem os planos Dele, mas foi através do mestrado que conheci o Alexandre, hoje meu marido, uma pessoa que tem me dado muito apoio e tem tido muita paciência. Ita, muito obrigada e te amo muito!

Agradeço aos meus orientadores Ricardo Reis e José Güntzel pela orientação e oportunidade que me foi dada, tanto em relação ao mestrado, como também de participar de vários congressos.

Agradeço em especial ao Gustavo Wilke por toda ajuda, contribuições técnicas, incentivos e paciência durante a realização dos experimentos práticos deste trabalho.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

BSD	*Binary Signed Digit* (Dígito Binário com Sinal)
BSDA_BSD	*Binary Signed Digit Adder* com operandos em BSD NAF
BSDA_C2	*Binary Signed Digit Adder* com operandos em complemento de dois
BSDA1-3_BSD	*Binary Signed Digit Adder* com Codificação 1 de 3 e operandos em BSD NAF
BSDA1-3_C2	*Binary Signed Digit Adder* com Codificação 1 de 3 e operandos em complemento de dois
BSDAPar_BSD	*Binary Signed Digit Adder* com Verificação de Paridade e operandos em BSD NAF
BSDAPar_C2	*Binary Signed Digit Adder* com Verificação de Paridade e operandos em complemento de dois
CMOS	*Complementary Metal-Oxide-Silicon*
DSP	*Digital Signal Processing*
LFSR	Linear Feedback Shift Register
NAF	*Non-Adjacent Form* (Forma Não-Adjacente)
PRBS	*Pseudo Random Bit Generator Source*
PTM	*Predictive Technology Model* (Modelo Tecnológico Preditivo)
RCA	*Ripple-Carry Adder*
RCAIOI	*Ripple Carry Adder* com Entradas e Saídas Invertidas
RCATMR	*Ripple Carry Adder* TMR
RESI	Recomputação com Entradas e Saídas Invertidas
SC	Somador Completo
SCCR	Somador Completo com *Carry* Redundante
SD	*Signed-Digit* (Dígito com Sinal)
SEB	*Single- Event Burnout*
SEE	*Single- Event Effect*
SEL	*Single- Event Latchup*
SEGR	*Single- Event Gate Rupture*

SET	*Single-Event Transient*
SHE	*Single Hard Error*
SOC	*System-on-Chip*
SOS	*Systems-on-Silicon* (Sistema Integrado em Silício)
TMR	*Triple Modular Redundancy* (Redundância Modular Tripla)

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Nos últimos anos, os sistemas integrados em silício (SOCs - *Systems-on-Chip*) têm se tornado menos imunes a ruído, em decorrência dos ajustes necessários na tecnologia CMOS (*Complementary Metal-Oxide-Silicon*) para garantir o funcionamento dos transistores com dimensões nanométricas. Dentre tais ajustes, a redução da tensão de alimentação e da tensão de limiar (*threshold*) tornam os SOCs mais suscetíveis a falhas transientes, principalmente aquelas provocadas pela colisão de partículas energéticas que provêm do espaço e encontram-se presentes na atmosfera terrestre. Quando uma partícula energética de alta energia colide com o dreno de um transistor que está desligado, ela perde energia e produz pares elétron-lacuna livres, resultando em uma trilha de ionização. A ionização pode gerar um pulso transiente de tensão que pode ser interpretado como uma mudança no sinal lógico. Em um circuito combinacional, o pulso pode propagar-se até ser armazenado em um elemento de memória. Tal fenômeno é denominado *Single-Event Transient* (SET). Como a tendência é que as dimensões dos dispositivos fabricados com tecnologia CMOS continuem reduzindo por mais alguns anos, a ocorrência de SETs em SOCs operando na superfície terrestre tende a aumentar, exigindo a adoção de técnicas de tolerância a falhas no projeto de SOCs.

O presente trabalho tem por objetivo avaliar circuitos somadores tolerantes a falhas transientes encontrados na literatura. Duas arquiteturas de somadores foram escolhidas: *Ripple Carry Adder* (RCA) e *Binary Signed Digit Adder* (BSDA). O RCA foi escolhido por ser o tipo de somador de menor custo e por isso, amplamente utilizado em SOCs. Já o BSDA foi escolhido porque utiliza o sistema numérico de dígito binário com sinal (*Binary Signed Digit* – BSD). Por ser um sistema de representação redundante, o uso de BSD facilita a aplicação de técnicas de tolerância a falhas baseadas em redundância de informação. Os somadores protegidos avaliados foram projetados com as seguintes técnicas: Redundância Modular Tripla (*Triple Modular Redundancy* - TMR) e Recomputação com Entradas e Saídas Invertidas (RESI), no caso do RCA, e codificação 1 de 3 e verificação de paridade, no caso do BSDA. As 9 arquiteturas de somadores foram simuladas no nível elétrico usando o Modelo Tecnológico Preditivo (*Predictive Technology Model* - PTM) de 45nm e considerando quatro comprimentos de operandos: 4, 8, 16 e 32 bits. Os resultados obtidos permitiram quantificar o número de transistores, o atraso crítico e a potência média consumida por cada arquitetura protegida. Também foram realizadas campanhas de injeção de falhas, por meio de simulações no nível elétrico, para estimar o grau de proteção de cada arquitetura. Os resultados obtidos servem para guiar os projetistas de SOCs na escolha da arquitetura de somador tolerante a falhas mais adequada aos requisitos de cada projeto.

Palavras-Chave: Tolerância a Falhas, SET (*Single-Event Transient*), Circuitos Somadores, Sistema de Numeração de Dígito Binário com Sinal.

Evaluating Delay, Power and Protection of Fault Tolerant Adders

ABSTRACT

In the past recent years, integrated systems on a chip (Systems-on-chip - SOCs) became less immune to noise due to the adjusts in CMOS technology needed to assure the operation of nanometric transistors. Among such adjusts, the reductions in supply voltage and threshold voltage make SOSs more susceptible to transient faults, mainly those provoked by the collision of charged particles coming from the outer space that are present in the atmosphere. When a heavily energy charged particle hits the drain region of a transistor that is at the off state it produces free electron-hole pairs, resulting in an ionizing track. The ionization may generate a transient voltage pulse that can be interpreted as a change in the logic signal. In a combinational circuit, the pulse may propagate up to the primary outputs and may be captured by the output storage element. Such phenomenon is referred to as Single-Event Transient (SET). Since it is expected that transistor dimensions will continue to reduce in the next technological nodes, the occurrence of SETs at Earth surface will increase and therefore, fault tolerance techniques will become a must in the design of SOSs.

The present work targets the evaluation of transient fault-tolerant adders found in the literature. Two adder architectures were chosen: the Ripple-Carry Adder (RCA) and the Binary Signed Digit Adder (BSDA). The RCA was chosen because it is the least expensive and therefore, the most used architecture for SOS design. The BSDA, in turn, was chosen because it uses the Binary Signed Digit (BSD) system. As a redundant number system, the BSD paves the way to the implementation of fault-tolerant adders using information redundancy. The evaluated fault-tolerant adders were implemented by using the following techniques: Triple Module Redundancy (TMR) and Re-computing with Inverted Inputs and Outputs (RESI), in the case of the RCA, and 1 out of 3 coding and parity verification, in the case of the BSDA. A total of 9 adder architectures were simulated at the electric-level using the Predictive Technology Model (PTM) for 45nm in four different bitwidths: 4, 8, 16 and 32. The obtained results allowed for quantifying the number of transistors, critical delay and average power consumption for each fault-tolerant architecture. Fault injection campaigns were also accomplished by means of electric-level simulations to estimate the degree of protection of each architecture. The results obtained in the present work may be used to guide SOS designers in the choice of the fault-tolerant adder architecture that is most likely to satisfy the design requirements.

Keywords: Fault Tolerance, SET (Single-Event Transient), Binary-Signed Digit Number System, Adder Architectures.

1 INTRODUÇÃO

A tecnologia CMOS (*Complementary Metal-Oxide-Silicon*) caracteriza-se por permitir a fabricação de transistores de dimensões extremamente pequenas, o que torna possível integrar uma grande quantidade de transistores em um mesmo *chip* e também alcançar frequências de funcionamento muito elevadas, da ordem de gigahertz. Um dos motivos que levou à adoção da tecnologia CMOS para a materialização de sistemas eletrônicos foi sua contínua evolução ao longo dos anos, caracterizada pelo aumento do número de transistores que podem ser integrados em um *chip*. Para possibilitar este aumento de integração, as dimensões dos transistores tem sido reduzidas de um fator (conhecido por *scaling factor*) a cada 18 a 24 meses. Porém, nos últimos anos, e notadamente, quando as dimensões mínimas dos transistores foram reduzidas abaixo de 180nm, algumas alterações realizadas no processo de fabricação, tais como a redução da tensão de alimentação para valores próximos ou abaixo de 1.0V e da tensão de limiar (*threshold*), resultaram na redução da imunidade dos sistemas integrados em silício (SOSs - *Systems-on-Silicon*) ao ruído, tornando-os suscetíveis a falhas transientes causadas principalmente pela radiação proveniente do espaço, conhecidas como *soft-errors* (COHEN et al., 1999) (MAVIS; EATON, 2002) (BAUMANN, 2005) (RABAEY; CHANDRAKASAN; NIKOLIC, 2003).

A radiação é constituída por partículas energéticas, compostas principalmente de íons provenientes de raios cósmicos, partículas alfa, nêutrons e prótons emitidos nos períodos de atividade solar intensa (BAUMANN, 2005). A principal causa de *soft-errors* é a colisão de tais partículas com a região sensível de um transistor, isto é, com as junções P-N do dreno de um transistor que se encontra em estado desligado (*off*) (BAUMANN, 2005). Caso esta partícula tenha energia suficiente, ela irá penetrar na região ativa, gerando uma trilha de ionização, a qual poderá se estender até o substrato (ou poço, conforme o tipo de transistor). Se esta trilha atingir o substrato, uma corrente elétrica irá se estabelecer. Esta corrente poderá ter amplitude e duração suficientes para carregar (ou descarregar) o nodo, gerando assim, um pulso transiente de tensão, o qual poderá ser interpretado como uma mudança de nível lógico.

Quanto mais próximo da superfície terrestre, menor é a energia contida nas partículas e menor é o fluxo destas (NORMAND; BAKER, 1993). Assim, antes do advento das tecnologias CMOS nanométricas, falhas transientes causadas pela radiação eram uma preocupação apenas para aplicações espaciais. No entanto, com a redução das dimensões dos transistores e consequentemente, das capacitâncias internas dos circuitos integrados, partículas de menor energia passam a poder ocasionar falhas transientes ao colidir com um circuito integrado operando na superfície terrestre (JOHNSTON, 2000) (DUPONT; NICOLAIDIS; ROHR ,2002).

A partícula energética pode colidir com uma região sensível da lógica combinacional ou com uma região sensível da lógica sequencial do circuito integrado (ALEXANDRESCU; ANGHEL; NICOLAIDIS, 2002). Caso a colisão da partícula ocorra em uma região sensível de um elemento de memória, *flip-flop* ou *latch*, o evento transiente poderá causar a inversão indesejada do valor lógico armazenado, ou seja, será gerado um *bit flip*, que só poderá ser corrigido se os dados forem escritos novamente. O tipo de *soft-error* gerado pelo processo de colisão de uma partícula em uma região sensível de um elemento de memória e a consequente inversão do valor lógico nele armazenado recebe o nome de *Single-Event Upset* (SEU) (BAUMANN, 2001) (DODD; MASSENGILL, 2003) (NICOLAIDIS, 2005). Caso a colisão da partícula energética ocorra com uma região sensível de um circuito combinacional, um pulso transiente pode ser gerado. Se este pulso não for mascarado lógica ou eletricamente (OMAÑA et al.; 2003) (BAZE; BUCHNER, 1997) (WIRTH et al., 2005) pelo circuito, ele irá se propagar até alguma saída primária, podendo eventualmente ser capturado por um elemento de memória e assim, ser erroneamente interpretado como um valor lógico correto. O fenômeno de geração de um pulso na lógica combinacional é denominado de *Single-Event Transient* (SET) (ALEXANDRESCU; ANGHEL; NICOLAIDIS, 2002) (VIOLANTE, 2003). Se o SET for capturado pelo registrador de saída da lógica, seu efeito será similar ao de um SEU: inversão de um ou mais bits. SEUs e SETs são classificadas como falhas transientes por não serem defeitos de fabricação e por não causarem nenhum dano físico ao dispositivo onde ocorrem (BUSHNELL; AGRAWAL, 2000) (HEIJMEN; NIEUWLAND, 2006).

Como a tendência de evolução da tecnologia CMOS aponta para uma redução das dimensões dos dispositivos por mais alguns anos, o problema relacionado dos *soft-errors* em circuitos integrados operando na superfície terrestre tende a se agravar. Em especial, considerando que os circuitos integrados atuais são verdadeiros sistemas integrados (SOCs - *Systems-on-Chip*), a proteção contra os *soft-errors* exige técnicas de projeto apropriadas à natureza de cada componente, como memória, lógica de controle, circuitos aritméticos etc.

A proteção contra falhas transientes pode ser realizada através da inclusão de recursos adicionais em tempo de projeto para garantir o funcionamento dos sistemas. Tais recursos têm, normalmente, a forma de elementos redundantes, incluindo componentes de *hardware* ou *software*, redundância de informação ou redundância temporal. A mais popular técnica de redundância de *hardware* é a Redundância Modular Tripla ou TMR (*Triple Modular Redundancy*) (NEUMANN, 1956) e requer três cópias do circuito a ser protegido. A redundância de informação, por sua vez, utiliza bits adicionais para codificar os dados de forma que erros possam ser detectados e posteriormente corrigidos. Códigos de paridade, de *Hamming* e de *Reed-Solomon* são alguns dos códigos de detecção e de detecção/correção de erro utilizados por algumas técnicas de detecção de erro (PETERSON, 1980) (HOUGHTON, 1997) (NICOLAIDIS, 2003). A redundância temporal consiste em amostrar o resultado em tempos distintos, de forma que o intervalo de tempo entre as amostragens seja suficiente para garantir que a falha desapareça (KASTENSMIDT, 2003). Além disto, podem ser projetadas técnicas de tolerância a falhas que utilizam mais de um tipo de redundância. A aplicação de qualquer uma destas técnicas resultará em *hardware* adicional.

Dentre os diversos blocos que compõem os SOCs, os somadores são os circuitos aritméticos de maior importância, uma vez que a adição é a operação aritmética mais comum (RABAEY; CHANDRAKASAN; NIKOLIC, 2003). Além disso, a adição serve

de base para a maioria das demais operações aritméticas, tais como a subtração, a multiplicação e a divisão. Além disso, frequentemente, a estrutura dos operadores aritméticos determina as características de desempenho, consumo de energia/dissipação de potência e confiabilidade (tolerância a falhas) do SOC como um todo. Desta forma, o projeto de somadores que satisfaçam aos requisitos de desempenho, consumo de energia/dissipação de potência e confiabilidade é um tema de alta relevância para a implementação de SOCs. Diversas propostas de somadores tolerantes a falhas são encontradas na literatura. No entanto, a maioria absoluta de tais propostas não apresenta dados referentes à implementação em tecnologia CMOS, tais como área/número de transistores, atraso ou potência/energia. Além disso, o grau de proteção tampouco é avaliado.

Deste modo, o objetivo primordial deste trabalho é realizar a avaliação detalhada de alguns somadores tolerantes propostos na literatura. Duas arquiteturas de somadores foram escolhidas para serem avaliadas e comparadas: *Ripple Carry Adder* (RCA) e *Binary Signed Digit Adder* (BSDA). O RCA é a arquitetura de referência para comparações de desempenho (atraso crítico), uso de recursos de *hardware* e potência, uma vez que é a arquitetura de menor custo (RABAEY; CHANDRAKASAN; NIKOLIC, 2003). O BSDA (PARHAMI, 2002) utiliza o sistema numérico de dígito binário com sinal (*Binary Signed Digit* – BSD). Tal sistema de representação é redundante e possibilita a eliminação da propagação do *carry*, levando aos somadores referenciados como *carry-free*. A escolha desta arquitetura deveu-se à possibilidade de se analisar o impacto do uso de redundância de informação na construção de somadores tolerantes a falhas transientes, impacto este medido em termos de número de transistores, atraso, potência e grau de proteção.

As técnicas de proteção aplicadas aos somadores foram as técnicas de tolerância a falhas que se baseiam em redundância de *hardware*, redundância temporal combinada com redundância de *hardware* e redundância de informação através do uso de codificações de dados. Assim, foram criadas duas versões de RCA protegidos, sendo uma usando Redundância Modular Tripla (*Triple Module Redundancy* - TMR), que é uma técnica de redundância de *hardware*, e outra versão com a técnica de Recomputação com Entradas e Saídas Invertidas (RESI), técnica que combina redundância temporal, redundância de *hardware* e redundância de informação (através de código de paridade). Já ao BSDA foram aplicadas técnicas de proteção que utilizam redundância de informação, explorando assim as características do sistema BSD, bem como as características arquiteturais do somador.

Ao total, foram desenvolvidas 9 arquiteturas de somadores, sendo 6 protegidos e 3 não-protegidos. Estas arquiteturas foram mapeadas em tecnologia CMOS para 4 comprimentos de operandos de entrada (4, 8, 16 e 32 bits), e descritas no formato Spice, originando um total de 36 descrições. Todas as descrições foram simuladas no nível elétrico utilizando o simulador HSpice (SYNOPSYS, 2011), para o Modelo Tecnológico Preditivo (*Predictive Tecnology Model* - PTM) de 45nm (ZHAO; CAO, 2006). Os resultados de atraso crítico e potência média consumida, obtidos por simulação, bem como o número de transistores, são comparados e analisados.

Como não existe nenhuma técnica capaz de garantir um projeto totalmente imune a falhas, também faz-se necessária a análise de qual das técnicas aplicadas aos somadores proporciona a maior proteção com uma menor penalidade, em termos de desempenho e custo de implementação. Para se avaliar o grau de proteção dos somadores, foram realizadas simulações de falha única (um SET por vez), também no nível elétrico,

porém, somente para os somadores de 4 bits, uma vez que as arquiteturas são regulares e os esforços de caracterização para mais bits não agregaria informação relevante.

1.1 Organização da Dissertação

Esta dissertação está organizada da seguinte forma. O capítulo 2 provê o embasamento necessário para a compreensão do restante do trabalho. Assim, o capítulo 2 descreve o sistema numérico de dígito com sinal (*Signed-Digit* - SD), detalhando o subconjunto deste onde a base é 2, o sistema numérico de dígito binário com sinal. Além disto, são detalhados os processos de conversão entre os sistemas numéricos binário e BSD. A adição no sistema BSD é ilustrada. O capítulo 2 também discorre sobre as fontes de dissipação de potência em circuitos CMOS e as consequências do avanço da tecnologia CMOS nos dispositivos construídos com tal tecnologia, em relação a falhas transientes. Também resume as fontes de tais falhas e como estas podem se propagar através dos circuitos. Finalmente, são explicados alguns tipos de codificação de dados utilizados na aplicação da redundância de informação.

O capítulo 3 apresenta as arquiteturas de somadores avaliadas neste trabalho, descrevendo detalhadamente as decisões de projeto tomadas para o mapeamento dos mesmos para tecnologia CMOS. Especial atenção foi dedicada para os somadores BSDA, visto que esta é a arquitetura escolhida para analisar o uso de redundância de informação na proteção contra *soft-errors*. Neste caso, foram explicadas com detalhes as etapas de conversão entre os sistemas de representação, bem como a etapa da soma propriamente dita.

O capítulo 4 apresenta, de forma sistematizada, os resultados dos experimentos práticos. Os resultados de simulação são apresentados e analisados de várias formas, por meio de comparações entre as técnicas de proteção implementadas e também em comparações entre as arquiteturas não-protegidas e protegidas dos somadores RCA e BSDA. Também são analisados o grau de proteção das arquiteturas de somadores protegidos, a partir dos resultados obtidos nas campanhas de injeção de falhas realizadas por meio de simulação no nível elétrico.

Finalmente, no capítulo 5 são apresentadas as conclusões obtidas com a realização deste trabalho.

2 EMBASAMENTO TEÓRICO

Este capítulo apresenta o embasamento teórico necessário para a compreensão das arquiteturas de somadores apresentadas no capítulo 3 cujos resultados de número de transistores, atraso crítico, potência e grau de proteção (tolerância a falhas) são apresentados no capítulo 4.

2.1 Sistemas Numéricos

A primeira forma de representação de números se deu através da criação de símbolos que representavam quantidades. Em aproximadamente 4000 A.C. os egípcios já realizavam operações aritméticas através da soma de símbolos. Porém estes eram diferentes dos conhecidos atualmente. Desde então esses símbolos vem sofrendo modificações onde novos conceitos e melhorias são inseridos no sistema de representação numérica, as quais podem se destacar o sistema criado pelos babilônios em 2000 A.C.. Também surgiram os números romanos na Roma antiga. Estes, porém não são eficientes para representar números grandes e o uso de números representados em romano dificulta a aritmética. Em 400 A.C. surgiram os numerais Brahmi na Índia, os quais foram usados até 400 D.C. Na Figura 2.1 são apresentados os diferentes sistemas numéricos utilizado nos dias atuais.

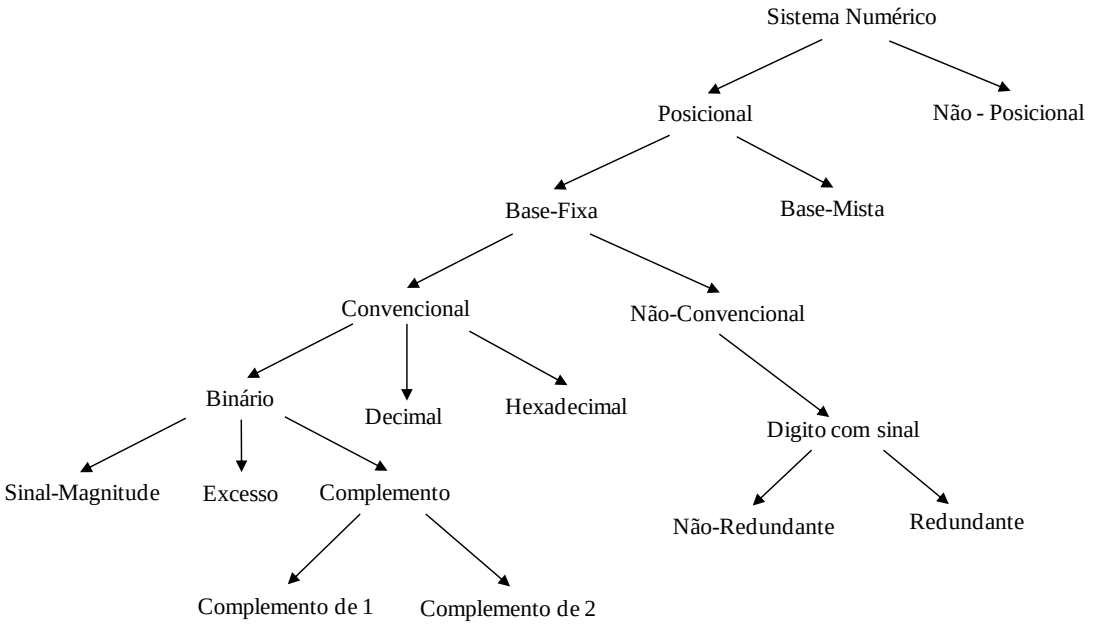


Figura 2.1: Sistemas Numéricos.

Como foi apresentado na figura 2.1, os sistemas numéricos onde a base é fixa podem ser subdivididos em convencionais e não convencionais. O sistema de representação convencional binário é subdividido em **sinal magnitude**, **excesso** e **complemento**. Como o foco do presente trabalho é o sistema numérico não-convencional redundante, o sistema de representação numérica convencional esta detalhados no Apêndice A.

Na representação não convencional os dígitos que representam um determinado número possuem sinal, como por exemplo, em um sistema base 4 que possui o conjunto de dígitos {-2, -1, 0, 1, 2} ao invés de um conjunto de dígitos {0, 1, 2, 3} como seria em um sistema numérico convencional sem dígitos com sinal. Os sistemas numéricos de dígitos com sinal são usados para a eliminação das cadeias de *carries* na adição e também para a aplicação de técnicas de tolerância a falhas, o que será detalhado mais à frente, juntamente com o sistema numérico de dígitos com sinal.

A equação que representa os números em um sistema numérico não convencional de dígitos com sinal com base fixa *r* é semelhante à equação do sistema numérico convencional base fixa *r* sem dígitos com sinal (equação 2). Porém, o que muda é o conjunto de dígitos do sistema numérico, ou seja, enquanto que para um sistema numérico convencional base fixa *r* $x_i \in \{0,1,...,r-1\}$, em um sistema numérico não convencional de dígitos com sinal base fixa *r* $x_i \in \{\overline{r-1},...,0,...,r-1\}$.

O sistema numérico não convencional que apresenta dígitos com sinal pode ser redundante ou não-redundante. Essa propriedade também depende do conjunto de dígitos que forma o sistema numérico. Para o sistema ser não-redundante, o número de dígitos existentes no conjunto de dígitos do sistema deve ser $(|\overline{r-1}|)+(r-1)+1 \leq r$, e para ser redundante o conjunto de dígitos deve conter $(|\overline{r-1}|)+(r-1)+1 > r$ dígitos.

Por exemplo, um sistema base *r* = 10 que apresente o conjunto de dígitos $\{\bar{9},...,0,...,9\}$ é redundante, pois possui 19 dígitos (19 > *r*) em seu conjunto de dígitos, permitindo que cada número tenha mais de uma única representação. Caso o conjunto de dígitos fosse $\{\bar{4},...,0,...,4\}$, o sistema seria não-redundante, pois possuiria apenas 9 dígitos $(9 \leq r)$.

2.1.1 Sistema Numérico de Dígito com Sinal

Os sistemas numéricos base fixa normalmente utilizam um conjunto de dígitos determinado pela base *r* do sistema, ou seja, {0,..., *r-1*}. Porém, é possível a utilização de um conjunto de dígitos mais amplo, como em $\{\overline{(r-1)},...,\bar{1},0,1,...,(r-1)\}$, onde cada dígito pertencente ao conjunto pode ser tanto positivo como negativo (os símbolos $\bar{x}$ representam os dígitos negativos). Tal sistema numérico é conhecido como **sistema numérico de dígito com sinal** (*Signed-Digit* - SD). O sistema numérico SD é um sistema numérico redundante, pois o conjunto de dígitos utilizados tem cardinalidade maior que o valor da base *r*, resultando em mais de uma representação para um mesmo número (KOREN, 2002) (HWANG, 1979) (THORNTON, 1997).

O SD pode utilizar qualquer valor de base. Por exemplo, com base 10 o conjunto de dígitos poderia ser definido como $\{\overline{9},...,\bar{1},0,1,...,9\}$, conduzindo a uma redundância máxima.

Apesar de esta redundância ser altamente benéfica em termos de tolerância a falhas, ela também representa um aumento do custo. Por outro lado, o custo de implementação de circuitos aritméticos que operam com o sistema numérico SD pode ser reduzido (ou controlado) através da restrição do conjunto de dígitos para $x_i \in \{\overline{\alpha},\overline{\alpha-1},...,\bar{1},0,1,...,\alpha\}$, onde α é o valor máximo do conjunto de dígitos, devendo estar entre os valores:

$$\left\lceil \frac{r-1}{2} \right\rceil \leq \alpha \leq r-1 \tag{1}$$

Por outro lado, para obter-se uma redundância mínima α é calculado como

$$\alpha = \left\lfloor \frac{r}{2} \right\rfloor \tag{2}$$

Como são necessários no mínimo *r* dígitos para representar um dígito em um sistema com base *r* e $\overline{\alpha} \leq x_i \leq \alpha$, se obtém $2\alpha+1$ dígitos (KOREN, 2002) (HWANG, 1979). Como este trabalho se concentra no estudo de sistemas digitais que utilizam números binários, as regras do sistema numérico SD serão explicadas para a base 2.

2.1.1.1 Sistema Numérico de Dígito Binário com Sinal

Representação de **dígito binário com sinal** (*Binary Signed Digit* – BSD) é um tipo do sistema numérico de dígito com sinal no qual o valor algébrico de um número *X* de *n* dígitos $(x_{n-1},...,x_0)_{BSD}$ é dado pela equação 3 onde *X* pode ser positivo ou negativo e onde $x_i \in \{\bar{1},0,1\}$ e $\bar{1}$ denota -1.

$$X = \sum_{i=0}^{n-1} x_i \times 2^i \tag{3}$$

O sistema BSD é redundante, pois permite que um inteiro seja representado de várias formas. Isto ocorre porque, enquanto os números binários são representados pelo conjunto de dígitos {0,1}, os números BSD são representados por $\{\bar{1},0,1\}$, o que gera uma quantidade maior de representações. Logo, alguns números binários possuem mais de uma representação equivalente em BSD.

Por ser um sistema redundante, a quantidade de representações BSD que um número inteiro *X* de tamanho *n* possui depende do seu valor algébrico e do seu tamanho (EBEID, 2007), pois como a representação binária do número é considerada uma de suas representações BSD, a obtenção das diferentes representações BSD de *X* pode ser realizada a partir da mesma, trocando 01 por $1\bar{1}$ e $0\bar{1}$ por $\bar{1}1$ e vice versa.

Por exemplo, dado um número $X = (3)_{10}$ representado com $n = 4$ bits, existem cinco representações válidas em BSD que possuem o mesmo valor algébrico, sendo que quatros são obtidas a partir de sua representação binária:

$$X = (0011) = (1\times 2^1 + 1\times 2^0) = (2+1) = 3$$
$$X = (01\bar{1}1) = (1\times 2^2 - 2^1 + 1\times 2^0) = (4-2+1) = 3$$
$$X = (010\bar{1}) = (-1\times 2^2 - 1\times 2^0) = (4-1) = 3$$
$$X = (1\bar{1}0\bar{1}) = (1\times 2^3 - 1\times 2^2 - 1\times 2^0) = (8-4-1) = 3$$
$$X = (1\bar{1}\bar{1}1) = (1\times 2^3 - 1\times 2^2 - 1\times 2^1 + 1\times 2^0) = (8-4-2+1) = 3$$

Essas trocas são realizadas exaustivamente até serem obtidas todas as possíveis representações BSD para o número inteiro. Quanto maior a quantidade de bits que

compõem o número, mais trocas são possíveis e então, maior será a quantidade de representações BSD para cada número. Logo, a quantidade de representações BSD de um número depende do seu valor e da quantidade de bits do mesmo. Porém, existem algoritmos que calculam a quantidade exata de representações BSD que um número inteiro possui, como o algoritmo mostrado em (EBEID; HASAN, 2007).

A redundância do sistema BSD pode ser analisada através do exemplo a seguir. Para um sistema de base 2 com o conjunto de dígitos {-1, 0,1}, n = 3 e uma faixa de representação BSD $\overline{111} \le x \le 111$, que representa 15 números, um sistema BSD é capaz de representar 3^3, ou seja, 27 representações. Então, obtém-se 27 –15 = 12 representações redundantes, o que equivale a 80% de redundância. Logo, alguns dos 15 números possuem mais de uma representação BSD.

Para a implementação de circuitos que utilizam o sistema numérico BSD, bem como para a realização de operações aritméticas, os dígitos BSD devem ser codificados em binário. Então, além da redundância natural existente no BSD são necessários pelo menos dois bits para a representação de cada dígito BSD, o que aumenta ainda mais o grau de redundância deste sistema. A Tabela 2.1 mostra duas possíveis codificações para os dígitos BSD, conhecidas como "Sinal e Valor" e "Flags Negativos e Positivos". Porém existem outras codificações possíveis, como mostram Thornton (1997) e Parhami (2000).

Tabela 2.1: Possíveis codificações para os dígitos BSD

Dígitos BSD	***Sinal e Valor***	***Flags Negativos e Positivos***
0	00	00 ou 11
1	01	01
$\bar{1}$	11	10

A realização de circuitos aritméticos que operam sobre dados representados em BSD requer mais espaço de armazenamento, maior quantidade de bits nos barramentos de dados e maior quantidade de recursos para a realização dos cálculos. Contudo, o ganho em velocidade nas operações aritméticas que utilizam o sistema BSD pode compensar o aumento do custo.

O dígito BSD $\bar{1}$ apresenta duas possibilidades de codificações. Logo, uma destas pode ser considerada como alternativa, ou então pode ser considerada inválida, assim como o dígito 0 na codificação Flags Negativos e Positivos. Neste caso, é perdida capacidade de representação de 25% da redundância gerada pela codificação dos dígitos BSD em complemento de 2. Além disso, são perdidas algumas representações BSD na conversão para complemento de dois, pois em complemento de 2 são possíveis apenas 2^n representações, ao passo que em BSD existem 3^n possibilidades de representações.

A Figura 2.2 apresenta o gráfico que mostra a quantidade de representações BSD (2^{n-1}), representações possíveis com três símbolos (3^n) e representações BSD codificadas em binário para n bits.

Baseado nestas propriedades de redundância do BSD, regras de adição podem ser geradas de modo que, juntamente com um *hardware* específico que implemente tais regras de adição, a propagação do *carry* seja limitada para somente uma posição de dígito, eliminando a propagação do *carry* desde o dígito menos significativo até o mais significativo. Isso torna o tempo de adição independente do comprimento dos

operandos, dispensando ainda um mecanismo explícito para lidar com o sinal do número, pois o mesmo é determinado pelo dígito mais significativo diferente.

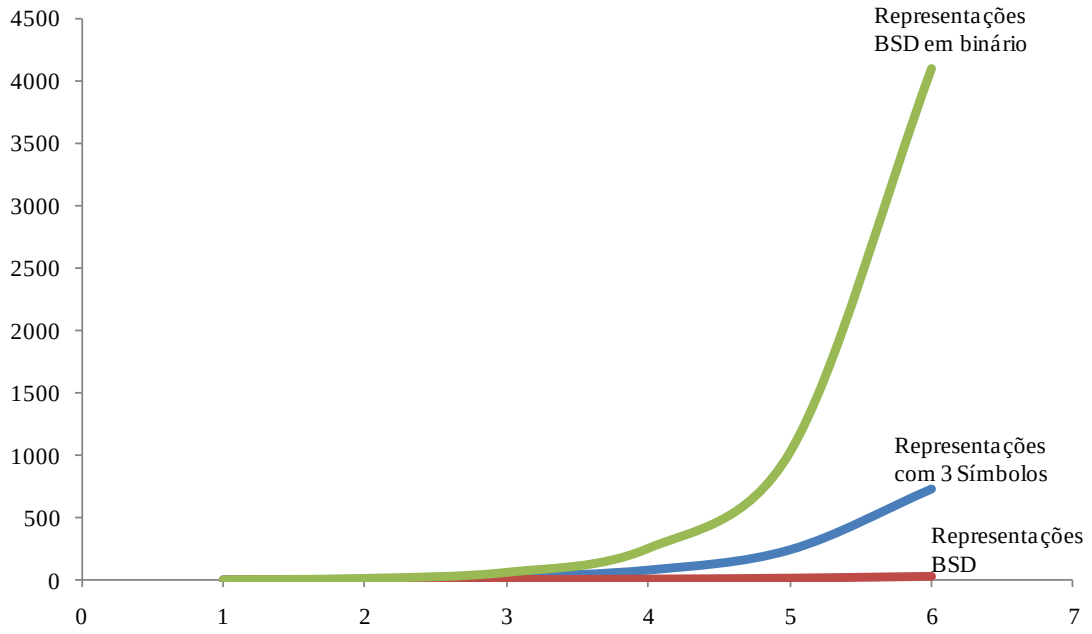


Figura 2.2: Relação entre possibilidades de representações.

O sistema BSD permite a implementação de somadores livres da propagação *carry*, conhecidos como *carry-free adders*. Tal adição é realizada em duas etapas. Na primeira etapa, determina-se um *carry* intermediário $c_i \in \{\bar{1},0,1\}$ e uma soma intermediária $s_i \in \{\bar{1},0,1\}$ para cada dígito que compõe os operandos, de acordo com a equação 4, onde x_i e y_i são os dígitos dos operandos de entrada.

$$x_i + y_i = 2c_i + s_i \tag{4}$$

O *carry* intermediário e a soma intermediária são determinados de maneira que s_i e c_{i-1} não sejam iguais a 1 e nem a $\bar{1}$.

Quando um dos dígitos dos operandos, x_i ou y_i, for 1 e o outro 0, *ci* e *si* são determinados, considerando que tanto $(01)_{BSD}$ como $(1\bar{1})_{BSD}$ representam 1, como:

1. $c_i = 1$ e $s_i = \bar{1}$ no caso de existir a possibilidade de um *carry* igual a 1 vindo da próxima posição menos significativa.
2. $c_i = 0$ e $s_i = 1$ no caso de existir a possibilidade de um *carry* igual a $\bar{1}$ vindo da próxima posição menos significativa.
3. $c_i = 0$ e $s_i = 1$ ou $c_i = 1$ e $s_i = \bar{1}$ no caso de não existir a possibilidade de *carry* vindo da próxima posição menos significativa.

Quando um dos dígitos dos operandos, x_i ou y_i, for $\bar{1}$ e o outro 0, c_i e s_i são determinados como:

1. $c_i = 0$ e $s_i = \bar{1}$ no caso de existir a possibilidade de um *carry* igual a 1 vindo da próxima posição menos significativa.

2. $c_i = \bar{1}$ e $s_i = 1$ no caso de existir a possibilidade de um *carry* igual a $\bar{1}$ vindo da próxima posição menos significativa.

É possível determinar a possibilidade de um *carry* vindo da próxima posição menos significativa através da análise dos bits x_{i-1} e y_{i-1} dos operandos. Quando x_{i-1} e y_{i-1} são iguais a 1, ou quando um é 1 e o outro é 0, existe a possibilidade de um *carry* igual a 1. Caso ambos sejam iguais a $\bar{1}$, ou quando um é $\bar{1}$ e o outro é 0, existe a possibilidade de um *carry* igual a $\bar{1}$. Em outros casos, não existe a possibilidade de propagação de *carry*. Desta forma c_i e s_i podem ser determinados através da análise dos bits x_i, y_i, x_{i-1} e y_{i-1} dos operandos.

Na segunda etapa, é determinada a soma final, $z_i \in \{\bar{1},0,1\}$, através da adição da soma intermediária, s_i, e do *carry* intermediário da próxima posição menos significativa, c_{i-1}, sem geração de *carry*, desde que s_i e c_i sejam determinados como mencionado anteriormente. Desta forma, cada bit da soma final, z_i, pode ser determinado analisando-se os bits x_i, y_i, x_{i-1}, y_{i-1}, x_{i-2}, y_{i-2}.

Deste modo, a propagação do *carry* pode ser eliminada da adição e, portanto, a adição paralela de dois números por um circuito combinacional é executada em um tempo independente do comprimento dos operandos. Isto é, o atraso de um somador binário redundante de *n* dígitos é independente de *n*.

As tabelas que mostram as regras da adição livre de *carry* são apresentadas no capítulo 3 (Tabela 3.4 e Tabela 3.5). Tais tabelas , bem como exemplos de adição livre de *carry* e circuitos que implementam a adição livre de *carry* também podem ser encontrados em (TAKAGI; YASUURA; YAJIMA, 1985) (HARATA et al.,1987) e (THORNTON, 1997).

Para que seja possível a utilização de representações BSD em circuitos digitais é necessário converter os operandos representados no sistema padrão complemento de 2 para o sistema BSD.

A Figura 2.3 mostra as conversões entre os sistemas binário e BSD necessárias para a realização de operações aritméticas. As conversões são realizadas nas entradas e nas saídas. Logo, se entre ambas for executada uma grande quantidade de operações, o impacto das conversões no custo total do *hardware* tende a ser atenuado.

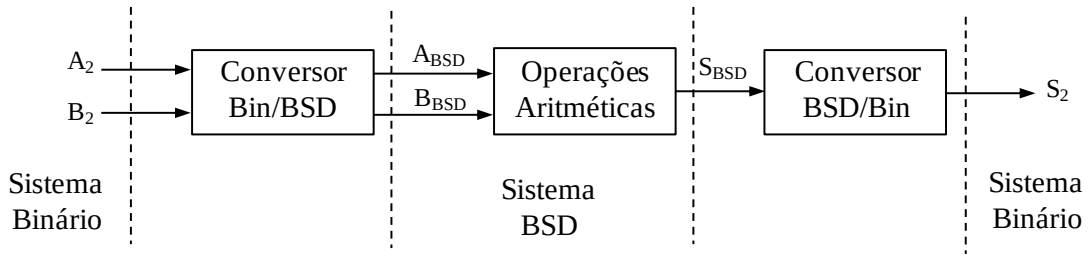


Figura 2.3: Conversões entre os sistemas Binário e BSD.

Um número $X = (x_{n-1},..., x_1, x_0)_2$ representado em binário complemento de 2 pode ser convertido para a sua representação BSD equivalente, ou seja, que possua o mesmo valor numérico, através dos seguintes passos (HWANG, 1979):

1. Para todos os bits x_i convencionais é gerado um dígito temporário d_i:

$$d_i = x_i - 2 \times c_{i+1} \qquad (5)$$

onde c_{i+1} é o dígito de *carry* da próxima posição mais significativa e é definido como:

$$c_{i+1} = \begin{cases} 0, \ se \ x_i < 1 \\ 1, \ se \ x_i \geq 1 \end{cases} \quad (6)$$

2. O i-ésimo dígito de *Y* da representação BSD é obtido a partir de d_i e c_i:

$$y_i = d_i + c_i \quad (7)$$

Portanto, dado um número $X = (11100101)_2$, o qual possui valor algébrico $X = (-27)_{10}$, é obtido o número $Y = (00\bar{1}01\bar{1}\bar{1}1)_{BSD}$. Neste método de conversão, não existe propagação de *carry* e todos os dígitos são gerados independentemente (HWANG, 1979).

A Tabela 2.2 mostra a seqüência de passos realizados para a obtenção de uma representação BSD a partir de sua representação binária correspondente.

Tabela 2.2: Conversão de um número binário para BSD

i	***9***	***8***	***7***	***6***	***5***	***4***	***3***	***2***	***1***	***0***	***Tipo de cada Dígito***
x_i		1	1	1	1	0	0	1	0	1	***Binário convencional***
d_i		$\bar{1}$	$\bar{1}$	$\bar{1}$	$\bar{1}$	0	0	$\bar{1}$	0	$\bar{1}$	***Dígito temporário***
c_i	1	1	1	1	0	0	1	0	1		***carry***
y_i		0	0	0	$\bar{1}$	0	1	$\bar{1}$	1	$\bar{1}$	***BSD***

Este método de conversão pode ser aplicado para converter números em qualquer base *r* para a sua representação *Signal Digit* - SD equivalente. Portanto, generalizando as equações (5) e (6) obtêm-se

$$d_i = x_i - r \times c_{i+1} \quad (8)$$

$$c_{i+1} = \begin{cases} 0, \ se \ x_i < \alpha \\ 1, \ se \ x_i \geq \alpha \end{cases} \quad (9)$$

onde α é o dígito máximo do conjunto de dígitos e deve estar entre os valores $\left| \frac{r-1}{2} \right| \leq \alpha \leq r-1.$

Porém, quando a base 2 é utilizada, como o valor algébrico de uma representação BSD *X* é dado por $X = \sum_{i=0}^{n-1} x_i \times 2^i$, o qual também é usado para obter o valor algébrico de uma representação binária sem sinal, a representação BSD pode ser obtida de forma mais simples, não sendo necessário realizar nenhuma conversão de um número binário sem sinal para obter o número BSD correspondente. Logo, a representação binária de um número pode ser considerada um caso especial de BSD (HWANG, 1979), que não apresenta o dígito $\bar{1}$ e é única (EBEID; HASAN, 2007).

No caso do sistema binário em complemento de 2, onde o valor algébrico de um número *X* é dado por $X = -x_{n-1} \times 2^{n-1} + \sum_{i=0}^{n-2} x_i \times 2^i$ é necessário somente realizar a troca

do bit mais significativo x_{n-1} por $-x_{n-1}$. Considerando o exemplo dado anteriormente, a representação BSD do número $X=(11100101)_2$ obtida desta forma será $Y=(\bar{1}1100101)_{BSD}$.

Portanto, a conversão de uma representação binária para uma representação BSD pode ser realizada de forma direta e o tempo e de conversão é independente do comprimento *n* do número (YEN et al., 1992) (TAKAGI; YASUURA; YAJIMA, 1985).

No entanto, esta conversão pode não ser tão simples caso seja necessário obter um tipo de representação BSD específica, como por exemplo, a Forma Não-Adjacente (*Non-Adjacent Form* – NAF) que será detalhada mais adiante.

A conversão de uma representação BSD para a sua representação binária correspondente é mais complexa que a conversão de uma representação binária para a sua representação BSD e envolve propagação de *carry*, sendo, portanto, bastante lenta, o que pode ser minimizado através do uso de arquiteturas de somadores rápidos.

O método de conversão mais conhecido e utilizado se dá através da subtração de dois números obtidos dos dígitos positivos e negativos da representação BSD e pode ser encontrado em (HWANG, 1979) (YEN et al., 1992) (TAKAGI; YASUURA; YAJIMA, 1985) (HARATA et al.,1987) (KUNINOBU et al., 1987)

$$X_2 = Y_{BSD}^{+} - Y_{BSD}^{-} \tag{10}$$

onde $Y_{BSD}^{+} = \sum_{y_i=1} y_i \times 2^i$ e $Y_{BSD}^{-} = \sum_{y_i=-1} -y_i \times 2^i$, sendo que $y_i \in \{\bar{1},0,1\}$ e os dígitos de Y_{BSD}^{+} e $Y_{BSD}^{-} \in \{0,1\}$.

Por exemplo, a representação BSD mostrada na seção anterior, $Y=(000\bar{1}01\bar{1}\bar{1}\bar{1})_{BSD}$, é convertida novamente na sua representação binária equivalente, $X=(111100101)_2$, da seguinte forma: encontram-se os dois números obtidos a partir da representação BSD $Y_{BSD}^{+}=000001010$ e $Y_{BSD}^{-}=000100101$ e então por meio do complemento de 2 executa a subtração de ambos como segue:

$$\begin{array}{r} 000001010 \\ 111011010 \\ + \qquad\quad 1 \\ \hline 111100101 \end{array}$$

Figura 2.4: Conversões entre os sistemas Binário e BSD.

Yen et al. (1992) apresentaram um circuito que realiza a conversão de uma representação BSD em sua representação binária equivalente. Também é apresentado um novo circuito de conversão que requer menos área e tempo de conversão que o circuito de conversão original.

A Figura 2.5 mostra o circuito conversor, que é implementado como um somador em série.

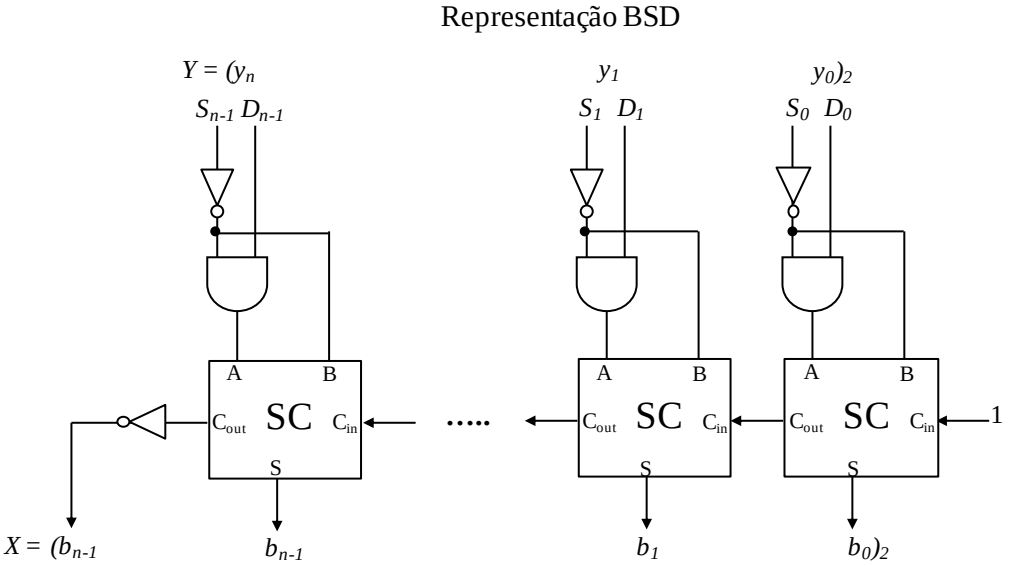


Figura 2.5: Conversor Convencional de Representações BSD para Representações Binárias

No conversor convencional mostrado na Figura 2.5, os dois bits que codificam os dígitos BSD são chamados de S(Sinal) e D(Dígito), sendo que cada um destes bits é usado como operando de entrada dos somadores em série. A saída destes somadores gera a representação binária equivalente aos dígitos BSD de entrada e para que esta conversão seja correta, uma representação BSD de *n* dígitos deve gerar uma representação binária em complemento de 2 de *n+1* bits (YEN et al., 1992).

Maiores detalhes sobre a codificação binária de cada dígito BSD podem ser encontradas em (YEN et al., 1992) (PARHAMI, 2002) (KOREN, 2002).

2.1.1.2 BSD NAF

Como mencionado anteriormente, o sistema BSD é um sistema numérico redundante, ou seja, um determinado valor *X* possui mais de uma representação BSD. Dentre todas as possibilidades de representações BSD de *X*, tem-se interesse na representação BSD mínima.

Tal representação é assim chamada por apresentar menor peso *Hamming* que é o número de dígitos diferentes de zero que a representação possui. Logo, a representação BSD mínima é a representação que apresenta o maior número de zeros, podendo existir mais de uma representação BSD mínima para um dado valor *X*. Maiores detalhes sobre peso *Hamming* podem ser encontrados em (HWANG, 1979) (CHRISTOPHE; DUHAMEL; MAILHES, 2007) (JOYE; YEN, 2000) (GORDON, 1998).

No entanto, a representação BSD mínima mais utilizada é conhecida como Forma Não-Adjacente (*Non-Adjacent Form* – NAF), também conhecida como canônica (ou esparsa) apresentada por Reitwiesner em 1960 (HWANG, 1979) (OKEYA et al., 2004) (JOYE; YEN, 2000).

Uma representação BSD NAF é definida pela propriedade de não possuir dois dígitos adjacentes diferentes de zeros, possuindo o número de zeros aproximadamente

igual a $\frac{2}{3}n$ (EBEID; HASAN, 2007). Logo, em uma representação BSD mínima $Y = y_{n-1},...,y_0$ que contem dígitos não adjacentes diferentes de zero, ou seja, uma representação BSD NAF, todos os dígitos não zero são separados por no mínimo um zero de modo que $y_i \times y_{i-1} = 0$ para todo $1 \le i \le n-1$.

Por exemplo, a representação BSD NAF do número binário usado anteriormente $X = (111100101)_2$ é $Y = (000\bar{1}00101)_{BSD}$, pois *Y* possui peso *Hamming* 3, enquanto *X* apresenta peso *Hamming* 6. Além disso, ambas as representações possuem o mesmo valor algébrico $V = (-27)_{10}$, o que pode ser visto na Tabela 2.3, pois $X = (-1 \times 2^8) + (1 \times 2^7) + (1 \times 2^6) + (1 \times 2^5) + (1 \times 2^2) + (1 \times 2^0) = (-27)_{10}$ e $Y = (-1 \times 2^5) + (1 \times 2^2) + (1 \times 2^0) = (-27)_{10}$.

Tabela 2.3: Representações Binária e BSD NAF de $V = (-27)_{10}$

i	***8***	***7***	***6***	***5***	***4***	***3***	***2***	***1***	***0***
2^i	*256*	*128*	*64*	*32*	*16*	*8*	*4*	*2*	*1*
Binário	1	1	1	1	0	0	1	0	1
BSD NAF	0	0	0	$\bar{1}$	0	0	1	0	1

A representação BSD NAF para qualquer número *Y* com qualquer valor algébrico *V* e um comprimento fixo *n* é única, desde que o produto dos dois dígitos da esquerda em *Y* não seja igual a 1, isto é, $y_{n-1} \times y_{n-2} \neq 1$ para todo $1 \le i \le n-1$ (HWANG, 1979).

Essa propriedade pode ser satisfeita pela adição de um dígito $y_n = 0$ a esquerda da representação de *Y*, gerando representações de *n+1* dígitos, desta forma, ignorando-se os dígitos zeros à esquerda da representação BSD, cada inteiro possui uma única representação BSD NAF.

A representação BSD NAF de um número pode ser obtida a partir da representação binária aplicando-se a conversão $*|1|1 \rightarrow * + 1|0|\bar{1}$ repetidamente, onde * aceita qualquer dígito binário (OKEYA et al., 2004). Porém, o método mais convencional para a conversão de uma representação binária em sua representação BSD NAF é conhecido como Método de Reitwiesner (JOYE; YEN, 2000).

Dado um número binário, $X = (x_{n-1}, \ldots, x_0)_2$, e $x_n = 0$, obtêm um número BSD NAF, $Y = (y_n, \ldots, y_0)_{\text{BSD}}$, de modo que ambos representem o mesmo valor algébrico, seguindo os passos apresentados no pseudocódigo da Figura 2.6 que implementa o método de Reitwiener (HWANG, 1979) (JOYE; YEN, 2000).

Entrada: $X = (x_{n-1}, \ldots, x_0)_2$
Saída: $Y = (y_n, \ldots, y_0)_{\text{BSD}}$
    $c_0 \leftarrow 0$; $x_{n+1} \leftarrow 0$; $x_n \leftarrow 0$;
    Para *i* **de** 0 **até** *n* **faça**
        $c_{i+1} \leftarrow \lfloor (c_i + x_i + x_{i+1})/2 \rfloor$
        yi $\leftarrow c_i + x_i - 2c_{i+1}$
    Fim

Figura 2.6: Algoritmo de codificação para o método de Reitwiesner

O algoritmo verifica os bits da direita para a esquerda e substitui blocos consecutivos de vários 1's por um bloco de 0 e $\bar{1}$ de acordo com $\left(\langle 1\rangle^{n}\right)_2 \mapsto \left(1\langle 0\rangle^{n-1}\ \bar{1}\right)_{BSD}$. Para o caso de um 0 estar isolado entre dois blocos de 1's, o algoritmo usa implicitamente o fato de que $\left(\bar{1}1\right)_{BSD} = \left(0\bar{1}\right)_{BSD}$, ou seja, substitui $\left(\langle 1\rangle^{n} 0\langle 1\rangle^{y}\right)_2$ por $\left(1\langle 0\rangle^{n}\ \bar{1}\langle 0\rangle^{y-1}\ \bar{1}\right)_{BSD}$.

Além disso, o método de Reitwiesner pode ser realizado a partir de uma tabela-verdade, através da qual é possível obter o circuito que implementa a conversão de uma representação binária em sua representação BSD NAF.

Tabela 2.4: Tabela-Verdade para o Método Reitwiesner

c_i	x_i	x_{i+1}	c_{i+1}	y_i
0	0	X	0	0
0	1	0	0	1
0	1	1	1	$\bar{1}$
1	0	0	0	1
1	0	1	1	$\bar{1}$
1	1	X	1	0

Para a implementação do circuito que irá converter os números representados em binário para a sua representação BSD NAF equivalente, os dígitos y_i BSD mostrados na Tabela 2.4 serão representados na codificação Flags Negativos e Positivos. Isto é, os dígitos $\{\bar{1},0,1\}$ serão representados por $\{10,00,01\}$, respectivamente, pois segundo Parhami (2000), esta codificação resulta em *hardware* mais simples e reduz o atraso crítico devido à redução do número de portas lógicas.

A Figura 2.7 mostra o circuito que implementa o Método Reitwiesner.

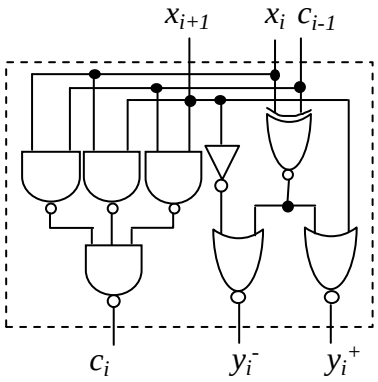


Figura 2.7: Circuito Lógico que implementa o Método Reitwiesner

2.2 Consumo de Potência em Circuitos CMOS

Até alguns anos atrás a atenção dos projetistas de sistemas digitais estava voltada apenas para o desempenho, o custo em área e confiabilidade dos circuitos integrados. Porém, com o avanço da tecnologia CMOS (*complementary metal-oxide-semiconductor)* além destes quesitos, a potência consumida pelos circuitos também passou a ser uma preocupação durante os projetos digitais, pois tal avanço permite dispositivos com tamanhos cada vez menores, o que aumenta a quantidade de

transistores consumindo potência em um único circuito.

A redução dos dispositivos também diminui o atraso de propagação elevando assim a freqüência de operação dos circuitos gerando portanto um aumento no consumo de potência. Existem muitas pesquisas com o objetivo de projetar circuitos com um baixo consumo de potência.

Um excessivo consumo de potência gera um superaquecimento prejudicando o desempenho, a confiabilidade e reduz o tempo de vida útil do circuito integrado (NAJM,1994). Logo, o aquecimento e temperatura do dispositivo influenciam diretamente na potência média consumida pelo mesmo (NAJM,1994). Devido a isso, outra preocupação nos projetos de circuitos integrados em relação ao consumo de potência é a temperatura dos dispositivos.

2.2.1 Fontes de Consumo de Potência em Circuitos CMOS

O consumo de potência em um circuito integrado é definido pela multiplicação da tensão e da corrente do circuito.

$$P = V_{DD} \times I \tag{11}$$

onde P é a potência consumida, V_{DD} é a tensão do circuito e I é a corrente.

O consumo de potência em uma porta lógica determina quanto calor este circuito está dissipando e quanta energia está sendo consumida em um dado momento da operação do mesmo. Em circuitos fabricados com tecnologia CMOS as principais fontes de consumo de potência são: correntes de fuga, que geram o consumo de potência estática, correntes de curto-circuito, cargas e descargas das capacitâncias de saída que ocorrem com a transição do sinal lógico e transições espúrias do sinal conhecidas com *glitching* (RABAEY; CHANDRAKASAN; NIKOLIC, 2003).

Desta forma a potência consumida por circuitos CMOS também pode ser dada como

$$P = P_{est} + P_{cc} + P_{din} + P_{glit} \tag{12}$$

onde P é a potência total consumida pelos circuitos, P_{est} é a potência estática gerada pela corrente de fuga, P_{cc} é a potência gerada pela corrente de curto circuito, P_{din} é a potência dinâmica gerada pela carga e descarga das capacitâncias e P_{glit} é a potência gerada pelos *glitchings*.

2.2.1.1 Consumo de Potência de Curto-Circuito

Quando ocorre uma transição no sinal de entrada de uma porta lógica CMOS, devido aos tempos de subida e descida do sinal não serem simultâneos, durante um pequeno intervalo de tempo, os transistores PMOS (*Positive Channel Metal Oxide Semiconductor*) e NMOS (*Negative Channel Metal Oxide Semiconductor*) conduzem simultaneamente.

Durante este intervalo de tempo existirá uma pequena corrente de curto-circuito saindo da fonte de alimentação indo diretamente para o terra. Tal corrente gera a potência de curto-circuito, que normalmente não é considerada nos circuitos integrados, pois seu valor é aproximadamente 10% da potência dinâmica, P_{din}. (RABAEY; CHANDRAKASAN; NIKOLIC, 2003)

A potência de curto-circuito é dada por (RABAEY; CHANDRAKASAN;

NIKOLIC, 2003)

$$P_{CC} = \frac{\beta}{12} \cdot \left(V_{DD} - V_{thn} - V_{thp} \right)^3 \frac{\tau}{T} \qquad (13)$$

onde β é a dimensão dos transistores, V_{DD} é a tensão de alimentação do circuito, V_{thn} é a tensão de *threshold* dos transistores NMOS, V_{thp} é a tensão de *threshold* nos transistores PMOS, τ é o tempo de subida ou descida do sinal de entrada e T é o período de transição do sinal de entrada.

A Figura 2.8 mostra a corrente de curto-circuito em um inversor durante uma transição do sinal de entrada.

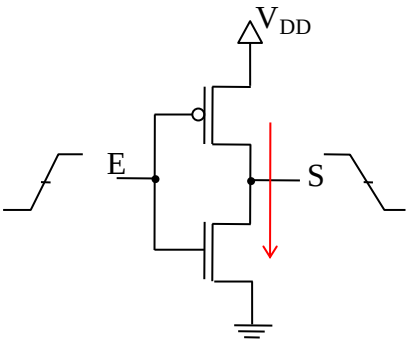


Figura 2.8: Corrente de curto circuito em um inversor

Com o ajuste entre os tempos de subida e descida dos sinais é possível diminuir o consumo da potência de curto-circuito.

2.2.1.2 Consumo de Potência Estática

O consumo de potência estática é gerado pela corrente de fuga e sublimiar existentes nas junções PN dos transistores. Em um circuito CMOS ideal o consumo de potência estática é nulo, pois teoricamente não existe corrente entre a fonte de alimentação e o terra, porém isto não acontece na prática, pois a transição dos sinais nos transistores não é ideal logo, existe corrente de fuga.

As correntes de fuga são geradas quando um transistor se encontra no estado desligado e outro transistor que está ligado carrega o dreno em relação ao potencial do substrato. A corrente de sublimiar ocorre quando existe corrente entre dreno e a fonte quando a tensão entre a porta e a fonte ainda possui um valor menor que a tensão de limiar (RABAEY; CHANDRAKASAN; NIKOLIC, 2003), logo a corrente de sublimiar é exponencialmente dependente da tensão entre a porta e a fonte.

A Figura 2.9 mostra as correntes e fuga presentes em um transistor.

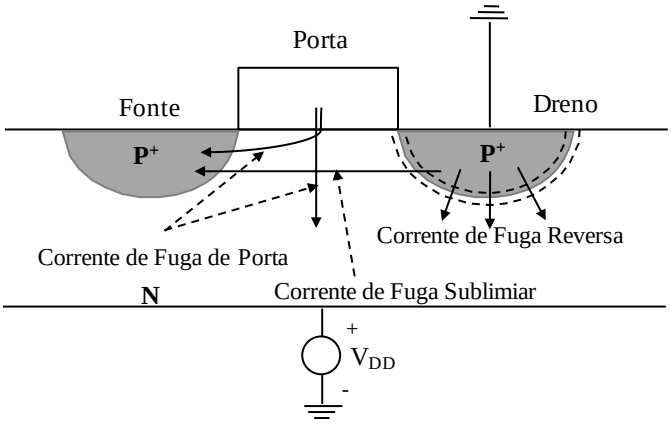


Figura 2.9: Correntes de fuga em um transistor PMOS

Logo a potência estática é dada por (RABAEY; CHANDRAKASAN; NIKOLIC, 2003)

$$P_{fuga} = \frac{V_{DD} \times \left(I_{f0n} + I_{f1p}\right)}{2} \tag{14}$$

onde V_{DD} é a tensão de alimentação do circuito, I_{f0n} é a corrente de fuga nos transistores NMOS com a entrada no nível lógico 0 e I_{f1p} é a corrente de fuga nos transistores PMOS com a entrada no nível lógico 1.

Para as tecnologias mais antigas, a potência estática era insignificante, pois era muito menor que a potência dinâmica. Porém, com a evolução da tecnologia essa parcela da potência passou a ser preocupante, pois a corrente estática aumenta conforme diminui o tamanhos dos transistores, passando a influenciar na potência total dos circuitos (KIM et al., 2003).

2.2.1.3 Consumo de Potência Dinâmica

A atividade de chaveamento ocasiona a carga e descarga das capacitâncias do circuito o que irá gerar o consumo de potência dinâmica. A potência dinâmica depende de quatro fatores: tensão de alimentação, capacitâncias associadas aos nós internos do circuito e atividades de chaveamento dos transistores e principalmente da freqüência de operação.

Para as tecnologias mais antigas, a potência dinâmica era a principal fonte de consumo da potência total do circuito. Porém, para tecnologias mais atuais, esta pode chegar a ser menor que o consumo de potência estática (KIM et al., 2003).

Para analisar o consumo de potência dinâmica, pode ser utilizado um circuito inversor (Figura 2.10) que é mais simples que os demais circuitos, mas que apresenta comportamento semelhante às demais portas lógicas CMOS, em relação à potência dinâmica.

A capacitância de carga existente na saída do inversor pode ser representada por $C_L = C_1 + C_2$, sendo C_1 a capacitância de carga ligada na fonte de alimentação e C_2 a capacitância ligada ao terra.

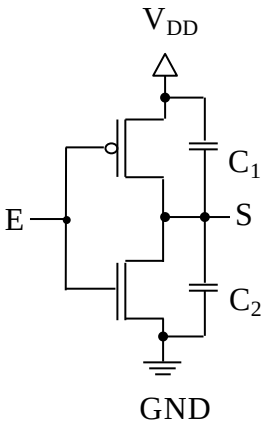


Figura 2.10: Capacitâncias de saída de um inversor

Quando o sinal de entrada do circuito encontra-se no nível lógico baixo, a saída estará em nível lógico alto, ao alterar o sinal da entrada para nível lógico alto a saída passará para nível lógico baixo e as cargas dos capacitores da saída serão alteradas. o capacitor C_1 será carregado, ou seja, a carga de C_1 irá alterar de 0 para $C_1 \times V_{DD}$, enquanto que a capacitância *C2* será descarregada para o terra, isto é, a carga de C2 irá alterar de $C_2 \times V_{DD}$ para 0, pois o transistor NMOS estará em curto circuito.

Quando a entrada for novamente alterada para o nível lógico baixo C_1 será descarregado, pois o transistor PMOS estará em curto circuito, e C_2 será carregado novamente até atingir $C_2 \times V_{DD}$. Desta forma, durante um ciclo completo de chaveamento dos valores de entrada a carga total do circuito será $C_L \times V_{DD}$.

Logo a potência dinâmica consumida pelo circuito dependerá de quantas vezes as capacitâncias C_1 e C_2 serão carregadas e descarregadas, ou seja, a potência dinâmica depende da freqüência com a qual C_1 e C_2 serão carregadas e descarregadas.

A potência dinâmica é dada por (RABAEY; CHANDRAKASAN; NIKOLIC, 2003)

$$P_{din} = \alpha \times f \times C_L \times V_{DD}^{\ 2} \qquad (15)$$

onde α é a probabilidade de chaveamento de todas as portas lógicas do circuito, f é a freqüência de operação do circuito, C_L é a capacitância de carga do circuito e V_{DD} é a tensão de alimentação do circuito.

2.2.1.4 Consumo de Potência devido ao efeito de Glitching

Devido ao fato de as portas lógicas possuírem um atraso de propagação diferente de 0 (RABAEY; CHANDRAKASAN; NIKOLIC, 2003), quando ocorre uma transição de sinal surgem nas portas lógicas transições espúrias e transitórias chamadas de *glitches* (ou *hazards*).

A quantidade de *gliches* que pode ocorrer em um circuito depende do atraso do mesmo, pois um nó do circuito pode exibir múltiplas transições espúrias em um único pulso de relógio antes de apresentar o valor lógico correto na saída. Isto irá dificultar o cálculo do consumo de potência devido ao efeito de *glitching*, pois este depende da profundidade lógica do circuito e do modelo de atraso utilizado na simulação.

A geração de um *glitch* na saída de uma porta e a propagação de um *glitch* através

de uma porta são as principais contribuições dos *glitches* para o consumo de potência dos circuitos CMOS. A Figura 2.11 mostra a geração e a propagação de um *glitch* através de um circuito.

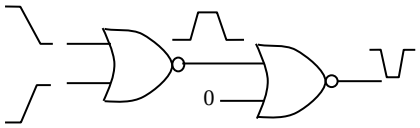


Figura 2.11: Geração e propagação de *glitches*

Maiores detalhes sobre consumo de potência podem ser encontrados em Rabaey (2009), Rabaey, Chandrakasan e Nikolic (2003) e Keating (2007) .

2.3 Falhas Transientes

Com o avanço das tecnologias de fabricação dos circuitos integrados CMOS (*Complementary Metal-Oxide-Silicon*), questões referentes à confiabilidade e à robustez dos sistemas eletrônicos tem se tornado preocupantes. Essa preocupação se deve à redução drástica das dimensões dos transistores, a qual tem por objetivo permitir maiores densidades de integração e velocidades de operação mais altas, à redução da tensão de alimentação e da tensão de limiar (*threshold*), menores capacitâncias e também freqüências de relógio cada vez mais elevadas.

Estes fatores têm contribuído para reduzir a imunidade dos circuitos a ruído, podendo permitir que pequenas variações de tensão sejam interpretadas como inversões do sinal lógico, de modo que está cada vez mais difícil garantir que os circuitos fabricados com tecnologias CMOS estado-da-arte irão operar de maneira confiável durante todo o período de vida útil. Logo, outra conseqüência importante destes ajustes é o aumento da suscetibilidade dos circuitos a falhas transientes, também conhecidas como *soft errors* (COHEN et al., 1999) (BAUMANN, 2001) (MAVIS; EATON, 2002)

Uma falha pode ser classificada como permanente transiente ou intermitente. Uma falha permanente é aquela que existirá indefinidamente se alguma ação corretiva não for tomada. Uma falha transiente é aquela que aparece e desaparece no sistema durante um curto período de tempo e não causa danos permanentes aos dispositivos onde ocorre. Já a falha intermitente aparece, desaparece e reaparece repetidamente.

Uma falha transiente pode ser decorrente de um evento de radiação, ou seja, quando uma partícula energética colide com um dispositivo causando um distúrbio de carga suficiente para reverter o estado lógico de um sinal ou mesmo dos dados de um elemento de armazenamento.

Um dos efeitos induzidos por irradiação mais preocupantes para aplicações terrestres comerciais são os *single-event effects* (SEEs). Estes podem ser classificados nas seguintes categorias: *Soft SEEs* (*Single-Event Transienst* (SETs) e *Single Event Upsets* (SEUs)), *Hard SEEs* (Single Hard Errors (SHEs)), e *Destructives SEEs* (*Single Event Latchups* (SELs), *Single Event Burnouts* (SEBs) e *Single Event Gate Ruptures* (SEGRs)).

O foco do presente trabalho é a análise da proteção de circuitos somadores contra *Soft SEEs* (que são *soft errors*), do tipo *Single Event Transients*, ou simplesmente, SETs.

2.3.1 Fontes de Falhas Transientes

A principal fonte de falhas transientes nos circuitos integrados é a radiação (BAUMANN, 2001) (BAUMANN, 2005). No espaço, a atividade solar é a principal causa da radiação, a qual é constituída por partículas energéticas, tais como elétrons, prótons ou íons energéticos, e também pela radiação eletromagnética, a qual pode ser constituída por raios *X*, raios gama ou luz ultravioleta (BARTH, 1997) (BAUMANN, 2001).

Ao nível do mar três mecanismos diferentes geram as partículas energéticas responsáveis por induzir os *soft errors*: as partículas alfa, oriundas de impurezas presentes no encapsulamento e no próprio circuito integrado, nêutrons, gerados pela interação entre raios cósmicos de alta energia com átomos da atmosfera terrestre e raios cósmicos de baixa energia (NORMAND, 1996) (BAUMANN, 2005).

Quanto mais próximo da superfície terrestre, menor é a energia das partículas (NORMAND; BAKER, 1993). No entanto, como mencionado anteriormente, os circuitos integrados fabricados nas tecnologias CMOS mais recentes permitiram transistores com dimensões muito reduzidas. Logo, as capacitâncias dos nós internos a tais circuitos são muito pequenas, o que os deixa mais vulneráveis aos efeitos transientes ocasionados por partículas com menor energia (JOHNSTON, 2000) (DUPONT; NICOLAIDIS; ROHR ,2002).

2.3.2 Geração do Pulso Transiente e Mecanismo de Charge Colection

As junções P-N dos drenos dos transistores que estão desligados (em estado *off*) constituem-se nas regiões sensíveis dos transistores (BAUMANN, 2001). Quando um íon energético colide com uma destas regiões sensíveis, uma trilha cilíndrica de pares elétron-lacuna e uma elevada concentração de portadores são formados seguindo a passagem do íon, conforme ilustra a Figura 2.12a. Quando a trilha de ionização resultante atravessa ou se aproxima da região de depleção (região vazia de cargas), portadores são coletados rapidamente pelo campo elétrico, que cria uma corrente/tensão transiente nesse nó. Uma característica notável deste evento é que a região de depleção toma forma de um funil. Este funil aumenta a eficiência da coleta da carga devido ao aumento da região de depleção dentro do substrato, o que é ilustrado pela Figura 2.12b. O tamanho deste funil é função da dopagem do substrato – o funil é maior para substratos menos dopados. Esta fase da coleta ocorre dentro de 1ns (BAUMANN, 2005) e é seguida pela fase onde a difusão começa a dominar o processo da coleta (Figura 2.12c).

A carga adicional é coletada enquanto os elétrons se difundem na região de depleção em uma escala de tempo mais longa (centenas de ns), até todos os portadores adicionais serem coletados, recombinados, ou difundidos pela área da junção. No geral, quanto mais distante da junção o evento ocorre, menor a quantidade de carga que será coletada, sendo menos provável que a colisão cause um soft error (BAUMANN, 2005).

Na Figura 2.12 a curva representa o pulso transiente real, gerado pela colisão da partícula energética com o dispositivo. Porém, para analisar a suscetibilidade dos circuitos combinacionais a falhas transientes e a propagação de tais falhas através dos circuitos, é necessário modelar matematicamente o pulso gerado.

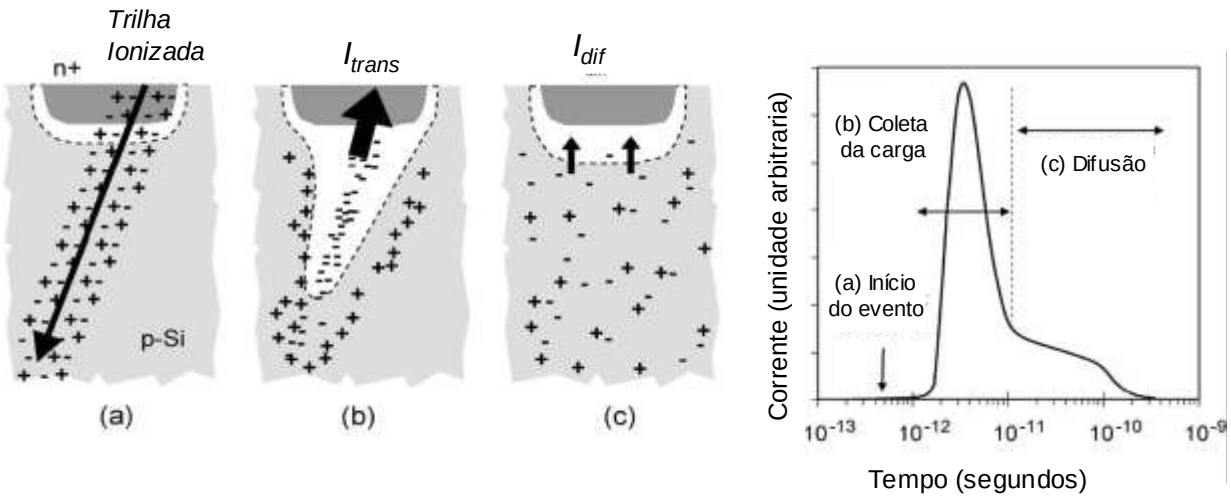


Figura 2.12: Fases da coleta da carga gerada pela colisão e o pulso gerado

Em 1982, Messenger modelou o pulso gerado na saída de uma porta lógica pela colisão de uma partícula através de uma fonte de corrente cujo comportamento obedece a uma dupla exponencial. Outros modelos também foram propostos em (DAHLGREN; LIDEN, 1995) (OMAÑA et al., 2003) (ALEXANDRESCU; ANGHEL; NICOLAIDIS, 2004) (WIRTH et al., 2005) porém não serão detalhados pois fogem ao escopo deste trabalho.

2.3.3 SEUs e SETs

O que define o tipo do SEE é a região em que a partícula energética colide. A partícula pode colidir com uma região sensível da lógica combinacional ou com um elemento de memória (ALEXANDRESCU; ANGHEL; NICOLAIDIS, 2002).

Quando uma partícula colide com o dreno de um transistor PMOS que se encontra em estado *off* em uma célula de memória SRAM, o pulso de tensão gerado pode carregar a capacitância associada a este dreno, ligando assim o transistor por ele controlado. Esta descarga de capacitor poderá ser interpretada como uma mudança de nível lógico. Caso a partícula energética colida com o dreno do transistor NMOS que se encontra em estado *off*, o efeito é similar, exceto que a capacitância associada ao dreno será descarregada. Em ambos os casos, o pulso de tensão induzido acaba por causar a inversão indesejada do valor lógico armazenado na célula de memória. Ou seja, pode ocorrer um *bit-flip* na célula de memória.

O fenômeno da colisão de uma partícula energética em uma região sensível de um elemento de memória e a conseqüente inversão do valor lógico nela armazenado recebe o nome de *Single-Event Upset*, ou SEU (BAUMANN, 2001) (DODD; MASSENGILL, 2003) (NICOLAIDIS, 2005). Quando uma partícula energética colide com uma região sensível de um circuito combinacional, um pulso transiente pode ser gerado. Neste caso, o fenômeno é denominado Single-Event Transient ou SET (ALEXANDRESCU; ANGHEL; NICOLAIDIS, 2002) (VIOLANTE, 2003). Se o SET se propagar até

alguma saída primária, ele poderá ser capturado por um elemento de memória e assim, ser interpretado como um valor lógico correto, gerando assim um soft error.

Existem três tipos de mascaramentos que podem impedir que o pulso gerado por um SET se propague pela lógica do circuito combinacional e atinja um elemento de memória: mascaramento lógico, mascaramento elétrico e mascaramento por *latching window* (KASTENSMIDT, 2003) (WIRTH et al., 2005). Mais detalhes sobre mascaramento podem ser encontrados em (GOLDSTEIN, 1979) (ABRAMOVICI; BREUER; FRIEDMAN, 1990) (BUSHNELL; AGRAWAL, 2000) (CHA et al., 1996) (SHIVAKUMAR et al., 2002) (ANGHEL; LEVEUGLE; VANHAUWAERT, 2005) (WIRTH et al., 2005).

SEUs e SETs são classificadas como falhas transientes (*soft errors*), uma vez que não se originam de defeitos de fabricação e tampouco causam algum dano ao dispositivo onde ocorrem (BUSHNELL; AGRAWAL, 2000) (HEIJMEN; NIEUWLAND, 2006).

2.3.4 Propagação do SET

Caso um SET ocorra em uma porta que não esteja conectada diretamente a um elemento de memória, é necessário então avaliar se o pulso gerado irá ou não se propagar até o elemento de memória da saída do circuito. A Figura 2.13 mostra um circuito que se encontrava estável sob o vetor de entrada "101" no momento em que um SET ocorreu na porta NAND P6, gerando um pulso 1→0→1 na saída desta porta.

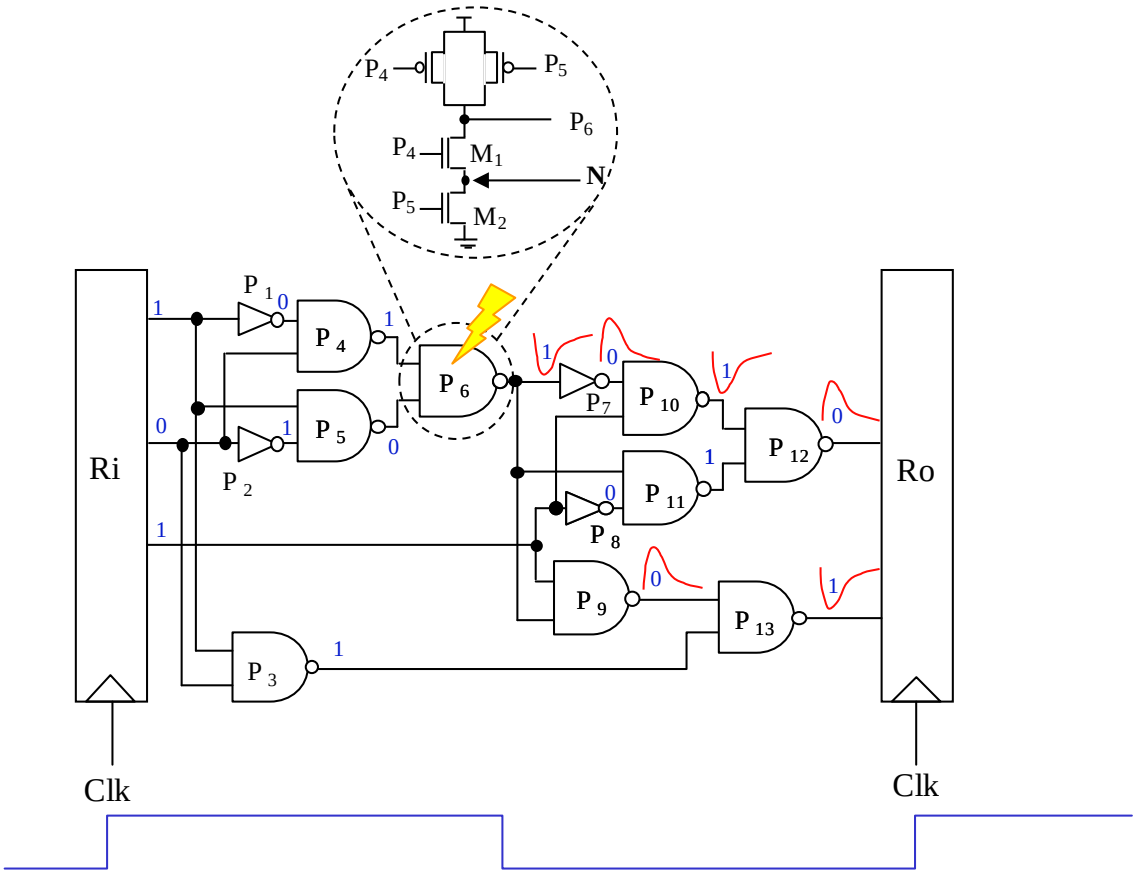


Figura 2.13: *Single-Event Transient* (SET) e sua possível propagação em um bloco combinacional

A porta NAND P6 tem um nó interno N, mostrado na Figura 2.13. Para este nó ser sensível à colisão de uma partícula energética, o transistor M2 deve estar em estado *off*, e para propagar o pulso gerado em N para a saída da porta P6, o transistor M1 deve estar em estado *on*. Esta condição pode ser satisfeita apenas quando as saídas das portas P4 e P5 forem 1 e 0, respectivamente, como ilustra a Figura 2.13.

Para saber a probabilidade do pulso gerado em N se propagar para a saída de P6 é necessário conhecer todos os vetores de entrada que farão com que N seja sensível à colisão de uma partícula energética. Neste caso, os vetores “100” e “101” são os únicos vetores que tornam o nó N sensível e o pulso gerado neste pode propagar-se para a saída da porta P6 (GILL et al., 2005) e através do circuito.

Examinando a Figura 2.13 pode-se inferir que a probabilidade de um pulso se propagar até alguma saída primária tende a ser inversamente proporcional ao número médio de níveis lógicos entre o ponto em que o pulso foi gerado e as saídas primárias. Ou seja, quanto maior o número de portas lógicas que o pulso precisa atravessar, maior é a probabilidade de ele ser completamente mascarado lógica ou eletricamente.

Assim, a análise da propagação de SETs em circuitos combinacionais é útil para identificação dos pontos mais sensíveis, permitindo identificar quais partes precisam ser protegidas contra radiação. Os métodos existentes para a análise da suscetibilidade de circuitos combinacionais a SETs podem ser classificados em métodos baseados em simulação e métodos baseados em propagação topológica. Porém, o estudo de tais métodos foge ao escopo deste trabalho.

Maiores detalhes sobre falhas transientes podem ser encontrados em (LISBOA, 2009) (BASTOS, 2006) (KASTENSMIDT et al., 2004) (ASSIS, 2009).

2.4 Técnicas de Projeto para Tolerância a Falhas

Tolerância a falhas pode ser realizada de por meio de várias técnicas que garantam o funcionamento correto do sistema, mesmo na ocorrência de falhas. Mascaramento é uma das abordagens de tolerância a falhas. Outra abordagem é a Detecção e Localização de Falhas, com a Reconfiguração do sistema. O objetivo da tolerância a falhas é garantir a confiança e a qualidade de serviços oferecidos por sistemas computacionais.

Se em um sistema algum tipo de tolerância a falhas ou detecção de falhas for necessário, então alguma forma de redundância também será necessária. O conceito de redundância se baseia na adição de informações, recursos ou tempo além do necessário para a operação normal do sistema. Porém, deve-se compreender que redundância pode ter um impacto importante no sistema, em termos de área, peso, desempenho, consumo de potência, confiabilidade, e outros.

2.4.1 Redundância de *Hardware*

A replicação física do *Hardware* é a forma mais comum de redundância usada nos sistemas. O aumento da densidade de integração dos componentes semicondutores e a redução de seu custo favorecem o uso desta técnica. Existem três tipos básicos de redundância de *hardware*: redundância passiva (ou estática), dinâmica (ou ativa) e híbrida. Maiores detalhes sobre estas são encontrados em (PRADHAN, 1996) e (LALA, 2001).

A forma mais comum de redundância de *Hardware* é a técnica NMR (*n-modular redundancy*) (PRADHAN, 1996) que consiste na utilização de n módulos iguais que realizam a mesma computação. Para a determinação da resposta correta é utilizado um votador. Essa técnica considera que a probabilidade de mais de um módulo apresentar falhas durante a computação é muito pequena e desse modo, a confiabilidade do sistema seria aumentada. Essa consideração exige que os módulos sejam independentes no que diz respeito às falhas.

A Redundância Modular Tripla ou TMR (*Triple Modular Redundancy*) é aplicação mais comum da técnica NMR. O conceito básico da TMR consiste em triplicar um módulo que se deseja proteger e aplicar suas saídas a um votador (Figura 2.14). Este votador reproduz em sua saída o valor lógico correspondente a pelo menos duas de suas entradas. Assim, mesmo que um dos módulos redundantes apresente alguma falha, os resultados dos outros dois módulos irão mascará-la na saída do votador.

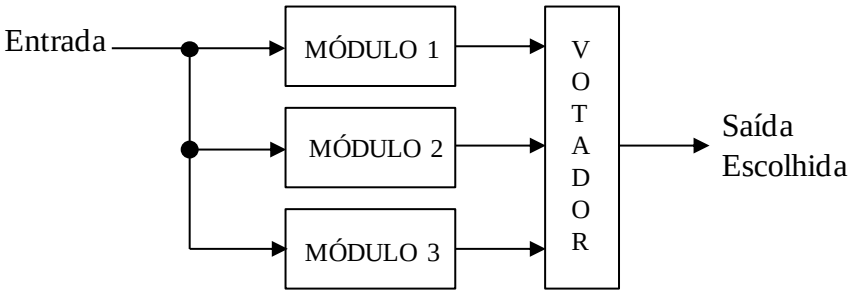


Figura 2.14: Redundância Modular Tripla (TMR).

O votador não tem a função de detectar qual módulo redundante que realizou a computação de forma errônea. Se o sistema necessitar desta detecção, um circuito detector de falhas deverá ser implementado, ou seja, a simples aplicação da técnica TMR não será suficiente. A Figura 2.15 mostra o circuito votador

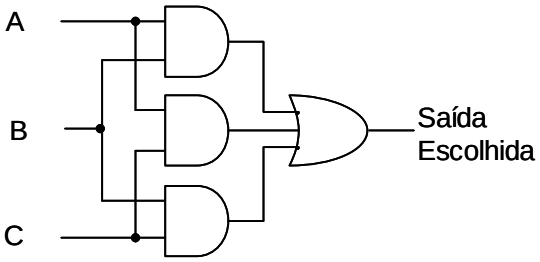


Figura 2.15: Circuito Votador

A principal desvantagem desta técnica é o significativo acréscimo de *hardware*, o qual é normalmente estimado como sendo maior que 200%.

Na técnica TMR uma falha que ocorre em qualquer um dos módulos redundantes poderá ser mascarada. Porém, uma falha no votador não poderá ser mascarada, pois nesta parte do sistema não existe nenhum tipo de redundância, resultando em uma falha em todo o sistema. Logo, o votador é um ponto único de falhas (*single-point of failure*) da técnica TMR. Além disto, se dois módulos apresentarem o mesmo erro, o votador escolherá um resultado incorreto.

Logo, para aumentar a confiabilidade do votador outras soluções precisam ser implementadas, tais como construir votadores com componentes de alta confiabilidade, triplicar o votador ou realizar a votação por meio de *software* (PRADHAN, 1996).

Outra solução possível para se obter tolerância a falhas e amenizar o acréscimo de *hardware* é proteger apenas os pontos mais vulneráveis dos circuitos integrados, ou seja, triplicar apenas as regiões mais sensíveis do circuito.

Como as células de memória se constituem em um dos pontos mais sensíveis dos circuitos integrados, a técnica TMR pode ser aplicada a tais componentes, para aumentar a tolerância destes a SEUs.

2.4.2 Redundância de *Software*

Muitas técnicas de Detecção de Falhas e de Tolerância a Falhas podem ser implementadas em *Software*. Redundância de *Software* pode ser implementada de várias formas. Na redundância de *software* diferentes versões do mesmo *software* são necessárias. A simples replicação de módulos idênticos é uma técnica inútil em *software*, uma vez que módulos idênticos de *software* irão apresentar erros idênticos. (PRADHAN, 1996).

Programação com N-Versões (*N-version programming*), Blocos de Recuperação (*Recovery Blocks*), Verificação de Consistência (*Consistency Checks*) e Verificação de Capacidade (*Capability Checks*) são exemplos de algumas técnicas que utilizam Redundância de *Software* (PRADHAN, 1996) (LALA, 2001).

Redundância de *Software* foge ao escopo deste trabalho.

2.4.3 Redundância de Tempo

Redundância temporal é uma opção de implementação que busca diminuir *hardware* necessário para realizar a detecção de falhas ou tolerância a falhas, utilizando tempo adicional. Em muitas aplicações o tempo é menos importante do que o *hardware*, por este último ser uma entidade física que gera impactos no tamanho, peso, energia consumida e custo. Logo, o tempo pode ser explorado nestas aplicações.

Redundância de Tempo é normalmente usada para detectar e corrigir erros causados por falhas temporárias e envolve a repetição de instruções, segmentos de programas ou programas inteiros (LALA, 2001), mas também pode ser utilizada para detectar falhas permanentes.

TR (*Time Redundancy*) corresponde à aplicação da redundância temporal pura, ou seja, em vez de triplicar o bloco a ser protegido, faz-se uma amostragem do resultado em três momentos diferentes. A diferença de tempo entre as amostras do resultado deve garantir o desaparecimento da falha, caso esta ocorra. Para a implementação desta técnica, mostrada na Figura 2.16, é necessária a triplicação de elementos de armazenamento para a obtenção das três amostras do resultado, que serão usadas como entrada do votador.

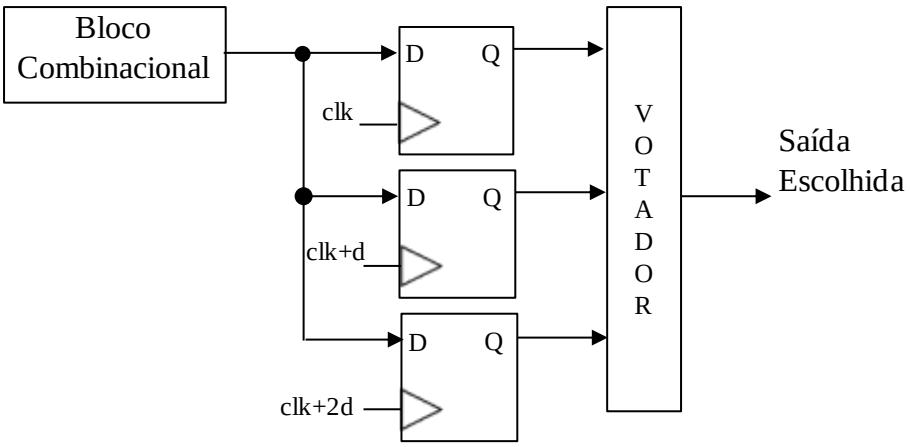


Figura 2.16: Redundância Temporal utilizada para mascarar falhas transientes

Nesta técnica o aumento de *Hardware* não é tão grande quanto na técnica TMR. Porém, esta técnica apresenta certa complexidade devido ao fato dos elementos de memória possuírem diferentes momentos de disparo. Logo, esta técnica possui uma desvantagem em relação ao tempo necessário para realização da computação, o qual será aproximadamente igual a *clk+2d+tp*, onde *tp* é o atraso do votador (KASTENSMIDT, 2003). Logo, não é ideal para ser utilizada em aplicações onde o tempo é um fator importante (LALA, 2001).

A combinação da redundância de *Hardware* e da redundância temporal é uma alternativa que tenta amenizar as desvantagens em relação ao aumento de *Hardware* provocado pela redundância de *Hardware* pura e melhorar o desempenho que é reduzido pela aplicação da redundância temporal pura. Um exemplo da aplicação desta técnica é o método que procura proteger um bloco combinacional contra SETs realizando a duplicação deste e utilizando um estágio CWSP (*Code Word State Preserving*) encontrado em (ANGHEL; ALEXANDRESCU; NICOLAIDIS, 2000).

2.4.4 Redundância de Informação

A redundância de informação consiste na duplicação dos dados ou armazenamento de informação redundante que permite a detecção e mascaramento de falhas e possibilita a tolerância a falhas. Redundância de informação é obtida por meio de Códigos de Detecção e de Correção de Erros (*Error Correction and Detection Codes*).

A codificação adiciona bits extras de verificação ao dado, permitindo verificar a exatidão dos dados antes de usá-los e, em alguns casos, permitindo também a correção dos bits do dado que contém erro.

Quando a codificação é aplicada, um dado de *n* bits é codificado em uma palavra de código de *c* bits, que consiste em um número maior de bits que o dado original. Uma conseqüência desta redundância de informações é que nem todas as 2^c combinações são palavras de código válidas. Desta forma, ao decodificar o resultado codificado da computação para extrair o dado original pode ser encontrada uma palavra de código inválida, o que irá indicar que um erro ocorreu.

Um código é definido como o conjunto de todas as palavras de código possíveis. Os parâmetros para medir o desempenho de um código são o número de bits errados que podem ser detectados e o número de erros que podem ser corrigidos. O overhead imposto pelo código é medido em termos dos bits adicionais que são necessários e o

tempo necessário para codificar e decodificar os dados originais. A distância *Hamming* é a métrica de espaço da palavra de código. Mais detalhes sobre esta métrica são encontrados em Koren e Krishna (2007).

Outra importante propriedade dos códigos é a separabilidade. Um código separável tem campos separados para os bits de dados e os bits de verificação. Portanto, a decodificação para um código separável simplesmente consiste em selecionar os bits de dados e ignorar os bits de verificação. Os bits de verificação devem ainda ser processados separadamente para verificar a correção dos dados. Um código não separável tem os bits de dados e de verificação integrados juntos, e a extração do dado da palavra codificada requer algum processamento, o que gera um tempo adicional para a obtenção do resultado.

Existem vários Códigos de Detecção e Correção de Erros. Dentre eles os principais são: Duplicação, Paridade, *Checksum*, Código M de N, Código Berger, Códigos Cíclicos, Códigos Aritméticos, *Hamming* e *Reed-Solomon* (KOREN; KRISHNA, 2007) (PRADHAN, 1996) (LALA, 2001).

2.4.4.1 Códigos de Paridade

Provavelmente o código mais simples entre todos os códigos, seja o Código de Paridade. Na sua forma mais simples, uma palavra codificada com paridade inclui *n* bits de dados e um bit *p*, que indica a paridade. Este bit *p* irá indicar se a quantidade de bits iguais a 1 pertencentes aos *n* bits de dados é par (p=0) ou impar (p=1).

A fração de overhead do Código de Paridade é 1/*n* e sua distância *Hamming* é igual a 2, ou seja, é garantida a detecção de todos os *single-errors*. Isto porque se um bit trocar de 0 para 1 (ou de 1 para 0), a paridade também será alterada. No entanto, paridade simples não pode corrigir nenhum bit com erro.

Muitas variações do código de paridade simples foram propostas e implementadas. Koren e Krishna (2007) detalham e exemplificam algumas destas variações.

2.4.4.2 Checksum

Checksum é um valor calculado o qual permite verificar a validade de algo. Normalmente, *Checksums* são usados em contextos de transmissão de dados através de canais de comunicação, para detectar se os dados foram transmitidos com êxito. *Checksum* pode ser realizado de várias maneiras, dependendo da natureza da transmissão e confiabilidade necessária.

O *Checksum* mais simples consiste na soma de todos os *bytes* de uma transmissão. Esse valor é acrescentado como o último *byte* da transmissão. Após o recebimento dos n *byte*s, o receptor então adiciona os primeiros n-1 *bytes* recebidos, para então comparar com o conteúdo do ultimo *byte*. Ambos devem ser iguais, caso contrário um erro ocorreu.

Assim como o código de paridade, existem variações do *Checksum* que são detalhadas em Koren e Krishna (2007). Todos estes esquemas de *Checksum* permitem a detecção, mas não a localização do erro, e no caso de um erro ser detectado, todo o bloco de dados deve ser retransmitido.

Para tratar com o problema das falhas transientes e os possíveis erros gerados por estas, em Huang e Gouda (2005) é proposto o conceito de um estado *Checksum* (*a state Checksum*) que é uma redundância que pode ser adicionada ao estado de um sistema de

auto-estabilização (*a self-stabilizing system*), para que algumas classes de falhas se tornem visíveis para o sistema, e o sistema possa limitar a propagação de seus efeitos nocivos.

Fechner (2006) apresenta a análise de um esquema para a detecção de falhas transientes baseado em *Checksum* para processadores *pipeline* e em Fechner (2007) também foi proposto um esquema baseado em *Checksum* para detecção de falhas em processadores *pipeline*, porém este otimiza os sistemas anteriores em termos de latência e cobertura de detecção de falhas por não utilizar compressão, mas os *Checksums* completos de vários estágios do *pipeline*.

2.4.4.3 Códigos M de N.

O código M de N é um exemplo de um código de detecção de erro unidirecional, ou seja, um erro onde todos os bits afetados mudam na mesma direção, ou de 0 para 1 ou de 1 para 0 mas não em ambas as direções.

Em um código M de N, cada uma das palavras de código de tamanho N tem exatamente M bits iguais a 1, resultando em $n!/(n-m)!m!$ palavras de código. Qualquer erro simples irá alterar a quantidade de bits iguais a 1 para uma quantidade de M+1 ou M-1 bits iguais a 1 e então será detectado. Erros múltiplos unidirecionais também serão detectados.

Um subconjunto do Código M de N é o Código 1 de N, onde as palavras do código apresentam apenas um bit igual a 1. Em (LALA; WALKER, 2001) (TOWNSEND; THORNTON; LALA, 2002) e (TOWNSEND; ABRAHAM; LALA, 2003) foi utilizado, para garantir a tolerância a falhas, o Código 1 de 3, no qual o comprimento das palavras do código é 3 sendo que elas apresentam apenas um bit igual a 1.

A principal vantagem do código M de N é sua simplicidade conceitual. No entanto, a codificação e decodificação são operações relativamente complexas, pois tais códigos não são, em geral, separáveis, ao contrário dos Códigos de Paridade e *Checksum*.

No entanto, códigos M de N separáveis podem ser construídos, mas apresentam um overhead maior (100 % ou mais) que os M de N não separáveis (KOREN; KRISHNA, 2007).

2.4.4.4 Código Berger

Um código Berger de tamanho *n* possui *d* bits de dados e *c* bits de verificação onde $c = \lceil \log_2(d+1) \rceil$ e $n = d + c$. O Código Berger é o código menos redundante e pode ser utilizado para detectar erros simples e múltiplos (LALA, 2001). As palavras pertencentes ao Código Berger são construídas através da formação de um número binário que corresponde à quantidade de bits iguais a 1 nos bits dos dados, e anexando o complemento bit a bit deste número, aos bits de dados, como bits de verificação (LALA, 2001).

Por exemplo, se *d* = 0101000, logo $c = \lceil \log_2(7+1) \rceil = 2$ e portanto o Código Berger deve ter um tamanho de 10 (=7+3). Então os *c* bits de verificação são derivados como segue:

Quantidade de bits iguais a 1 nos bits de dados: *d* = 2, sendo que o número binário equivalente é 010. Assim, o complemento bit a bit de 010 é 101 que serão os bits de verificação anexados ao dado. Portanto, *n* = 0101000101(LALA, 2001).

Os bits de verificação *c*, podem ser obtidos através do número binário que representa a quantidade de 0s nos *d* bits da informação. Lala (2001) explica os dois diferentes modos que geram os bits de verificação, quando estes representam a quantidade de 0s contidos no dado.

Lo, Thanawastien e Rao (1990) utilizaram o código Berger para detecção de erros concorrentes e Ossi (2009) utiliza o código Berger e repetições de instruções para mitigação de *soft-error* a nível arquitetural.

2.4.4.5 Código Cíclico

No código cíclico, a codificação do dado consiste em multiplicar a palavra de dado por um número constante, e a palavra de código será o produto que resultar. A decodificação é feita dividindo-se a palavra de código pelo menos número constante: se o restante não for igual a 0, isto indica que um erro ocorreu (KOREN; KRISHNA, 2007). Estes códigos são chamados de cíclicos porque para toda palavra de código $a_{n-1},a_{n-2},...,a_0$, sua mudança cíclica $a_0,a_{n-1},a_{n-2}...,a_1$ também é uma palavra de código. Códigos cíclicos são muito utilizados em armazenamento de dados e comunicação.

Supondo que *d* é o número de bits do dado que será codificado. A palavra codificada de comprimento *n* é obtida multiplicando os d bits do dado por um número de comprimento n-d+1 .

Em Peterson (1961) códigos cíclicos são definidos e descritos utilizando polinômios. Além disto, Peterson (1961) descreve em detalhes os equipamentos necessários para implementar sistemas de detecção de erros utilizando Códigos Cíclicos.Explicações detalhadas sobre o código cíclico, bem como os processos de codificação e decodificação para este código também podem ser vistas em Koren e Krishna (2007) e Lala (2001).

2.4.4.6 Código Aritmético

Códigos Aritméticos são aqueles que são preservados durante as operações aritméticas (KOREN; KRISHNA, 2007). Esta propriedade permite detectar erros que podem ocorrer durante a execução de uma operação aritmética. Tal detecção concorrente de erros pode ser alcançada através da duplicação da unidade aritmética, o que implica em um elevado custo de *Hardware* (KOREN; KRISHNA, 2007).

O código é preservado durante uma operação aritmética *a* se para quaisquer dois operandos *X* e *Y*, e as correspondentes entidades codificadas *X'* e *Y'*, existe uma operação *b*, que quando aplicada aos operandos codificados *X'* e *Y'*, produzirá o mesmo resultado que o resultado produzido com a aplicação da operação original *a* aos operandos originais *X* e *Y*. Conseqüentemente, o resultado da operação aritmética será codificado com o mesmo código que os operandos.

Os Códigos Aritméticos podem ser separáveis ou não separáveis. *Residue Codes* é o tipo mais simples de código aritmético de detecção de erro separável. Maiores detalhes sobre *Residue Codes* podem ser encontrados em (PARHAMI, 2000) (LALA, 2001) (KOREN; KRISHNA, 2007).

Maiores detalhes sobre código aritmético podem ser encontrados em Koren e Krishna (2007). Parhami (2000) também explica detalhadamente Códigos Aritméticos para Detecção de Erros e Códigos Aritméticos para Correção de Erros.

Os códigos detalhados acima apenas detectam os erros. Os códigos que possibilitam a correção de erros são os Códigos *Hamming, Reed-Solomon e Hsiao.* Em Lisboa (2009) é proposta uma técnica que adota o Código *Hamming* como um meio para proteger a lógica combinacional. Maiores detalhes sobre estes códigos de detecção e correção de erros também podem ser encontrados em Lala (2001).

3 SOMADORES TOLERANTES A FALHAS

Este capítulo apresenta arquiteturas de somadores não-protegidos e protegidos contra falhas transientes analisadas neste trabalho. As estruturas dos somadores são apresentadas, já prevendo-se sua implementação em tecnologia CMOS, de modo a viabilizar a obtenção das características de custo (i.e., número de transistores), atraso crítico e consumo médio de potência em bases realistas, conforme será detalhado no próximo capítulo.

Dois tipos básicos de somadores foram escolhidos: RCA (*Ripple-Carry Adder*) e BSDA (*Binary Signed Digit Adder*). O primeiro é o somador mais frequentemente utilizado em projetos onde o desempenho não é fator preponderante. No contexto deste trabalho, o RCA representa a classe de somadores que operam com sistema numérico binário convencional. O BSDA é um somador que opera com sistema numérico de dígito binário com sinal - *Binary Signed Digit* (BSD), o qual é um sistema numérico não convencional do tipo redundante.

Três arquiteturas de somadores baseadas no RCA são consideradas:

- RCA - *Ripple-Carry Adder,*
- RCATMR - *Ripple-Carry Adder* TMR e
- RCAIOI - *Ripple-Carry Adder* com Entradas e Saídas Invertidas,

Destas, o RCA corresponde à versão não-protegida, o RCATMR utiliza a técnica de proteção TMR (*Triple-Module Redundancy*) e o RCAIOI utiliza a técnica de proteção RESI (Recomputação com Entradas e Saídas Invertidas), a qual é uma técnica baseada em redundância de informação, mas também utiliza redundância de tempo e redundância de *hardware*. Tanto o RCA quanto o RCATMR podem servir de referência, nas comparações com as demais arquiteturas de somadores protegidos.

Um total de seis arquiteturas baseadas no BSDA são analisadas. São elas:

- BSDA_BSD - *Binary Signed Digit Adder* com operandos em BSD NAF,
- BSDA_C2 - *Binary Signed Digit Adder* com operandos em complemento de dois,
- BSDA1-3_BSD - *Binary Signed Digit Adder* com Codificação 1 de 3 e operandos em BSD NAF,
- BSDA1-3_C2 - *Binary Signed Digit Adder* com Codificação 1 de 3 e operandos em complemento de dois,
- BSDAPar_BSD - *Binary Signed Digit Adder* com Verificação de Paridade e operandos em BSD NAF e

- BSDAPar_C2 - *Binary Signed Digit Adder* com Verificação de Paridade e operandos em complemento de dois.

Destas, o BSDA_BSD e o BSDA_C2 são versões de BSDA não-protegidas, ao passo que as demais são versões de BSDA protegidas por meio de redundância de informação.

A escolha dos somadores baseados no BSDA, bem como do RCAIOI, deveu-se ao objetivo principal deste trabalho, que é avaliar somadores protegidos contra falhas que usam a técnica de redundância de informação.

3.1 *Ripple-Carry Adder* (RCA)

O *Ripple-Carry Adder* (RCA) é um somador de baixo custo, cuja estrutura baseia-se na fatoração sucessiva da expressão lógica do sinal de *carry*. Em função de seu baixo custo e também por ser uma opção arquitetural bastante utilizada, frequentemente, o RCA é utilizado como referência na avaliação de outros tipos de somadores. Para a implementação de um RCA de *n* bits, *n* somadores completos (SCs) como o mostrado na Figura 3.1 são utilizados, conectando-se o *carry-out* do SC de ordem "*i*" ao *carry-in* do SC de ordem "*i+1*". O *carry-in* do SC de ordem zero corresponde a uma entrada primária do RCA, o que torna o somador mais genérico, facilitando a implementação de outras operações aritméticas, como por exemplo, a subtração e o incremento.

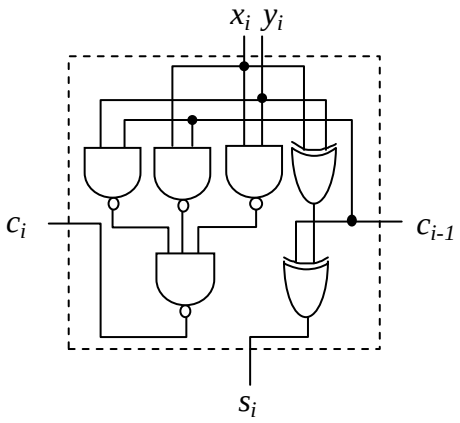


Figura 3.1: Circuito Lógico do Somador Completo (SC)

3.2 *Ripple-Carry Adder* TMR (RCATMR)

A Figura 3.2 apresenta o diagrama de blocos de um RCA de *n* bits projetado com a técnica *Triple Modular Redundancy* (TMR) (NEUMANN, 1956), apresentada na seção 2.4. A Figura 3.3 apresenta o circuito lógico do votador de 1 bit, o qual é constituído por portas NAND.

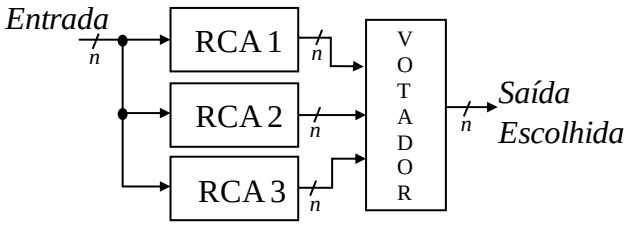


Figura 3.2: RCA protegido com TMR.

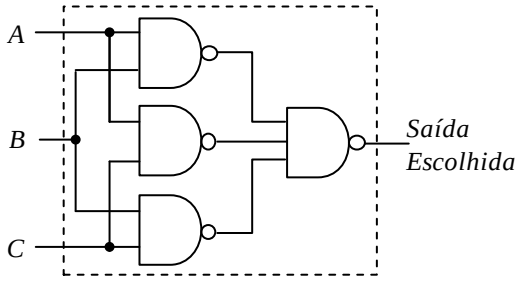


Figura 3.3: Circuito Votador

3.3 *Ripple-Carry Adder* com Entradas e Saídas Invertidas (RCAIOI)

Oikonomakos e Fox (2006) propuseram uma nova técnica para correção de erros em circuitos aritméticos. Tal técnica é chamada no presente trabalho de RESI (Recomputação com Entradas e Saídas Invertidas). A técnica RESI é baseada em recomputação com inversão de entradas e saídas combinando as idéias de *Rollback and Recomputation* de Orailoglu e Karri (1996) com as soluções de auto-verificação aritméticas encontradas em Nicolaidis et al. (1997). Além disso, a técnica RESI baseia-se no fato de que as funções lógicas das saídas de um SC (s_i e c_{out}) são funções auto-duais. Uma função lógica é auto-dual quando $\overline{f}(\overline{v}) = f(v)$, onde v é um vetor de variáveis lógicas.

A Tabela 3.1 mostra a tabela-verdade de um SC, onde se pode observar que ambas as funções de saída, s_i e c_{out}, são auto-duais, isto é, invertendo-se logicamente todas as entradas do SC, as saídas irão produzir valores invertidos. Caso ocorra uma falha no SC, se esta se propagar até uma de suas saídas e for detectada, a computação poderá ser refeita, porém com os operandos de entrada invertidos. Logo, os resultados desejados serão obtidos com a inversão das saídas produzidas.

Tabela 3.1: Tabela-Verdade de um Somador Completo

x_i	y_i	c_{in}	s_i	c_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Portanto, esta técnica pode ser aplicada em circuitos aritméticos para a correção de falhas simples, desde que as falhas nas entradas e saídas primárias nos SCs possam ser detectadas e que o princípio da correção por inversão das entradas e saídas possa ser aplicado também para as falhas que ocorram internamente ao SC, e não apenas nas entradas. Para satisfazer à primeira exigência, Oikonomakos e Fox (2006) propuseram um SC com *carry* redundante (SCCR), conforme mostrado na Figura 3.4.

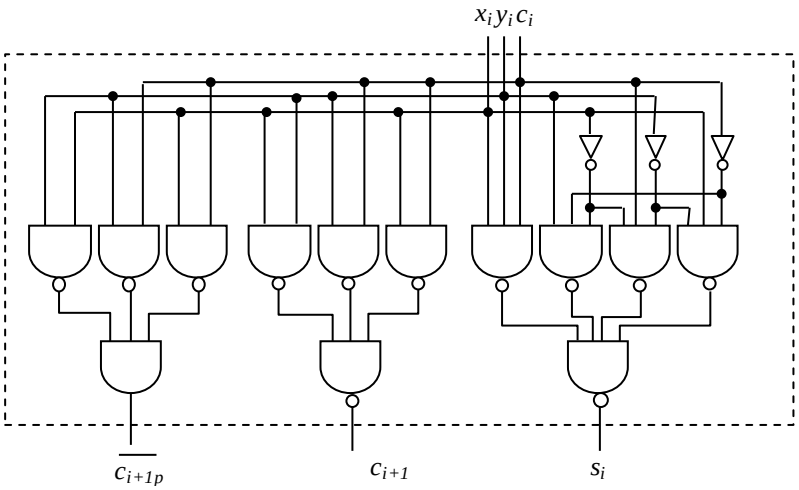


Figura 3.4: Somador Completo com *Carry* Redundante (SCCR)

A predição de paridade é realizada por portas CMOS XOR, conectadas entre todos os *carries* redundantes (cp_i), conforme ilustrado na Figura 3.5.

Este somador requer um bloco controlador (não implementado neste trabalho), o qual também contribui para o acréscimo de transistores. Por outro lado, vale observar que não há compartilhamento de lógica na geração de *carry*, característica muito apropriada no contexto de tolerância a falhas. Caso ocorra uma falha em uma das portas do somador, apenas uma saída será corrompida.

Inicialmente, o controlador faz $I = 0$, fazendo com que o somador seja alimentado com os operandos não invertidos. As saídas geradas, juntamente com o bit de paridade previsto, são armazenadas em um registrador de $n+1$ bits que posteriormente alimenta um verificador de paridade, junto com o *carry* de saída do somador e o sinal de controle *I*.

Caso ocorra uma falha em um dos SCCRs, em um multiplexador ou no registrador, ela será detectada pelo verificador de paridade e o controlador não avança para o próximo estado. Ao invés, ele fará $I = 1$ para forçar a repetição da computação, porém com todas as entradas dos somadores completos invertidas (inclusive o *carry* de entrada que está conectado diretamente em *I*). Desta maneira, o complemento dos *carries* de saída dos SCCRs será produzido livre de falhas, bem como o complemento das saídas das somas que serão armazenadas no registrador. Assumindo que *n* é par, os bits de paridade dos operandos de entrada A e B permanecem inalterados com a inversão, não prejudicando assim a tolerância a falhas.

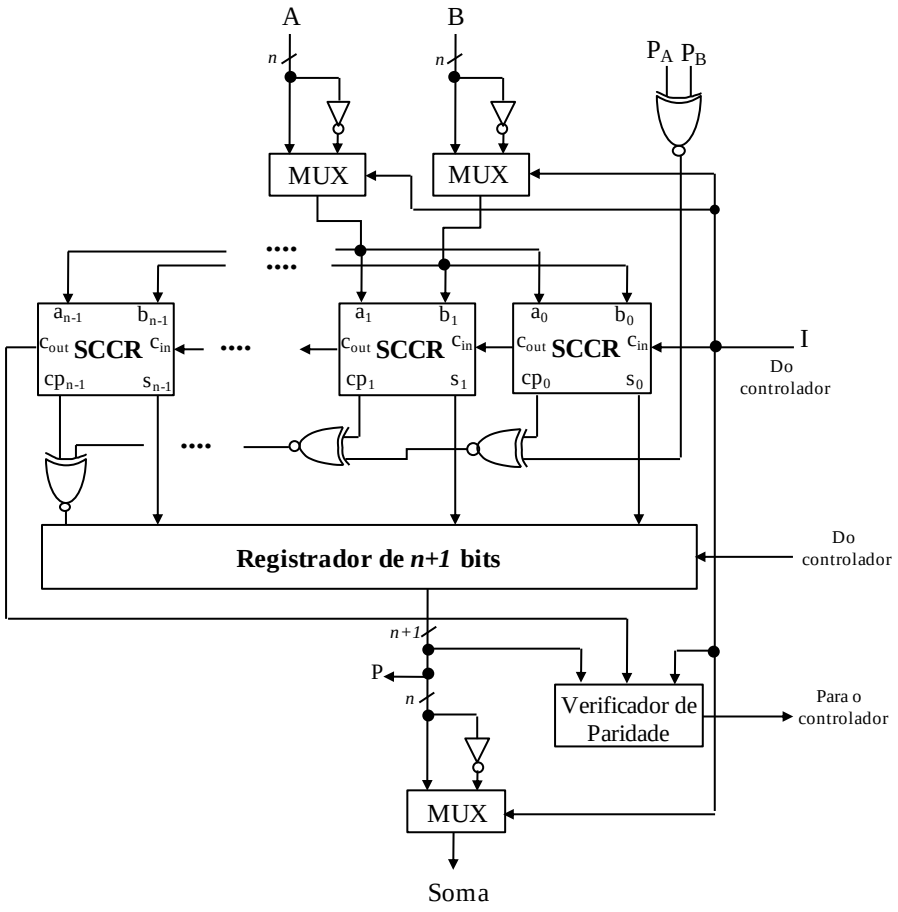


Figura 3.5: RCAIOI de *n* bits

Além de falhas ocorridas nos SCCRs, esta técnica também garante proteção contra falhas ocorridas nos multiplexadores e no registrador. A Figura 3.6 apresenta o multiplexador utilizado.

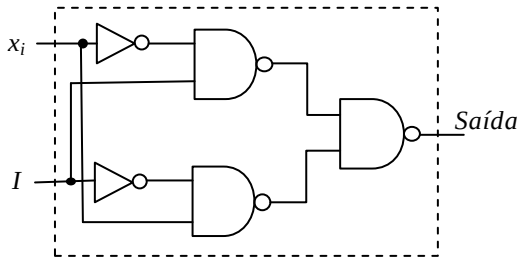


Figura 3.6: Multiplexador utilizado na técnica RESI.

Caso ocorra uma falha na entrada A ou no inversor conectado a esta entrada, ela não afetará a computação normal com $I = 0$, não prejudicando assim o resultado. Já no caso em que a falha ocorra no sinal de controle I ou no inversor conectado ao mesmo e corrompa a computação direta, ela será mascarada na computação inversa, pois o sinal I será invertido. Caso ocorra uma falha em uma das portas NANDs conectadas às entradas, ou ela não se manifestará na computação direta, ou então será mascarada na computação inversa, visto que cada uma destas portas será alimentada pelo menos por um 0 lógico em cada computação.

Para a construção do registrador, pode ser utilizado um *flip-flop* D mestre-escravo. No entanto, falhas podem ocorrer nestes *flip-flops.* Então, para uma melhor garantia de tolerância a falhas, foi adotado o *flip-flop* apresentado na Figura 3.7.

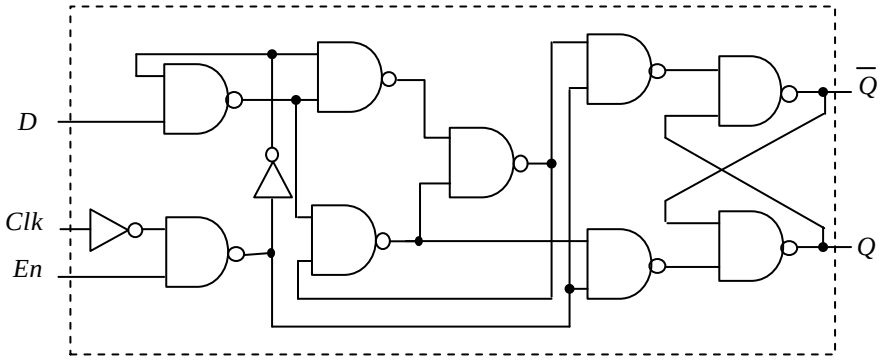


Figura 3.7: *Flip -Flop* utilizado na técnica RESI.

Este *flip-flop* também trabalha como um mestre-escravo, armazenando $\overline{D}$ na borda negativa do *clock* e registrando a saída adequada na próxima borda positiva do *clock*. Qualquer falha que ocorra na arquitetura apresentada na Figura 3.5 será detectada pela verificação da paridade e será corrigida pela inversão dos operandos de entrada e posterior recomputação. Portanto, o registrador irá armazenar o resultado desejado ou o complemento do mesmo.

Esta técnica utiliza redundância de tempo, pois caso uma falha seja detectada, a computação será executada duas vezes. Além disso, utiliza também redundância de *hardware* para a geração dos *carries* redundantes e para a geração e verificação da paridade.

Para a avaliação desta técnica de proteção, será assumido que o controlador, os sinais de controle e os operandos de entrada A e B são livres de falhas.

3.4 *Binary Signed Digit Adder* com Operandos representados em BSD NAF (BSDA_BSD)

O *Binary Signed Digit Adder* (BSDA) utiliza o sistema de representação de dígito com sinal de base 2 (*Binary Signed Digit* – BSD). Tal sistema numérico é considerado redundante, pois apresenta o conjunto de dígitos $\{\bar{1},0,1\}$, o que permite que um inteiro possa ter mais de uma representação.

Como os sistemas digitais utilizam o sistema de representação binário, para este somador foram desenvolvidos dois conversores, um de entrada, que converte as entradas representadas em binário para a sua correspondente representação BSD NAF, e um conversor de saída, que realiza o processo inverso, isto é, converte o resultado da soma representada em BSD para sua correspondente representação binária. A representação BSD NAF foi escolhida por apresentar a propriedade de não possuir dois dígitos adjacentes iguais a 1, isto é, em uma representação BSD NAF, todos os dígitos iguais a 1 são separados por, no mínimo, um zero. Tal propriedade garante uma redução na atividade de chaveamento dos sinais do circuito, reduzindo assim o consumo de potência.

A seguir, são apresentados detalhes destes conversores. Também é apresentada a soma BSD propriamente dita, que é dita "livre de *carry*" (*carry-free*).

3.4.1 Conversor de Representação Binária para BSD NAF

O projeto do conversor de representação binária para BSD NAF foi baseado no método de Reitwiener (HWANG, 1979) (JOYE; YEN, 2000) detalhado no capítulo anterior. O método Reitwiener considera apenas representação binária sem sinal. Portanto, foi projetado um conversor que trata de forma diferenciada o bit mais significativo da representação binária.

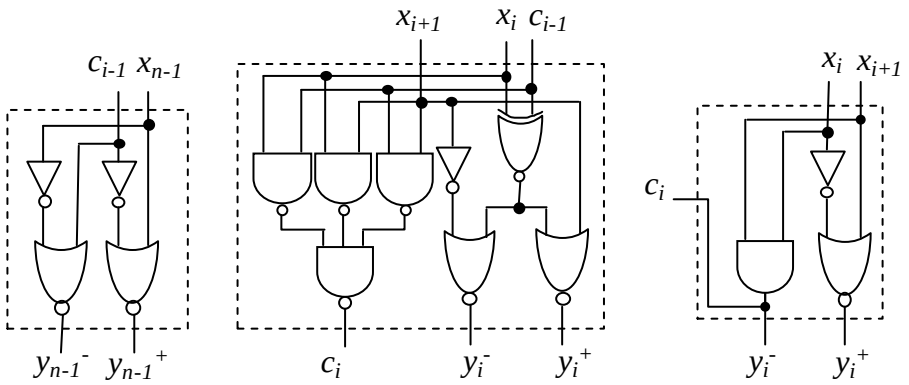
Considerando por exemplo a representação binária em complemento de dois $(1110)_2 = (-2)_{10}$, caso seja aplicado o método Reitwiener original, a representação resultante seria $(100\bar{1}0)_{BSD} = (14)_{10}$. Portanto, além de tratar de forma diferenciada o cálculo do bit mais significativo, o conversor utilizado realiza *n-1* interações e não *n*, como no método Reitwiener original. Desta forma, o resultado obtido será $(00\bar{1}0)_{BSD} = (-2)_{10} = (1110)_2$, conforme o esperado.

Para a representação de cada dígito BSD, serão necessários dois bits. A Tabela 3.2 mostra a codificação escolhida para representar cada dígito BSD, que foi escolhida porque permite a obtenção do sinal do dígito através da análise do bit mais significativo, isto é, o bit x_i^- representa o sinal do dígito BSD.

Tabela 3.2: Codificação do dígitos BSD

x_i	x_i^-	x_i^+
0	0	0
1	0	1
$\bar{1}$	1	0

A Figura 3.8 apresenta os circuitos lógicos para os conversores de entrada para os bits mais significativo (a), menos significativo (c) e demais bits (b).



(a) (b) (c)

Figura 3.8: Circuitos Lógicos do conversor de Representação Binária para representação BSD.

Os circuitos lógicos para os conversores do bit menos significativo e dos bits intermediários foram implementados a partir da tabela-verdade original do método Reitwiener, sendo que o circuito conversor do bit menos significativo foi projetado de forma otimizada devido ao *carry* de entrada ser sempre 0. O circuito lógico para a conversão do bit mais significativo também foi projetado de forma otimizada, visto que a entrada x_n será sempre 0. No entanto, para a implementação deste conversor a tabela-

verdade do método Reitwiener foi alterada para que o método gerasse uma representação em complemento de 2, conforme explicado acima. A Tabela 3.3 mostrada na Tabela 3.3. A Figura 3.9 mostra o diagrama de blocos de um conversor para um operando de entrada de 4 bits.

Tabela 3.3: Tabela-Verdade alterada para o Método Reitwiener

c_{i-1}	x_i	x_{i+1}	y_i
0	0	0	0
0	1	0	$\bar{1}$
1	0	0	1
1	1	0	0

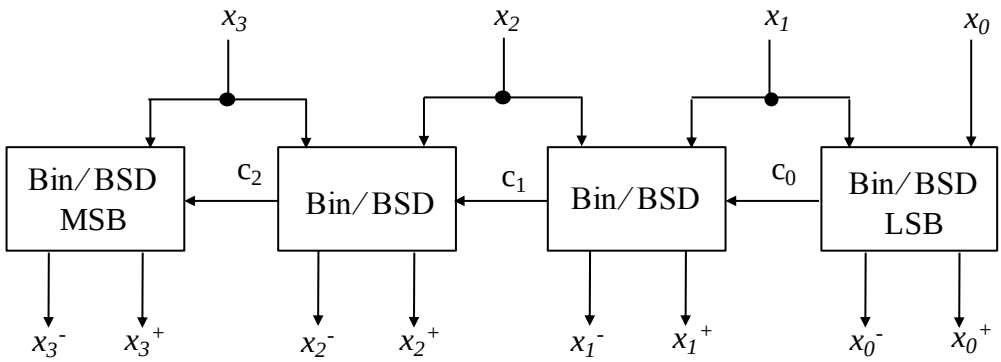


Figura 3.9: Conversor de Binário para BSD de 4 bits.

3.4.2 Soma BSD

O sistema de representação BSD permite a implementação de somadores livres da propagação de *carry*, conhecidos como *carry-free adders*. Isto é possível devido ao fato da adição ser realizada em duas etapas.

Na primeira etapa, são adicionados os dois dígitos da mesma posição de cada operando. O resultado desta adição será a geração de uma soma e um *carry* intermediários, $w_i \in \{\bar{1},0,1\}$ e $c_i \in \{\bar{1},0,1\}$, respectivamente. Isto é feito utilizando as regras para primeira etapa da adição *carry-free*, mostradas na Tabela 3.4 .

Tabela 3.4: Regras para a primeira etapa da adição carry-free.

x_i	y_i	x_{i-1} y_{i-1}	c_i	w_i
1	1	____	1	0
1	0	Ambos são positivos	1	$\bar{1}$
1	0	Pelo menos um negativo	0	1
1	$\bar{1}$	____	0	0
0	0	____	0	0
0	$\bar{1}$	Ambos são positivos	0	$\bar{1}$
0	$\bar{1}$	Pelo menos um negativo	$\bar{1}$	1
$\bar{1}$	$\bar{1}$	____	$\bar{1}$	0

Portanto, a partir das regras para a primeira etapa da adição *carry-free*, a soma e o *carry* intermediários são determinados de forma que w_i e c_{i-1} não sejam ambos iguais a 1 e tampouco a $\bar{1}$, evitando assim a geração de *carry* na geração da soma final.

Na segunda etapa, é determinada a soma final, $s_i \in \{\bar{1},0,1\}$, através da adição da soma intermediária, w_i e o *carry* intermediário da posição menos significativa imediatamente anterior, c_{i-1}, ou seja, a soma final é determinada por $s_i = w_i + c_{i-1}$. A Tabela 3.5 apresenta a tabela-verdade para a geração da soma final.

Tabela 3.5: Tabela-verdade para a geração da soma final

w_i	c_{i-1}	s_i
0	0	0
0	1	1
0	$\bar{1}$	$\bar{1}$
1	0	1
1	1	—
1	$\bar{1}$	0
$\bar{1}$	0	$\bar{1}$
$\bar{1}$	1	0
$\bar{1}$	$\bar{1}$	—

Logo, a partir das duas etapas necessárias para a realização da soma BSD, a propagação do *carry* é limitada a somente um dígito, e portanto, a adição é feita em paralelo, como ilustra o diagrama de blocos da Figura 3.10. Assim, esses somadores são chamados de somadores *carry-free*, visto que é eliminada a necessidade de propagação do *carry*.

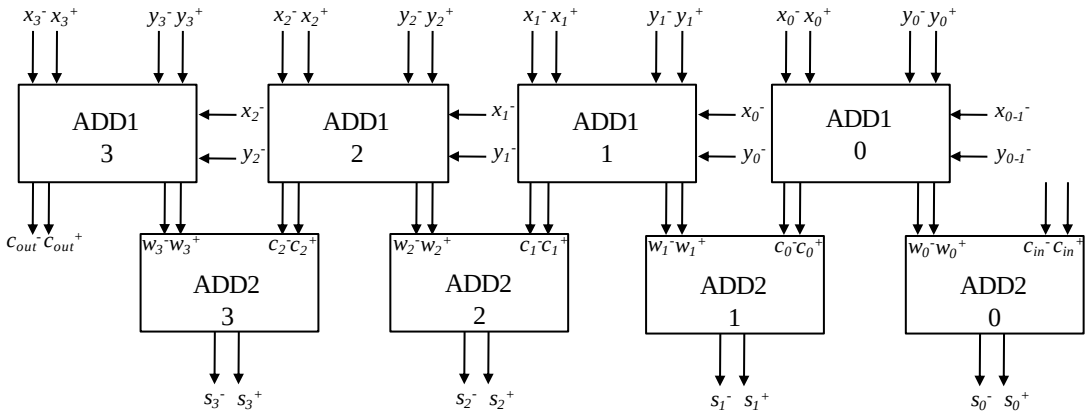


Figura 3.10: BSDA de *n* bits.

Como pode ser visto na Figura 3.10, para a implementação do BSDA foram projetados dois blocos básicos: o ADD1, que gera a soma e o *carry* intermediários, e o ADD2, que gera a soma final.

3.4.2.1 ADD1 – Gerador da Soma/Carry Intermediários

Este bloco foi implementado a partir da Tabela 3.4 e utiliza portas CMOS XNORs, NANDs, NOTs e NORs (TAKAHASHI; KONTA, 2003). Como se pode observar na Tabela 3.4, para a geração da soma e *carry* intermediários, é necessário avaliar o sinal dos operandos x_{i-1} e y_{i-1}. Desta forma, com a codificação escolhida para a representação dos dígitos BSD apresentada na Tabela 3.2, somente os bits mais significativos dos operandos x_{i-1} e y_{i-1} serão necessários e portanto, o bloco ADD1 pode ser implementado com apenas 6 entradas (CARDARILLI et al., 2003), x_i^-, x_i^+, y_i^-, y_i^+, x_{i-1}^- e y_{i-1}^- , e 2 saídas, w_i e c_i.

A Figura 3.11 apresenta o circuito lógico do bloco ADD1.

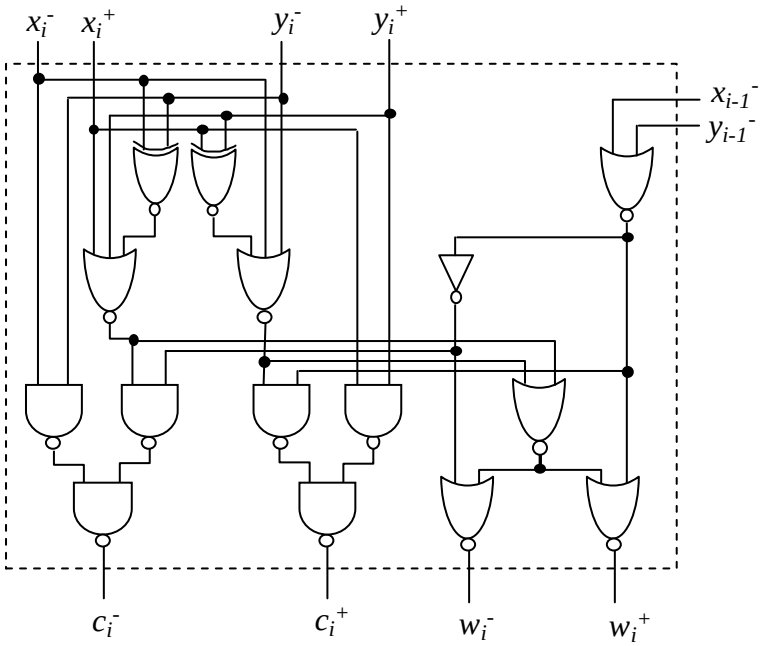


Figura 3.11: Gerador da Soma/*Carry* Intermediários – ADD1.

3.4.2.2 ADD2 – Gerador da Soma Final

Este bloco tem como entrada as saídas dos blocos ADD1, w_i e c_{i-1} e como saída, a soma final, s_i. Ele foi implementado a partir da Tabela 3.5 e foi projetado apenas com portas NANDs e NOTs. A Figura 3.12 apresenta o circuito lógico do bloco ADD2.

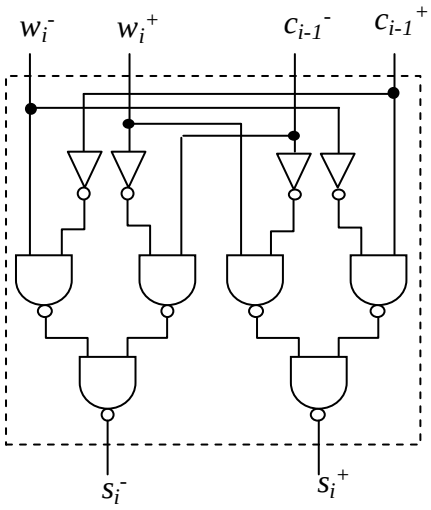


Figura 3.12: Gerador da Soma Final – ADD2

3.4.3 Conversor BSD para Representação Binária

Os blocos ADD1 e ADD2 apresentados anteriormente geram a soma, porém em representação BSD. Portanto, após a execução do cálculo, o resultado deve ser convertido para a representação binária. Para a realização desta conversão, foi projetado um conversor de BSD para binário.

A conversão de BSD para binário consiste em dividir a representação BSD em duas partes, uma positiva e outra negativa, e somar a primeira com o complemento da segunda, como mostra o exemplo da Figura 3.13.

	(0	**-1**	**0**	**1)$_{BSD}$**	**= (-3)$_{10}$**
P:	0	0	0	1	
N:	0	1	0	0	
	1	0	1	1	
+				1	
	1	1	0	0	
+	0	0	0	1	
	(1	**1**	**0**	**1)$_2$**	**= (-3)$_{10}$**

Figura 3.13: Exemplo de conversão da Representação BSD para a Representação Binária.

Devido à codificação binária dos dígitos BSD apresentada na Tabela 3.2, esta conversão pode ser implementada utilizando SCs juntamente com a inversão das entradas do bit x_i^-, que representa o sinal do dígito BSD. A Figura 3.14 apresenta o diagrama de blocos do conversor de BSD para binário utilizado para a conversão do resultado de uma soma de dois operandos de 4 bits.

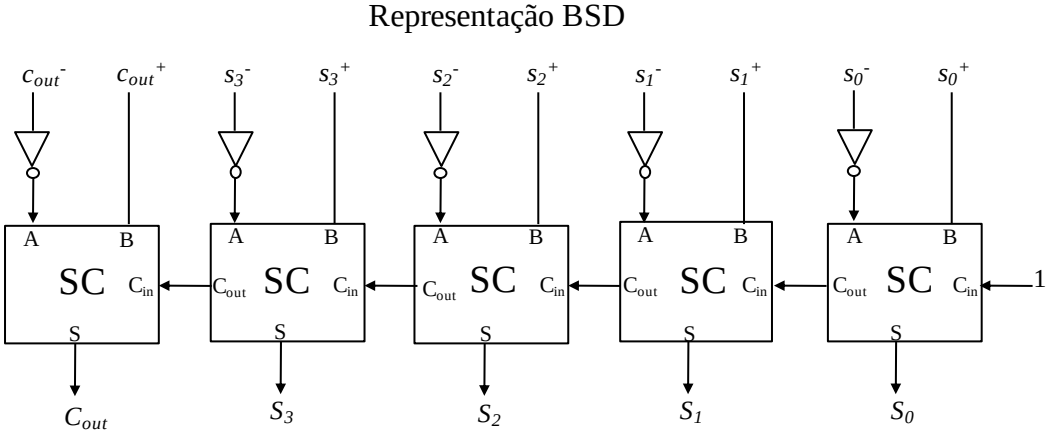


Figura 3.14: Conversor BSD para Binário.

As conversões necessárias para a realização da soma dos operandos representados em BSD NAF se tornam a principal desvantagem desta implementação. O ganho em desempenho obtido na etapa de soma propriamente dita é relevante no caso de sistemas que realizem todas operações com dados representados em BSD NAF. Caso contrário, o ganho de desempenho da adição provavelmente será perdido devido ao tempo necessário para as conversões de dados de entrada e de saída.

3.5 *Binary Signed Digit Adder* – Operandos representados em Complemento de dois (BSDA_C2)

Como o BSDA não é um somador tradicional, foi implementada uma segunda versão do mesmo, com operandos representados em complemento de dois, objetivando comparar com a versão deste com operandos representados em BSD NAF.

Conforme mencionado na seção anterior, a principal desvantagem de uma soma BSD são os estágios de conversão entre os tipos de representação. Logo, para amenizar esta desvantagem, esta versão de BSDA não utiliza operandos representados em BSD NAF, e sim em complemento de dois.

3.5.1 Conversor de Representação Binária para BSD

Visto que a representação binária é um tipo especial de representação BSD (HWANG, 1979) (EBEID; HASAN, 2007), não é necessária a conversão de representação binária para BSD. Nesta versão, os dígitos dos operandos são apenas codificados de acordo com a Tabela 3.2. A Figura 3.15 mostra que os operandos de entrada podem ser conectados diretamente às entradas do somador. Os sinais foram conectados ao terra apenas para ilustração. A Figura 3.15a apresenta como é tratado o bit mais significativo, ao passo que a Figura 3.15b mostra como são tratados os demais bits.

Figura 3.15: Operandos de entrada do BSDA em complemento de dois.

3.5.2 Somador BSD – Entradas em Complemento de Dois.

Nesta versão, a soma BSD também é executada em duas etapas, sendo primeiramente gerados a soma e o *carry* intermediários e após, a soma final. Porém, devido ao fato de considerar as entradas representadas em complemento de dois, os circuitos responsáveis por executar a soma BSD podem ser simplificados, visto que o conjunto de possibilidades de entrada foi reduzido. A Figura 3.16 apresenta o diagrama de blocos de um BSDA para operandos de entrada de 4 bits representados em complemento de dois.

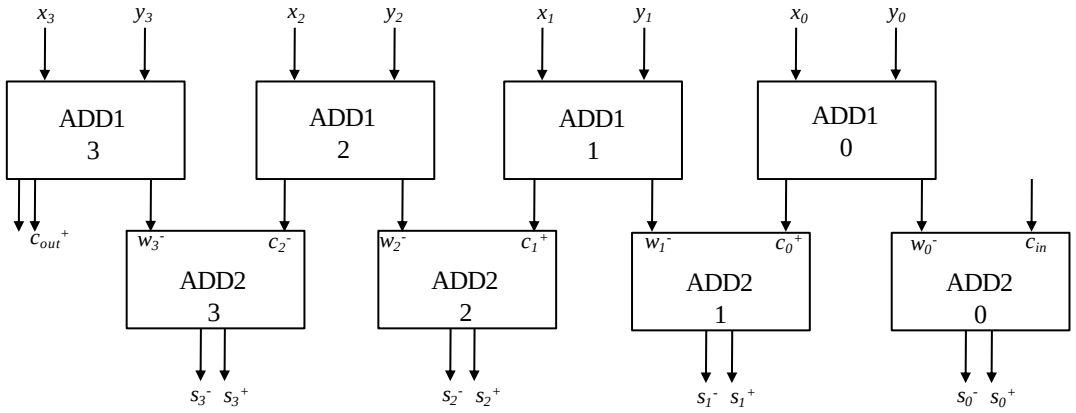


Figura 3.16: BSDA com entrada em complemento de dois

3.5.2.1 ADD1 – Gerador da Soma/Carry Intermediários

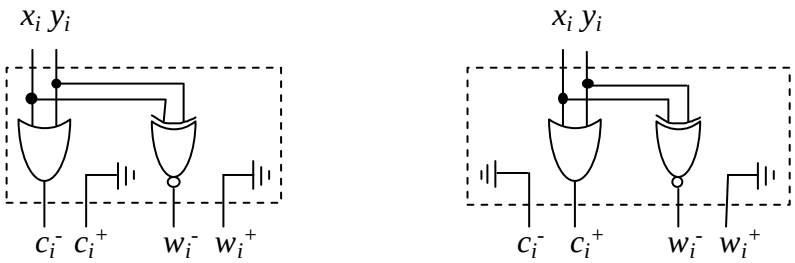
Com os operandos representados em complemento de dois, o conjunto de possibilidades de entradas é reduzido. Portanto, as regras da adição *carry-free* apresentadas nas Tabela 3.4 e Tabela 3.5 podem ser simplificadas, e consequentemente, os circuitos que executam a soma também são simplificados. A Tabela 3.6 apresenta as regras da primeira etapa da adição.

Nesta versão, o bloco ADD1 pode ser implementado apenas por uma porta OR e uma XOR, pois não é necessário avaliar os operandos menos significativos, x_{i-1} e y_{i-1} e também a possibilidade de um dígito $\bar{1}$ existirá apenas para o bit mais significativo dos operandos (FRANCO, 2008). Além disto, como pode ser visto na Tabela 3.6, ao codificar os dígitos BSD de acordo com a Tabela 3.2, os sinais c_i^- e $wi+$ gerados pelos *n-1* bits menos significativos serão sempre iguais a 0, assim como os sinais $ci+$ e $wi+$ gerados pelo bit mais significativo.

Tabela 3.6: Regras para a primeira etapa da adição *carry-free* para operandos representados em complemento de dois.

Bit Mais Significativo				**Bits Menos Significativos**			
x_{n-1}	y_{n-1}	$c_{n-1}^- c_{n-1}^+$	$w_{n-1}^- w_{n-1}^+$	x_i	y_i	$c_i^- c_i^+$	$w_i^- w_i^+$
0	0	00	00	0	0	00	00
0	1	00	10	0	1	01	10
1	0	00	10	1	0	01	10
1	1	10	00	1	1	01	00

A Figura 3.17a apresenta o circuito lógico do bloco ADD1 para o bit mais significativo do operando, ao passo que a Figura 3.17b mostra o circuito lógico para os demais bits do operando.



(a) (b)

Figura 3.17: Gerador da Soma/*Carry* Intermediários – ADD1.

3.5.2.2 ADD2 – Gerador da Soma Final

Como na versão anterior, este bloco também tem como entrada as saídas dos blocos ADD1, w_i^- e c_{i-1}^+ e gera a soma final, s_i, representado em BSD. A Tabela 3.7 apresenta a tabela-verdade simplificada para a geração da soma final, de acordo com as saídas geradas pelos blocos ADD1.

Tabela 3.7: Tabela-verdade para a geração da soma final para operandos representados em complemento de dois.

$w_i^- w_i^+$	$c_{i-1}^- c_{i-1}^+$	$s_i^- s_i^+$
00	00	00
00	01	01
10	00	10
10	01	00

A partir da Tabela 3.7 foi projetado um circuito lógico para o bloco ADD2 usando somente portas NOR e NOT, conforme ilustra a Figura 3.18.

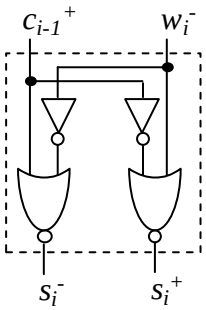


Figura 3.18: Gerador da Soma Final – ADD2

3.5.3 Conversor BSD para Representação Binária

Apesar da versão do BSDA com entradas em complemento de dois não necessitar do estágio de conversão de binário para BSD, ela ainda necessitará do estágio de conversão do resultado gerado em BSD para a sua representação em complemento de dois. Portanto, o conversor apresentado na Figura 3.14 também é utilizado nesta versão, sendo esta a principal desvantagem deste BSDA.

3.6 *Binary Signed Digit Adder* com Codificação 1 de 3 – Operandos em Representação BSD (BSDA1-3_BSD)

O BSDA foi protegido utilizando redundância de informação, obtida através de codificações numéricas. No BSDA foi aplicada a codificação 1 de 3, um subconjunto da codificação N de M, onde as palavras do código possuem o tamanho de 3 bits, sendo apenas um deles igual a 1

3.6.1 Conversor 1 de 3 de Representação Binária para BSD NAF

O conversor 1 de 3 de representação binária para BSD NAF também foi baseado no método de Reitwiener (HWANG, 1979) (JOYE; YEN, 2000), com as alterações mencionadas anteriormente. Porém, para a aplicação da codificação 1 de 3, os dígitos BSD foram codificados com três bits, em vez de dois bits. A Tabela 3.8 apresenta a codificação 1 de 3 aplicada a cada dígito BSD. Tal codificação visa facilitar a análise dos dígitos x_{i-1} e y_{i-1} na primeira etapa da adição BSD, apresentada na Tabela 3.4.

Tabela 3.8: Codificação 1 de 3 dos dígitos BSD.

x_i	x_i^3	x_i^2	x_i^1
0	0	1	0
1	0	0	1
$\bar{1}$	1	0	0

Desta maneira, as tabelas-verdade do método de Reitwiener para a implementação da conversão de representação binária para representação BSD foram codificadas seguindo a codificação apresentada na Tabela 3.8. A Figura 3.19 apresenta o diagrama de blocos de um conversor 1 de 3 de binário para BSD para um operando de 4 bits.

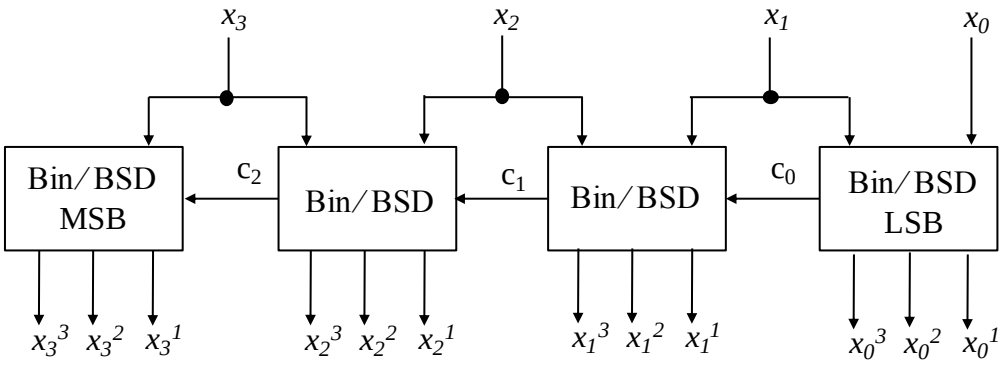


Figura 3.19: Conversor 1 de 3 de representação binária para representação BSD.

A Figura 3.20a apresenta o circuito lógico do bloco Bin/BSD MSB, que converte o bit mais significativo do operando. A Figura 3.20c mostra o circuito lógico do conversor para o bit menos significativo, enquanto que a Figura 3.20b mostra o circuito lógico do conversor para os demais bits.

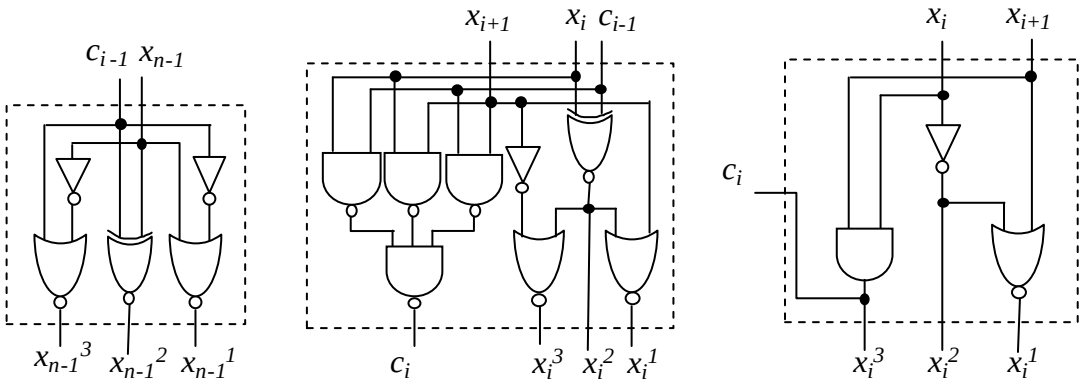


Figura 3.20: Circuitos Lógicos do Conversor 1 de 3 de binário para BSD.

3.6.2 Soma BSD 1 de 3

Assim como os BSDAs sem proteção apresentados anteriormente, o BSDA protegido com codificação 1 de 3 também executa a soma em duas etapas, onde a primeira gera os sinais de soma e *carry* intermediários e a segunda gera a soma final. Porém, para a implementação do BSDA 1 de 3, as Tabela 3.4 e Tabela 3.5 que apresentam as regras para a geração da soma BSD foram codificadas de acordo com a codificação 1 de 3 apresentada na Tabela 3.8.

A Figura 3.21 mostra o diagrama de blocos de um BSDA com codificação 1 de 3 para operandos de 4 bits.

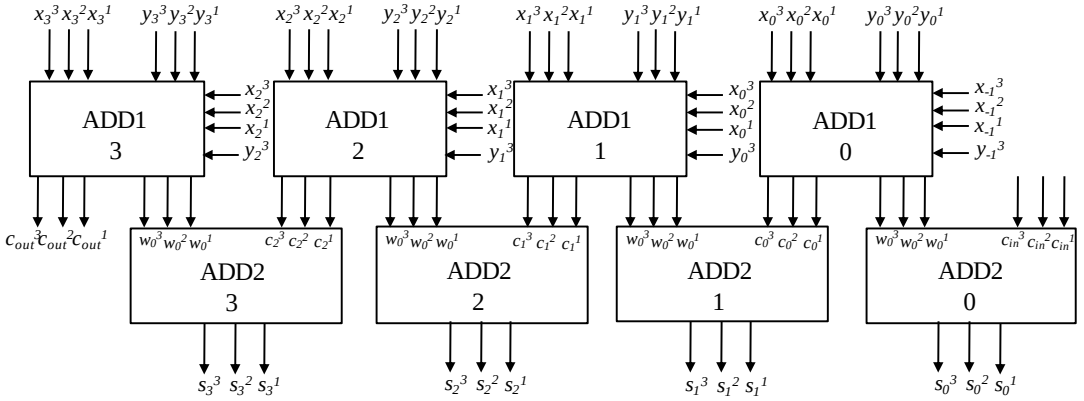


Figura 3.21: Somador BSD 1 de 3 de 4 bits.

3.6.2.1 ADD1 1 de 3 – Gerador da Soma/Carry Intermediários

Como no BSDA sem proteção, este bloco foi implementado a partir das regras de adição BSD apresentadas na Tabela 3.4, porém, como mencionado anteriormente, utilizando o código 1 de 3 para representar cada dígito BSD.

O bloco ADD1 foi implementado utilizando portas NAND, NOT e NOR. A Figura 3.22 apresenta o circuito lógico do bloco ADD1.

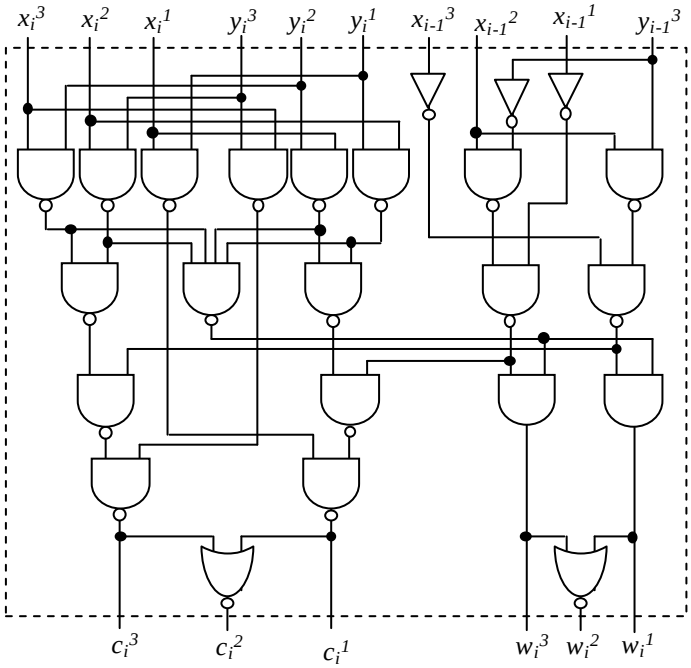


Figura 3.22: Gerador da Soma/*Carry* Intermediários – ADD1.

3.6.2.2 ADD2 – Gerador da Soma Final

Este bloco gera a soma final e tem como entrada a soma intermediária e o *carry* intermediário de uma posição menos significativa, w_i e c_{i-1} codificados com o código 1 de 3, provenientes do bloco ADD1. Ele foi implementado a partir da Tabela 3.5, onde os dígitos BSD também foram codificados em 1 de 3. A Figura 3.23 mostra o circuito lógico do bloco ADD2 que foi projetado apenas com portas NAND.

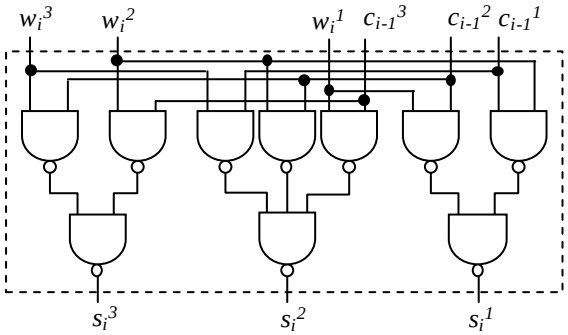


Figura 3.23: Gerador da Soma Final – ADD2

3.6.3 Verificador de Erros

O circuito verificador de erros confere se a soma final gerada pelo bloco ADD2 pertence ao código 1 de 3. Portanto, quando a saída deste circuito for "1", o resultado final é uma palavra pertencente ao código. Caso a saída seja "0", um erro ocorreu durante o cálculo da adição e a soma final gerada pelo bloco ADD2 não é uma palavra válida do código 1 de 3.

A Figura 3.24 apresenta o circuito lógico do Verificador de Erros, o qual foi projetado utilizando portas NAND, NOR, XOR e NOT.

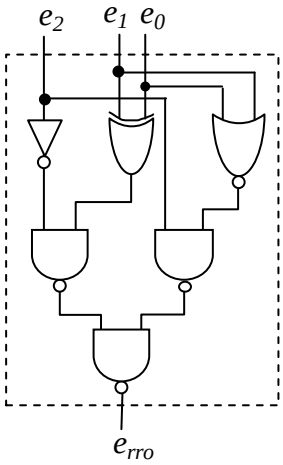


Figura 3.24: Verificador de Erros

3.6.4 Conversor BSD para Representação Binária

Como nas arquiteturas BSD apresentadas anteriormente, o BSDA protegido com código 1 de 3 também necessita ter o resultado final convertido para sua representação binária. As codificações escolhidas para codificar os dígitos BSD, tanto no BSDA sem proteção quanto no BSDA protegido com código 1de 3, permitem que o conversor de BSD para binário utilizado por ambos somadores seja o mesmo, apresentado na Figura 3.14.

A Tabela 3.9 mostra em destaque as entradas do conversor BSD para binário. Em ambas as versões, o conversor será alimentado pelos bits menos e mais significativos, que são iguais para as duas codificações.

Tabela 3.9: Codificações dos Dígitos BSD

Codificação Binária			Codificação 1 de 3		
x_i	x_i^-	x_i^+	x_i^3	x_i^2	x_i^1
0	**0**	**0**	**0**	1	**0**
1	**0**	**1**	**0**	0	**1**
$\bar{1}$	**1**	**0**	**1**	0	**0**

A Figura 3.25 apresenta o diagrama de blocos do BSDA protegido com codificação 1 de 3.

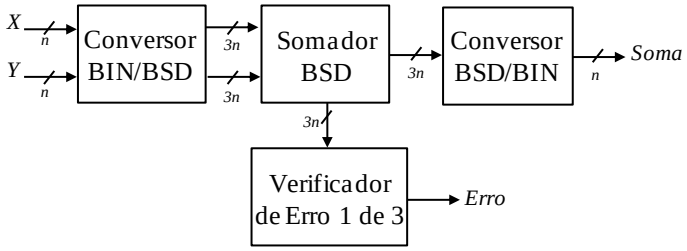


Figura 3.25: Somador BSDA protegido com codificação 1 de 3.

3.7 *Binary Signed Digit Adder* com Codificação 1 de 3 – Operandos representados em Complemento de dois (BSDA1-3_C2)

Assim como o BSDA sem proteção, foi implementada uma versão do BSDA com codificação 1 de 3 com entradas representadas em complemento de dois. Desta forma, os circuitos que compõem o BSDA com codificação 1 de 3 podem ser simplificados, visto que o conjunto de possibilidades de entradas é reduzido.

Esta versão também é *carry-free*, isto é, a soma é executada em duas etapas. Primeiramente, são gerados a soma e o *carry* intermediários de forma que não exista a propagação de *carry* na geração da soma final.

3.7.1 Conversor de Representação Binária para BSD

Nesta versão não há necessidade de um conversor propriamente dito, e sim um circuito que codifique as entradas em complemento de dois, de acordo com o código 1 de 3 apresentado na Tabela 3.9. A Figura 3.26 apresenta os circuitos que executam a conversão das entradas em complemento de dois em palavras pertencentes ao código 1 de 3 (FRANCO, 2008).

A Figura 3.26a mostra que a codificação do bit mais significativo é similar, porém diferente dos *n-1* bits dos operandos de entrada apresentados pela Figura 3.26b.

(a) (b)

Figura 3.26: Codificação 1 de 3 das entradas em complemento de dois.

3.7.2 Somador BSD 1 de 3 – Entradas em Complemento de Dois

Nesta versão do BSDA com codificação 1 de 3, a soma também é executada em duas etapas, como nas demais versões. Porém, os blocos responsáveis por executar a soma podem ser simplificados pois, devido ao fato dos operandos estarem representados em complemento de 2, as possibilidades de entrada são reduzidas. A Figura 3.27 apresenta o diagrama de blocos de um BSDA 1 de 3 com entradas representadas em complemento de dois de 4 bits.

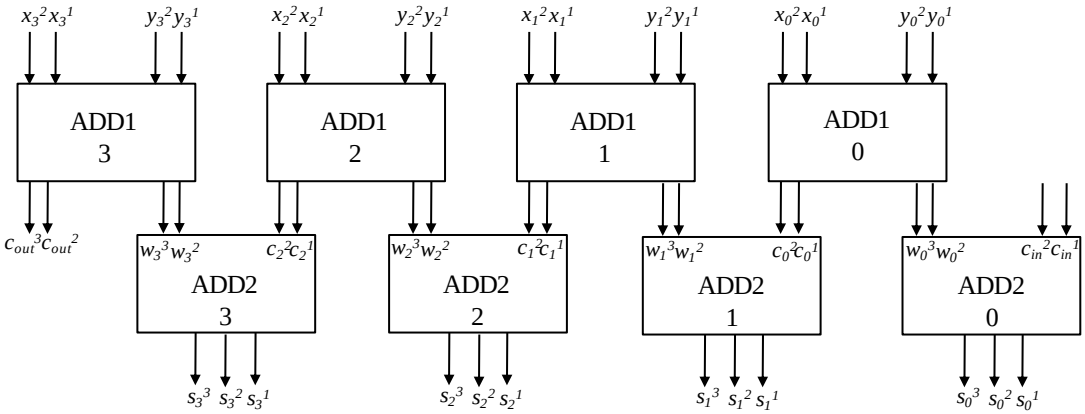


Figura 3.27: Somador BSD 1 de 3 de 4 bits.

3.7.2.1 ADD1 1 de 3 – Gerador da Soma/Carry Intermediários

Nesta versão do BSDA com codificação 1 de 3, o bloco ADD1 também pode ser simplificado devido à redução do conjunto de possibilidades de entradas e por não haver a necessidade da análise dos bits da próxima posição menos significativa para cada bit dos operandos. Portanto, por apresentar uma lógica reduzida, este bloco foi projetado sem lógica compartilhada, o que garante uma maior confiabilidade em relação à detecção das falhas.

A Tabela 3.10 apresenta as regras para a geração da soma e do *carry* intermediários para o somador BSDA com código 1 de 3 para entradas em complemento de dois. Os sinais de soma e *carry* intermediários estão codificados em palavras pertencentes ao código 1 de 3. Além de reduzir o conjunto de possibilidades de entradas, nesta versão os sinais c_i^3 e w_i^1 gerados pelos bits menos significativos poderão ser eliminados, pois serão sempre iguais a "0", assim como os sinais c_{n-1}^1 e w_{n-1}^1 gerados pelo bit mais significativo.

Tabela 3.10: Regras para a primeira etapa da adição carry-free com codificação 1 de 3 para operandos representados em complemento de dois.

Bit Mais Significativo				**Bits Menos Significativos**			
x_{n-1}	y_{n-1}	$c_{n-1}^3\ c_{n-1}^2\ c_{n-1}^1$	$w_{n-1}^3\ w_{n-1}^2\ w_{n-1}^1$	x_i	y_i	$c_i^3\ c_i^2\ c_i^1$	$w_i^3\ w_i^2\ w_i^1$
0	0	010	010	0	0	010	010
0	1	010	100	0	1	001	100
1	0	010	100	1	0	001	100
1	1	100	010	1	1	001	010

A Figura 3.28 apresenta o circuito que executa a geração da soma e do *carry* intermediários para os bits menos significativos, ao passo que a Figura 3.29 apresenta o circuito que gera a soma e o *carry* intermediários para o bit mais significativo. Ambos foram projetados a partir da Tabela 3.10.

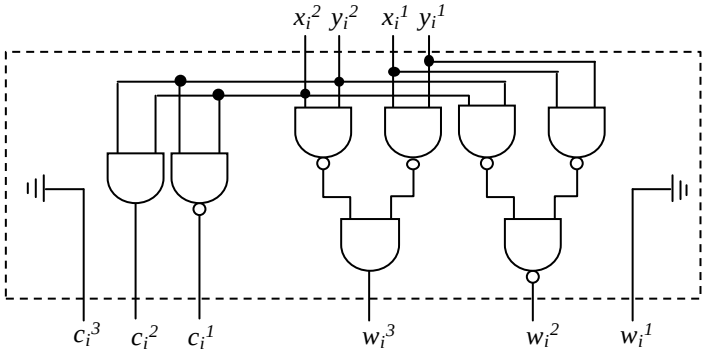


Figura 3.28: Gerador da Soma/*Carry* Intermediários para os bits menos significativos.

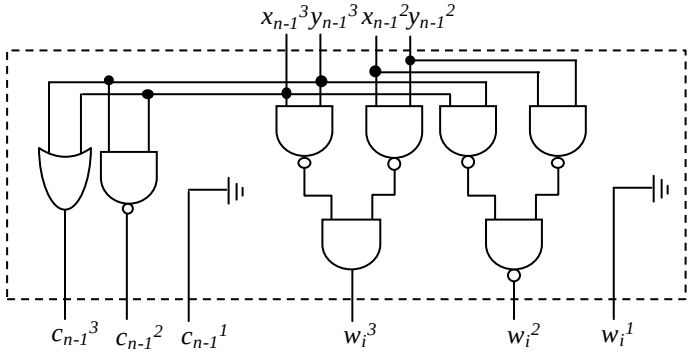


Figura 3.29: Gerador da Soma/*Carry* Intermediários para o bit mas significativo.

3.7.2.2 ADD2 – Gerador da Soma Final

Assim como nas demais versões, este bloco recebe as saídas dos blocos ADD1 e gera a soma final. Devido às simplificações mencionadas anteriormente, a tabela-verdade para a implementação do circuito lógico que gera a soma apresentada pela

Tabela 3.11 também pode ser simplificada, visto que a possibilidade da geração de um $c_{i-1} = \bar{1}$ existe apenas no bit mais significativo. Além disto, os sinais w_i^1 e c_{i-1}^3 serão sempre iguais a 0.

Tabela 3.11: Tabela-verdade para a geração da soma final para operandos representados em complemento de dois.

$w_i^3\ w_i^2\ w_i^1$	$c_{i-1}^3\ c_{i-1}^2\ c_{i-1}^1$	$s_i^3\ s^2\ s_i^1$
010	010	010
010	001	001
100	010	100
100	001	010

A Figura 3.30 apresenta o circuito lógico simplificado que executa a soma final na versão do somador BSDA com codificação 1 de 3 com entradas representadas em complemento de dois.

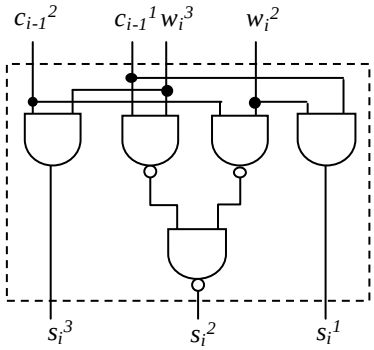


Figura 3.30: Gerador da Soma Final – ADD2.

3.7.3 Verificador de Erros e Conversor de Representação BSD para Binário

Os circuitos Verificador de Erros e Conversor da representação BSD para binário utilizados nesta versão são os mesmos apresentados na versão do BSD 1 de 3 com entradas em representação BSD (Figura 3.24 e Figura 3.14, respectivamente).

3.8 *Binary Signed Digit Adder* com Verificação de Paridade (BSDAPar_BSD)

O somador *Binary Signed Digit* com entradas representadas em BSD também foi protegido contra falhas através da aplicação da técnica de verificação de paridade, explorando a redundância existente no sistema de representação BSD (CARDARILLI et al., 2003). Como mencionado anteriormente, o sistema BSD possui um conjunto de três dígitos, $\{-1,0,1\}$, e a representação binária de cada um destes é realizada com dois bits, como mostra a Tabela 3.9. Caso ocorra uma falha em um dos bits que representa cada dígito, esta será detectada, pois isto acarretará em uma mudança da paridade.

Com a codificação binária escolhida para representar os dígitos BSD permite a dedução de propriedades da operação de adição em relação à verificação de paridade (CARDARILLI et al., 2003). Assumindo que $P(x_i)$ e $P(y_i)$ são as paridades dos operandos de entrada, então:

$$P(w_i) = P(x_i) \oplus P(y_i)$$

A Tabela 3.12 mostra os valores da paridade de $P(x_i)$, $P(y_i)$ e $P(w_i)$ e comprova a validade da propriedade apresentada acima.

Tabela 3.12: Valores de $P(x_i)$, $P(y_i)$ e $P(w_i)$.

x_i	y_i	x_{i-1} y_{i-1}	w_i	$P(x_i)$	$P(y_i)$	$P(x_i) \oplus P(y_i)$	$P(w_i)$
01	01	____	00	1	1	**0**	**0**
01	00	Ambos são positivos	10	1	0	**1**	**1**
01	00	Pelo menos um negativo	01	1	0	**1**	**1**
01	10	____	00	1	1	**0**	**0**
00	00	____	00	0	0	**0**	**0**
00	10	Ambos são positivos	10	0	1	**1**	**1**
00	10	Pelo menos um negativo	01	0	1	**1**	**1**
10	10	____	00	1	1	**0**	**0**

Desta forma, uma falha nas entradas do bloco ADD1 de somador pode ser detectada através da verificação da paridade $P(w_i)$. Uma falha em um dos bits dos operandos pode alterar o valor de w_i, c_i, w_{i+1} e c_{i+1}. Porém, w_{i+1} e c_{i+1} somente serão alterados de $\bar{1}$ e 0 para 1 e $\bar{1}$, respectivamente, e vice-versa, ou então, de 1 e 0 para $\bar{1}$ e 1 e vice-versa. Portanto, o valor da paridade de w_{i+1} não muda, pois $P(1) = P(\bar{1})$ (CARDARILLI et al., 2003). Porém, no que diz respeito à w_i, uma falha em um dos operandos de entrada sempre irá alterar o seu valor e, consequentemente, a sua paridade. Assumindo que $P(w_i)$, $P(c_{i-1})$ e $P(s_i)$ são, respectivamente, as paridades da soma intermediária, do *carry* intermediário e da soma final, tem-se que:

$$P(s_i) = P(w_i) \oplus P(c_{i-1})$$

A Tabela 3.13 apresenta os valores da paridade de $P(s_i)$, $P(w_i)$ e $P(c_{i-1})$ e mostra que a paridade da soma final é igual à paridade entre soma intermediária e o *carry* intermediário.

Tabela 3.13: Valores de $P(s_i)$, $P(w_i)$ e $P(c_{i-1})$.

w_i	c_{i-1}	s_i	$P(w_i)$	$P(c_{i-1})$	$P(w_i) \oplus P(c_{i-1})$	$P(s_i)$
10	10	__	1	1	**0**	**__**
10	00	10	1	0	**1**	**1**
10	01	00	1	1	**0**	**0**
00	10	10	0	1	**1**	**1**
00	00	00	0	0	**0**	**0**
00	01	01	0	1	**1**	**1**
01	10	00	1	1	**0**	**0**
01	00	01	1	0	**1**	**1**
01	01	__	1	1	**0**	**__**

Com a verificação da paridade $P(s_i)$ podem ser detectadas falhas nas saídas do bloco ADD1(ou entradas do bloco ADD2). Neste caso a falha afeta apenas um bloco e altera o

valor de $P(s_i)$. Também são detectadas falhas nas saídas do bloco ADD2, pois estas também irão alterar a paridade $P(s_i)$.

A Figura 3.31 apresenta o somador BSDA protegido contra falhas através da técnica de verificação de paridade. Ao BSDA não-protegido foram acrescentados o bloco Preditor de Paridade que gera os valores de $P(c_i)$, o bloco Indicador de Erro *Pw*, que verifica se $P(w_i) = P(x_i) \oplus P(y_i)$ e gera um sinal indicativo de erro, caso tenha ocorrido uma falha, e o bloco Indicador de Erro *Ps*, que verifica se $P(s_i) = P(w_i) \oplus P(c_{i-1})$ e também gera um sinal de erro no caso da ocorrência de falha.

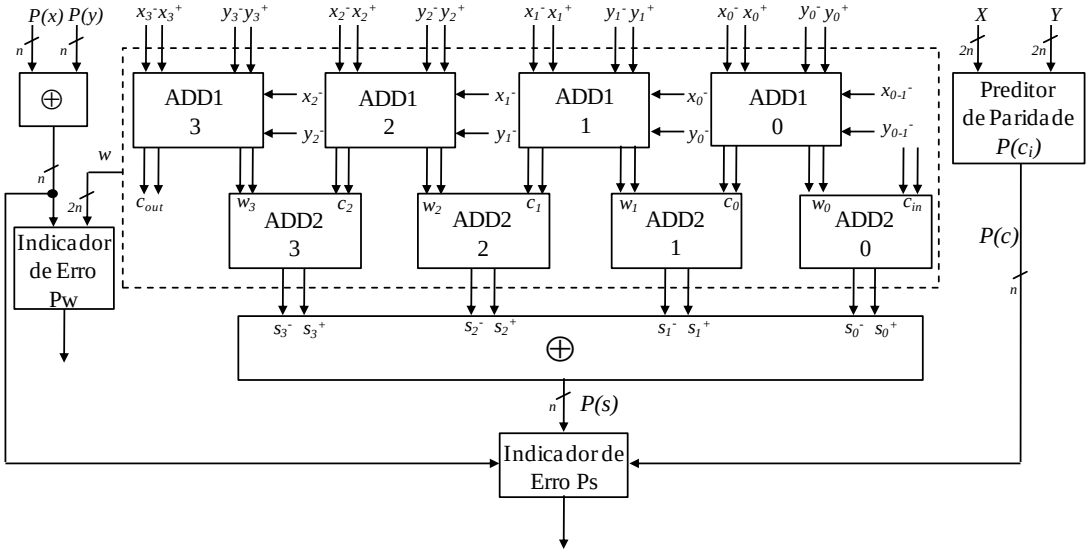


Figura 3.31: Somador BSDA com verificação de paridade.

A geração de $P(x_i)$, de $P(y_i)$, de $P(w_i)$ e de $P(s_i)$, bem como os blocos **Indicadores de Erros** *Pw* e *Ps* foram implementados através de portas XOR. Já o bloco **Preditor de Paridade** foi implementado a partir das tabelas apresentadas anteriormente que contêm as regras para a geração da soma e *carry* intermediários. Em tais tabelas, pode ser visto que o cálculo de c_i depende do valor de x_i^-, x_i^+, y_i-, y_i^+, x_{i-1}^- e y_{i-1}^-. Portanto, sua paridade, $P(c_i)$, também dependerá destes sinais. A Figura 3.32 apresenta o circuito lógico do bloco **Preditor de Paridade**.

A paridade $P(c_i)$ poderia ser gerada através das saídas c_i do bloco ADD1. Porém, para uma melhor confiabilidade da técnica de proteção, não foi utilizada lógica compartilhada.

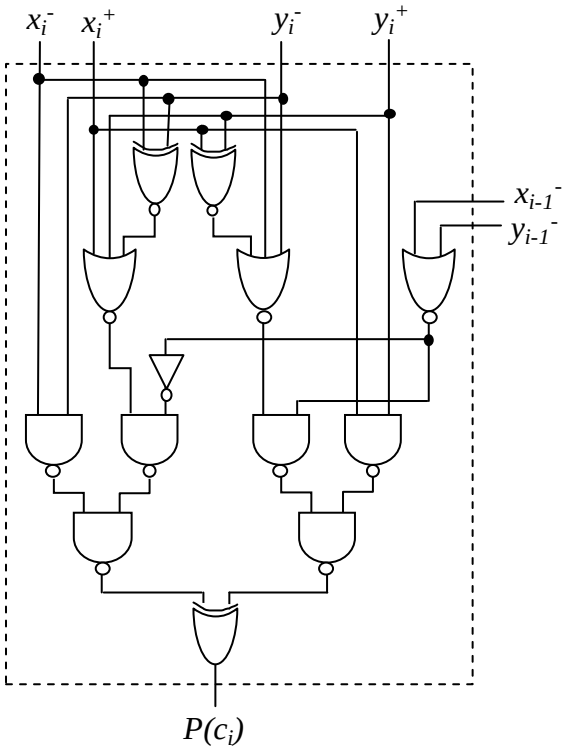


Figura 3.32: Preditor de Paridade $P(c_i)$

3.9 *Binary Signed Digit Adder* com Verificação de Paridade – Operandos representados em complemento de dois (BSDAPar_C2)

A técnica de verificação de paridade também foi aplicada ao somador BSDA com entradas representadas em complemento de 2. Apesar dos dígitos de entrada do somador não serem representados por dois bits, como no caso do somador BSDA com entradas representadas em BSD, as propriedades da operação de adição em relação à verificação de paridade apresentadas anteriormente são válidas. Portanto, para este somador também se tem

$$P(w_i) = P(x_i) \oplus P(y_i)$$

Porém como nesta versão os operandos de entrada não necessitam ser representados por dois bits, $P(x_i) = x_i$ assim como $P(y_i) = y_i$. A Tabela 3.14 mostra que a paridade entre os operandos de entrada é igual à paridade da soma intermediária comprovando que esta propriedade também é válida para esta versão do somador BSDA.

Tabela 3.14: Valores da Paridade entre os operandos de entrada

x_i	y_i	$w_i^- w_i^+$	$P(x_i) \oplus P(y_i)$	$P(w_i)$
0	0	00	0	0
0	1	10	1	1
1	0	10	1	1
1	1	00	0	0

Porém como o sinal w_i^+ não é gerado nesta versão por ser sempre igual a 0, não é necessário a geração de $P(w_i)$ pois $P(w_i) = w_i^-$. Portanto, primeiramente foram geradas

as paridades de cada operando independentemente, *PX* e *PY*, para então ser gerada a paridade entre os mesmos. Também foi gerada a paridade da soma intermediária *PW*, isto é, a paridade dos sinais w_i^- para assim verificar se $PW = PX \oplus PY$, caso resulte em uma desigualdade é gerado um sinal indicativo de erro. Através desta verificação, pode ser detectada uma falha que tenha ocorrido no bloco ADD1 na geração da soma intermediária.

Na saída do bloco ADD2 do somador BSDA com operando em complemento de dois também é possível verificar que a paridade da soma final é igual à paridade entre soma intermediaria e o *carry* intermediário

$$P(s_i) = P(w_i) \oplus P(c_{i-1})$$

Assim como o sinal w_i^+, o sinal c_{i-1}^- não é gerado nesta versão do somador, pois também será sempre igual a 0, logo $P(c_{i-1}) = c_{i-1}^+$. Na Tabela 3.15 pode ser visto que $P(w_i) = w_i^-$ e $P(c_{i-1}) = c_{i-1}^+$ portanto $P(s_i) = w_i^- \oplus c_{i-1}^+$.

Tabela 3.15: Valores de $P(s_i)$, $P(w_i)$ e $P(c_{i-1})$.

$w_i^- w_i^+$	$c_{i-1}^- c_{i-1}^+$	$s_i^- s_i^+$	$P(w_i)$	$P(c_{i-1})$	$\overline{w_i^- \oplus c_{i-1}^+}$	$P(s_i)$
00	00	00	0	0	0	0
00	01	01	0	1	1	1
10	00	10	1	0	1	1
10	01	00	1	1	0	0

Assim como na versão do somador com operandos representados em BSD NAF o sinal c_{i-1}^- usado para a verificação da paridade é gerado pelo Preditor de Paridade para evitar o uso de lógica compartilhada com o objetivo de obter uma melhor confiabilidade da técnica de proteção. A Figura 3.33 apresenta o Preditor de Paridade $P(c_i)$ utilizado por esta versão do somador BSDA.

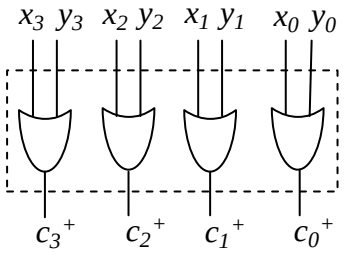


Figura 3.33: Preditor de Paridade $P(c_i)$.

4 RESULTADOS EXPERIMENTAIS

Este capítulo apresenta e discute os resultados experimentais, obtidos por meio de simulações no nível elétrico das 9 arquiteturas de somadores detalhadas no capítulo 3. Para gerá-los, os somadores foram mapeados manualmente para tecnologia CMOS assumindo uma biblioteca de células cujos elementos estão enumerados na Tabela 4.1. Destes elementos, a XOR e a XNOR não são portas CMOS estáticas complementares.

Tabela 4.1: Biblioteca de células para o mapeamento dos somadores

Célula	Portas CMOS utilizadas	Nº de entradas	Nº de transistores
INV	inversor	1	2
NAND	NAND	2	4
NAND3	NAND3	3	6
NAND4	NAND4	4	8
NAND5	NAND5	5	10
NAND6	NAND6	6	12
AND	NAND + inversor	2	6
AND3	NAND3 + inversor	3	8
AND4	NAND4 + inversor	4	10
NOR	NOR	2	4
NOR2	NOR3	3	6
NOR4	NOR4	4	8
OR	OR + inversor	2	4
OR3	NOR + inversor	3	8
OR4	NOR3 + inversor	4	10
XNOR	XNOR	2	10
XOR	XOR	2	6

A fim de garantir a qualidade dos resultados das simulações, as portas listadas na Tabela 4.1 foram descritas em Spice assumindo-se os dimensionamentos de transistores utilizados na Biblioteca de células "Nangate 45nm Open Cell Library" (NangateOpenCellLibrary_PDKv1_3_v2009_07) da empresa Nangate (NANGATE, 2011).

Os somadores mapeados foram descritos, com o auxilio de uma ferramenta implementada em C, no formato Spice para 4 comprimentos de operandos de entrada, 4, 8, 16 e 32 bits, originando um total de 36 descrições. As descrições foram simuladas no nível elétrico utilizando o simulador HSpice (SYNOPSYS, 2011), assumindo-se o

Modelo Tecnológico Preditivo (*Predictive Tecnology Model* - PTM) de 45nm (ZHAO; CAO, 2006).

A validação do funcionamento dos somadores foi feita por meio de simulação funcional utilizando a ferramenta CosmosScope da Synopsys, na versão 4.0. A determinação do atraso crítico foi feita mediante a análise dos resultados das simulações no nível elétrico.

A fim de gerar conjuntos de vetores aleatórios a serem usados nas simulações para estimar a potência média consumida pelos somadores, foi utilizada uma função do HSpice chamada *Pseudo Random Bit Generator Source* (PRBS), a qual utiliza fontes de tensão do tipo *Linear Feedback Shift Register* (LFSR) (SYNOPSYS, 2011). A frequência escolhida para ser usada nas fontes LFSR foi calculada a partir do maior atraso dos somadores. Para obter a potência média consumida, foram realizadas simulações com 1000 vetores de entrada, visto que testes com 5000 e 10000 vetores não apresentaram mudanças significativas nos resultados de potência média. Detalhes dos comandos HSpice usados na medida da potência média, incluindo os parâmetros usados no LFSR, são mostrados no Apêndice B.

Para a análise da proteção dos somadores, foram realizadas campanhas de injeção de falhas por meio de simulações de falhas (ALEXANDRESCU; ANGHEL; NICOLAIDIS, 2004), também usando o simulador de nível elétrico HSpice. Para cada somador, foi realizada uma simulação para cada falha transiente possível e as saídas foram analisadas para identificar se a falha se propaga até alguma das saídas primárias. A injeção da falha foi simulada por meio de uma fonte de corrente que modela a dupla exponencial de Messenger (1982), devidamente calibrada para a tecnologia PTM de 45nm.

As seções 4.1 e 4.2 apresentam o atraso, o número de transistores utilizados pelos somadores e a potência média, obtidos por simulações HSpice para as arquiteturas consideradas. Por questão de concisão, a partir deste ponto, se usará a palavra "potência" para fazer menção à potência média. É importante ressaltar que todos os gráficos apresentados a seguir apresentam curvas geradas por regressão linear, uma vez que se identificou, a priori, que os dados apresentam um comportamento praticamente linear.

A seção 4.3 apresenta os resultados das campanhas de injeção de falhas.

4.1 Análise das Arquiteturas utilizando Conversores de Entrada e Saída

Os resultados apresentados nesta seção foram obtidos a partir das arquiteturas BSDAs completas apresentadas no capítulo 3, isto é, levando em consideração os conversores de entrada e saída necessários para a conversão entre os sistemas numéricos BSD e complemento de dois, além dos circuitos que executam a soma BSD e os circuitos que implementam as técnicas de tolerância.

A Tabela 4.2 apresenta os resultados de atraso crítico, número de transistores e potência para as arquiteturas que utilizam somadores do tipo RCA (*Ripple-Carry Adder*) e BSDA (*Binary Signed Digit Adder*). As arquiteturas destacadas em vermelho são as arquiteturas não-protegidas de cada tipo de somadores. Os gráficos que mostram o atraso crítico, número de transistores e potência são apresentados, respectivamente, nas Figura 4.1, Figura 4.2 e Figura 4.3.

Tabela 4.2: Resultados de Atraso Crítico, Número de Transistores e Potência

	Atraso (ns)				Número de Transistores				Potência (mW)			
Arquitetura	**4**	**8**	**16**	**32**	**4**	**8**	**16**	**32**	**4**	**8**	**16**	**32**
RCA	0,287	0,423	0,704	1,302	120	240	480	960	0,016	0,033	0,068	0,139
RCATMR	0,441	0,577	0,858	1,455	432	864	1728	3456	0,060	0,128	0,261	0,528
RCAIOI	0,326	0,477	0,786	1,407	798	1550	3054	6062	0,120	0,230	0,449	0,884
BSDA_BSD	0,845	1,107	1,648	2,728	684	1388	2796	5612	0,057	0,123	0,246	0,501
BSDA1-3_BSD	0,873	1,137	1,677	2,754	926	1862	3734	7478	0,087	0,188	0,380	0,778
BSDAPAR_BSD	0,845	1,107	1,648	2,728	1110	2326	4758	9640	0,078	0,167	0,335	0,682
BSDA_C2	0,485	0,620	0,903	1,469	256	480	928	1824	0,027	0,051	0,099	0,197
BSDA1-3_C2	0,499	0,634	0,916	1,483	512	992	1952	3872	0,057	0,103	0,191	0,371
BSDAPAR_C2	0,485	0,620	0,903	1,469	424	840	1672	3336	0,044	0,088	0,173	0,347

Ao aplicar as técnicas de proteção nas arquiteturas citadas acima, pode ser visto nos dados apresentados na Tabela 4.2 e no gráfico da Figura 4.1 que ocorre um aumento no atraso crítico dos somadores em relação as suas versões não-protegidas. Assim o do RCATMR é, em média, 31% maior e o atraso do RCAIOI é, em média, 12% maior do que o atraso do RCA. (Note-se que o circuito verificador de paridade dos *carries* redundantes existente no RCAIOI opera em paralelo com o registrador de saída e portanto, não influencia no atraso crítico.)

Também pode ser visto na Figura 4.1 e na Tabela 4.2 que o BSDAPar_BSD não apresenta aumento no atraso crítico em relação ao BSDA_BSD, dado que a verificação da paridade executada pela técnica ocorre em paralelo ao cálculo da soma BSD, o mesmo ocorre com o atraso do BSDAPar_C2 em relação ao atraso do BSDA_C2. Já quando o código 1 de 3 é aplicado, o atraso crítico do somador continua praticamente o mesmo, no entanto o BSDA1-3_BSD fica, em média, 2% mais lento que o BSDA_BSD (o BSDA1-3_BSD também é 2% mais lento que o BSDAPar_BSD uma vez que este apresenta o mesmo atraso crítico que o BSDA_BSD). O aumento no atraso do BSDA1-3_C2 em relação ao BSDA_C2(e BSDAPar_C2) também é de, em média, 2%.

Apesar dos somadores mostrados acima serem do tipo *carry-free* (exceto o RCA, o RCATMR e o RCAIOI), como mencionado no capítulo 3, e portanto possuírem atraso constante independentemente do comprimento dos operandos de entrada, nota-se que os valores apresentados na Tabela 4.2 e as curvas mostradas na Figura 4.1 não são constantes. Isso ocorre devido ao uso de conversores necessários para a conversão do sistema de representação binário convencional para representação redundante BSD NAF, sendo que o atraso desses conversores é mais significativo que o atraso do circuito que executa a soma propriamente dita.

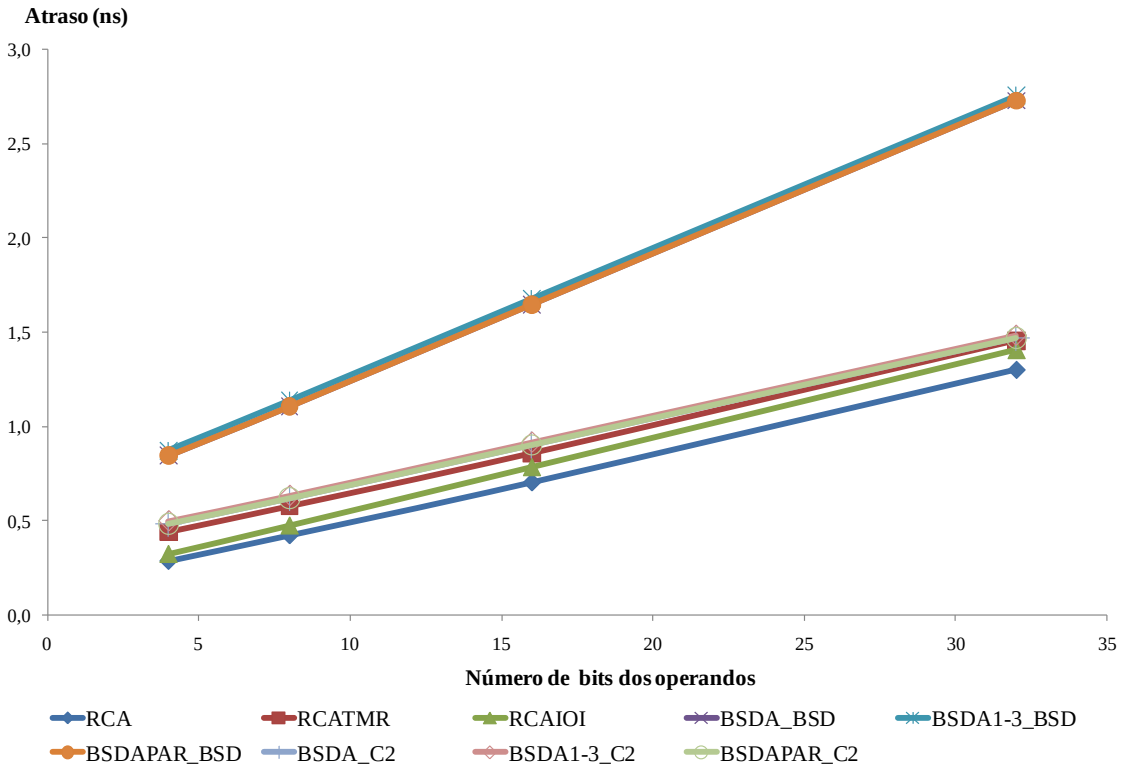


Figura 4.1: Atraso crítico das arquiteturas considerando o uso de conversores

Vale lembrar que nos somadores BSDA_BSD, BSDA1-3_BSD e BSDAPar_BSD há um conversor para cada operando de entrada e um conversor para o resultado. Desta forma o atraso crítico destes somadores será composto pela soma entre o atraso de um conversor de entrada, o atraso da soma BSD e o atraso do conversor de saída. Como o atraso crítico de cada conversor é semelhante ao atraso de um RCA de igual comprimento os somadores BSDA_BSD, BSDA1-3_BSD e BSDAPar_BSD apresentam, portanto os maiores atrasos crítico dentre as arquiteturas estudadas.

Assim, a partir dos dados mostrados na Tabela 4.2, pode-se verificar que os somadores BSDA_BSD, BSDA1-3_BSD e BSDAPar_BSD apresentam um atraso praticamente idêntico e, em média, 92% maior que RCATMR e 126% maior que o RCAIOI (considerando que nenhuma falha transiente tenha ocorrido no RCAIOI, uma vez que este executa a soma novamente quando uma falha é detectada, alterando desta forma o atraso crítico do mesmo). Já em relação ao somador RCA, que não possui nenhum tipo de proteção e apresenta o menor atraso crítico dentre todas as arquiteturas, o atraso destes somadores é, em média, 152% maior.

Também é possível observar na Figura 4.1 e na Tabela 4.2 que os somadores RCA, RCATMR e RCAIOI também apresentam atraso crítico menor que os somadores BSDA_C2, BSDA1-3_C2 e BSDAPar_C2, que assim como os somadores BSDAs com entradas representadas em BSD NAF possuem atrasos críticos semelhantes. Além disso, observando o gráfico da Figura 4.1 nota-se que o atraso crítico destes somadores é praticamente identido ao atraso crítico do RCATMR, sendo, em média, apenas 7% maior. Isso ocorre porque no RCATMR o atraso crítico é composto pelo atraso de um RCA(que é mais significativo) e do votador, que é constante independentemente do comprimento dos operandos de entrada das arquiteturas ao passo que o atraso dos somadores BSDA_C2, BSDA1-3_C2 e BSDAPar_C2 é composto pelo atraso da soma BSD, que também é contante, e pelo atraso do conversor de saída, sendo que este ultimo

equivale praticamente ao atraso de um RCA. Desse modo o atraso inserido pelo conversor de saída utilizado pelos somadores BSDA_C2, BSDA1-3_C2 e BSDAPar_C2 torna-se dominante em relação ao atraso constante do circuito que executa a soma BSD e praticamente iguala o atraso crítico destes somadores ao atraso crítico do RCATMR.

Já a diferença entre o atraso dos somadores BSDA_C2, BSDA1-3_C2 e BSDAPar_C2 e o atraso do RCAIOI é mais representativa como mostra o gráfico da Figura 4.1, sendo o atraso destes somadores, em média, 25% maior que o atraso do RCAIOI. No entanto esta vantagem do RCAIOI em relação a estes somadores é relativa pois, como mencionado anteriormente, caso seja detectada uma falha no RCAIOI a soma será executada novamente, o que altera o atraso crítico deste somador. Em relação ao RCA, o atraso dos somadores BSDA_C2, BSDA1-3_C2 e BSDAPar_C2 é, em média, 40% maior.

A semelhança no atraso dos somadores BSDA_C2, BSDA1-3_ C2 e BSDAPar_ C2 mencionada acima e observada no gráfico da Figura 4.1 é devido ao atraso do conversor de saída, utilizado por estes somadores, que torna-se dominante em relação ao atraso do circuito que executa a soma propriamente dita. Isso também ocorre com os somadores BSDA_BSD, BSDA1-3_BSD e BSDAPar_BSD, no entanto como estes necessitam também de um conversor de entrada apresentam portanto um atraso, em média, 80% maior que os somadores BSDA_C2, BSDA1-3_C2 e BSDAPar_C2.

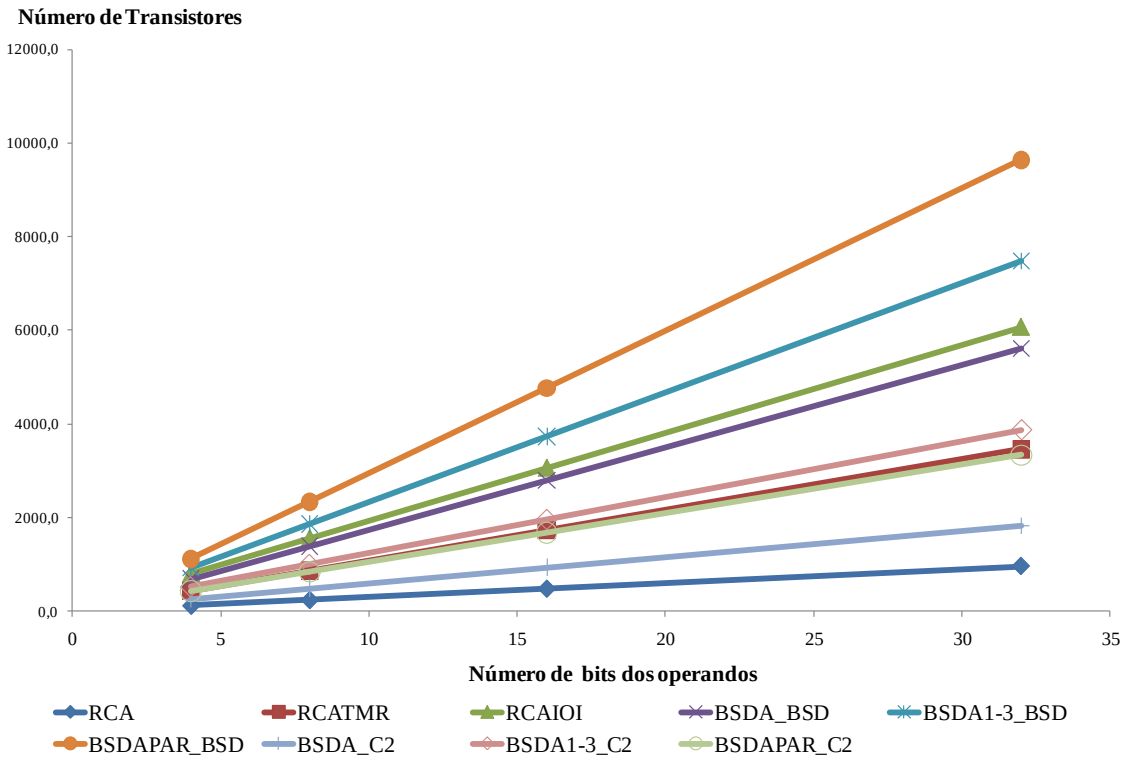


Figura 4.2: Números de transistores utilizados pelas arquiteturas considerando o uso de conversores

A necessidade do uso de conversores também influencia diretamente na quantidade de transistores utilizados pelas arquiteturas. Desta forma, o BSDA_BSD utiliza, em média, 479% mais transistores do que o RCA, como mostra a Figura 4.2. Isso porque cada conversor utilizado pelo BSDA_BSD equivale praticamente a um RCA em

número de transistores, e o BSDA_BSD utiliza três conversores (dois para os operandos de entrada e um para a saída) e ainda possui a lógica que executa a soma propriamente dita.

Além disso, observa-se que o BSDA_C2 também utiliza mais transistores que o RCA, no entanto a diferença entre a quantidade de transistores utilizada não é tão significativa, como pode ser visto na Figura 4.2 e através dos dados da Tabela 4.2, sendo que o BSDA_C2 utiliza, em média, 99% mais transistores que o RCA. Isso porque o BSDA_C2 necessita apenas de um conversor na saída. Em contrapartida, o BSDA_C2 utiliza uma quantidade de transistores significativamente menor que o BSDA_BSD, pois além de não necessitar de conversores de entrada, o bloco que calcula a soma propriamente dita do BSDA_C2 utiliza menos transistores que o bloco que calcula a soma do BSDA_BSD. Assim o BSDA_BSD utiliza, em média, 191% mais transistores que o BSDA_C2.

Ao aplicar as técnicas de proteção nas arquiteturas, pode ser visto nos dados apresentados na Tabela 4.2 e no gráfico da Figura 4.2 que ocorre um aumento na utilização de transistores, o que era esperado, uma vez que para a implementação de tais técnicas é necessária alguma forma de redundância de *hardware*. Desta forma, o RCATMR apresenta um acréscimo de, em média, 260% no uso de transistores em relação ao RCA, já que a técnica TMR consiste na triplicação do bloco que se deseja proteger, neste caso, o próprio somador RCA. Também pode ser visto que o RCAIOI utiliza uma quantidade de transistores ainda maior do que o RCATMR. Isso porque o RCAIOI possui um circuito verificador de paridade, multiplexadores de entrada para cada operando, registrador e multiplexador de saída, além de árvores de portas CMOS XORs para a verificação da paridade dos operandos e dos *carries* redundantes. Além disso, pode ser visto na Figura 4.2 que o RCAIOI chega a utilizar, em média, 11% mais transistores que o BSDA_BSD apesar deste usar três conversores.

Também pode ser observado que, devido a lógica extra inserida pela aplicação da técnica RESI, o número de transistores utilizados pelo RCAIOI não é tão inferior à quantidade de transistores usados pelos somadores BSDA1-3_BSD e BSDAPar_BSD, que são as arquiteturas que utilizam mais transistores dentre todos os somadores apresentados. Assim, o BSDA1-3_BSD utiliza, em média, apenas 20% mais transistores que o RCAIOI e o BSDAPar_BSD usa, em média, 51% mais transistores que o RCAIOI. Já em relação aos somadores BSDAs com entradas representadas em complemento de 2, o RCAIOI apresenta um comportamento contrário, isto é, ele utiliza mais transistores que os mesmos. Logo, o RCAIOI utiliza, em média, 56% mais transistores que o BSDA1-3_C2 e 84% mais transistores que o BSDAPar_C2.

Em relação ao RCATMR, pode ser visto na Figura 4.2 que apesar da técnica TMR triplicar o RCA, este também utiliza menos transistores do que os somadores BSDA1-3_BSD e BSDAPar_BSD, no entanto a diferença entre o uso de transistores é bem mais representativa que o RCAIOI. Isso porque além dos somadores BSDA1-3_BSD e BSDAPar_BSD utilizarem os conversores de entrada e saída, a aplicação da codificação 1 de 3 e verificação de paridade aumenta consideravelmente a quantidade de transistores necessários para a implementação das técnicas de proteção. Assim, o BSDA1-3_BSD utiliza, em média, 116% mais transistores que o RCATMR. Já o BSDAPar_BSD utiliza, em média, 170% mais transistores que o RCATMR. Além disso, é importante resaltar que o RCATMR utiliza, em média, 3% mais transistores que o BSDAPar_C2 e o BSDA1-3_C2 utiliza, em média, apenas 15% mais transistores que o RCATMR.

Com a aplicação das técnicas de proteção, o BSDAPar_BSD utiliza, em média, 68% mais transistores que o somador BSDA_BSD. Já a aplicação da codificação 1 de 3 não resulta em um acréscimo tão significativo em relação ao número de transistores utilizados pelo BSDA_BSD, ficando o BSDA1-3_BSD, em média, 34% maior. Já os somadores BSDA1-3_C2 e BSDAPar_C2 utilizam, em média, 107% e 76% mais transistores que o BSDA_C2, respectivamente.

Outra observação importante que pode ser feita baseada na Tabela 4.2 e na Figura 4.2 é que a aplicação das técnicas que utilizam codificação 1 de 3 e verificação de paridade apresentam comportamento inverso quando são aplicadas nos somadores BSDA com entradas representadas em BSD NAF e ao serem aplicadas aos somadores BSDA com entradas representadas em complemento de 2. Ao serem aplicadas nos somadores com entradas representadas em BSD NAF pode ser visto que a técnica que utiliza verificação de paridade utiliza mais transistores que a técnica que utiliza codificação 1 de 3. Em média, o BSDAPar_BSD utiliza 25% mais transistores que o BSDA1-3_BSD. Já quando são aplicadas nos somadores com entradas representadas em complemento de 2, a técnica que utiliza a codificação 1 de 3 necessita mais transistores que a técnica de verificação de paridade. Assim o BSDA1-3_C2 utiliza, em média, 18% mais transistores que o BSDAPar_C2.

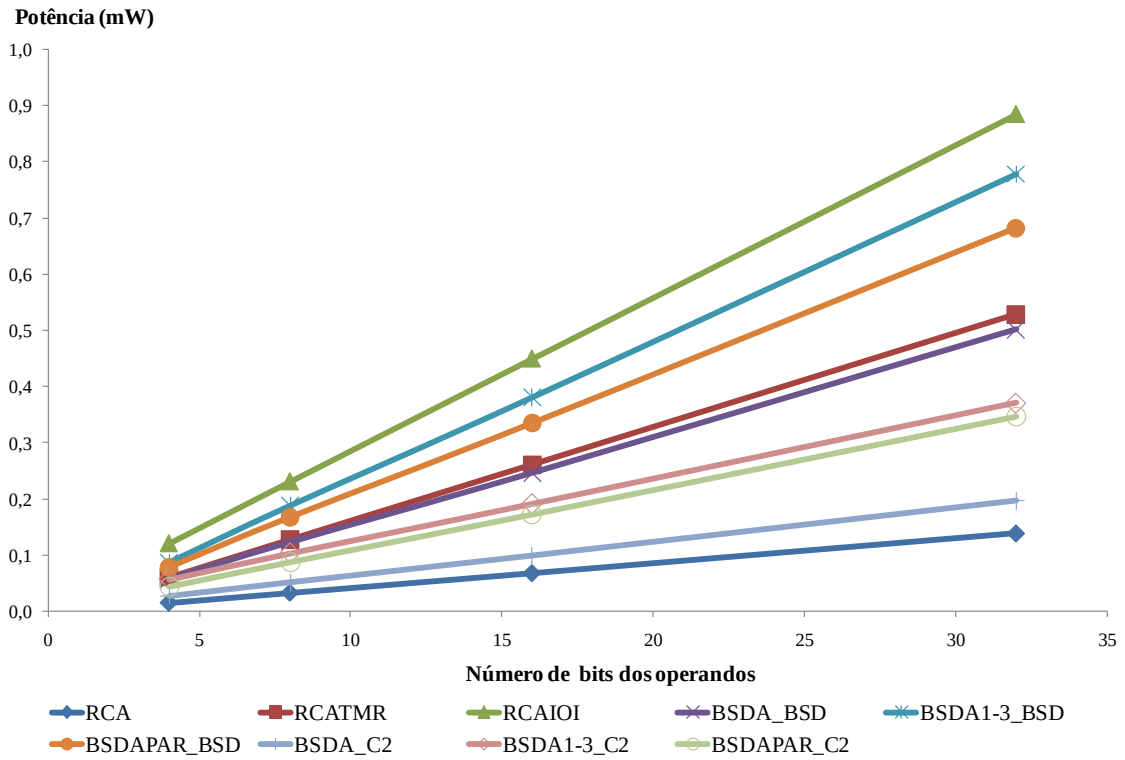


Figura 4.3: Potência consumida pelas arquiteturas considerando o uso de conversores

Analisando-se o consumo de potência das arquiteturas estudadas, pode ser observado na Figura 4.3 e na Tabela 4.2 que as versões protegidas consomem mais potência que os somadores não-protegidos RCA, BSDA_BSD e BSDA_C2, o que era esperado visto que a aplicação das técnicas de proteção, como mencionado anteriormente, resulta em uma maior quantidade de transistores, aumentando assim a atividade de chaveamento do somador. Desta forma, o RCATMR apresenta um acréscimo na potência consumida, em média, de 284%, como era esperado, visto que utiliza 260% mais transistores que o RCA. Já o RCAIOI consome, em média, 590%

mais potência do que o RCA uma vez que a aplicação da técnica RESI adiciona uma quantidade considerável de lógica extra ao RCA (em média 545% a mais de transistores). Comparando as duas versões protegidas, nota-se que o RCAIOI consome mais potência do que o RCATMR (em média 80% mais potência).

Em relação aos somadores BSDAs com entradas representadas em BSD NAF observa-se que a aplicação da codificação 1 de 3 resulta em um aumento, em média, de 54% no consumo de potência em relação ao BSDA_BSD e o BSDAPar_BSD apresenta um aumento de 36% em relação ao BSDA_BSD. Além disso, apesar de utilizar mais transistores que o BSDA1-3_BSD, o somador BSDAPar_BSD consumiu menos potência que este. Observa-se que o BSDA1-3_BSD consome, em média, 13% mais potência que o BSDAPar_BSD. Isso se deve ao modo como foi implementada a verificação de paridade dos operandos de entrada, da soma intermediária, da soma final e do *carry* intermediário gerado pelo preditor de paridade, que faz com que os blocos Indicador de Erro 1 e Indicador de Erro 2 não apresentem um consumo de potência significativo, pois suas entradas serão praticamente sempre alimentadas com 0 diminuido assim o número de chaveamentos de sinais do somador. Maiores explicações sobre essa implementação podem ser encontradas no Apêndice C.

No caso dos somadores BSDAs com entradas em complemento de 2 a aplicação da codificação 1 de 3 gera um aumento de, em média, 99% na potência consumida em relação ao BSDA_C2. Já o BSDAPar_C2 apresenta um consumo de potência, em média, 72% maior que consumo do BSDA_C2.

Considerando o consumo de potência apenas dos somadores não-protegidos, pode ser visto que apesar de utilizar, em média, 479% mais transistores que o RCA, o BSDA_BSD consumiu "apenas" 265% mais potência que o RCA. Isso acontece devido à codificação BSD NAF gerada pelos blocos conversores BIN/BSD (que converte operandos em complemento de 2 para BSD NAF), a qual reduz o consumo dos blocos ADD1 e ADD2 do BSDA_BSD que executam a soma propriamente dita e também reduz a potência consumida pelo bloco conversor BSD/BIN (que converte o resultado em BSD NAF para complemento de 2). Tal redução de potência tem como origem o menor número de chaveamentos de sinais que ocorre quando BSD NAF é adotada.

A fim de examinar em detalhes como o uso de BSD NAF influencia o consumo de potência do BSDA_BSD, os blocos que compõem este somador foram simulados separadamente com vetores de entrada aleatórios, isto é, sem a codificação BSD NAF gerada pelas saídas do conversor BIN/BSD. Eles também foram simulados conectados, de modo que o BSDA_BSD recebe como entrada a codificação BSD NAF da saída do conversor BIN/BSD. Assim, é possível concluir que a soma BSD propriamente dita, quando operando sobre codificação BSD NAF consome, em média, 72% menos potência do que quando operando sobre dados sem codificação BSD NAF. Maiores detalhes sobre essa análise, bem como o comportamento do conversor BSD/BIN ao utilizar a codificação BSD NAF encontram-se no Apêndice C.

No entanto mesmo com esta redução, pode ser visto no gráfico da Figura 4.3 que os somadores BSDA1-3_BSD e BSDAPar_BSD são os somadores que apresentam o maior consumo de potência, sendo menores apenas que o RCAIOI, que é o somador que mais consome potência dentre todas as arquiteturas analisadas, devido à grande quantidade de lógica extra necessária para implementar a técnica RESI. Além disso, o BSDA_BSD consome praticamente o mesmo que o RCATMR apesar deste ser constituído da triplicação do RCA. Isso ocorre devido ao consumo de potência dos três

conversores utilizados pelo BSDA_BSD, que resulta em um aumento na atividade de chaveamento do somador e praticamente iguala o consumo deste ao consumo de potência d RCATMR.

Também pode ser visto na Figura 4.3 e através dos dados da Tabela 4.2 que o BSDA_C2 consome, em média, 53% mais potência que o RCA. Isso porque o BSDA_C2 além de possuir um RCA para implementar a etapa de conversão de saída possui a soma BSD propriamente dita. Já o BSDA_BSD consome, em média, 139% mais potência que o BSDA_C2.

Ao comparar os somadores BSDAs protegidos que utilizam operandos em BSD NAF com os somadores que utilizam o RCA pode ser visto que o BSDA1-3_BSD consome, em média, 46% mais potência que o RCATMR e o BSDAPar_BSD apresenta um consumo, em média, 29% maior que o consumo de potência do RCATMR. O RCAIOI consome, em média, 23% e 39% mais potência que o BSDA1-3_BSD e o BSDAPar_BSD, respectivamente.

Já os somadores BSDAs protegidos com entradas representadas em complemento de 2 apresentam um consumo de potência menor que os somadores protegidos que utilizam o RCA. Assim o RCATMR consome, em média, 27% mais potência que o BSDA1-3_C2 e, em média, 46% mais potência que o BSDAPar_C2.O RCAIOI consome uma quantidade de potência consideravelmente maior que os somadores BSDAs operandos representados em complemento de 2. Isso ocorre porque o somador RCAIOI além de utilizar os blocos necessarios para a implementação da técnica RESI, ainda utiliza o Somador Completo com *Carry* Redundante – SCCR o que contribui para o aumento da atividade de chaveamento gerando assim um maior consumo de potência, ao contrário dos somadores BSDA_C2 que utilizam um RCA normal para a etapa de conversão de saída. Desta forma o RCAIOI apresenta um consumo de potência, em média, 127% maior que o BSDA1-3_C2 e o RCAIOI apresenta um consumo, em média, 162% maior que o BSDAPar_C2.

Comparando as versões protegidas dos somadores BSDAs observa-se que o BSDA1-3_BSD consome, em média, 65% mais potência que o BSDA1-3_C2 e o BSDAPar_BSD consome, em média, 89% mais potência que o BSDAPar_C2.

As comparações apresentadas nesta sessão foram obtidas através de cálculos realizados a partir dos dados contidos na Tabela 4.2. Maiores explicações e mais análises sobre os resultados de atraso crítico, número de transistores e potência das arquiteturas completas podem ser encontradas no Apêndice C.

4.2 Análise das Arquiteturas sem a utilização de Conversores de Entrada e Saída

Para salientar a influencia da utilização dos conversores de entrada e saída dos somadores BSDAs nos quesitos avaliados, nesta seção são apresentados os resultados obtidos desconsiderando o uso de tais circuitos. Desta forma, os resultados analisados foram obtidos a partir dos circuitos que executam a soma propriamente dita, juntamente com os circuitos necessários para a aplicação das técnicas de proteção (no caso das arquiteturas protegidas).

A Tabela 4.3 apresenta os resultados de atraso crítico, número de transistores e consumo de potência para as arquiteturas que utilizam somadores do tipo RCA e BSDA, desconsiderando os conversores de entrada e saída das arquiteturas BSDAs. Os

gráficos que mostram o atraso crítico, número de transistores e potência sem as etapas de conversão são apresentados, respectivamente, nas Figura 4.4, Figura 4.5 e Figura 4.6.

Tabela 4.3: Resultados de Atraso Crítico, Número de Transistores e Potência das arquiteturas desconsiderando o uso dos conversores de entrada e saída nos somadores BSDAs

	Atraso (ns)				Número de Transistores				Potência Dinâmica (mW)			
Arquitetura	**4**	**8**	**16**	**32**	**4**	**8**	**16**	**32**	**4**	**8**	**16**	**32**
RCA	0,287	0,423	0,704	1,302	120	240	480	960	0,016	0,033	0,068	0,139
RCATMR	0,441	0,577	0,858	1,455	432	864	1728	3456	0,060	0,128	0,261	0,528
RCAIOI	0,326	0,477	0,786	1,407	798	1550	3054	6062	0,120	0,230	0,449	0,884
BSDA_BSD	0,199	0,199	0,199	0,199	424	848	1696	3392	0,008	0,021	0,049	0,105
BSDA1-3_BSD	0,225	0,225	0,225	0,225	655	1312	2624	5248	0,025	0,063	0,134	0,285
BSDAPAR_BSD	0,199	0,199	0,199	0,199	850	1786	3658	7420	0,029	0,065	0,138	0,285
BSDA_C2	0,149	0,149	0,149	0,149	96	192	384	768	0,020	0,040	0,078	0,157
BSDA1-3_C2	0,163	0,163	0,163	0,163	352	704	1408	2816	0,045	0,087	0,169	0,335
BSDAPAR_C2	0,149	0,149	0,149	0,149	264	552	1128	2280	0,037	0,076	0,153	0,308

O gráfico da Figura 4.4 e a Tabela 4.3 mostram que, sem o uso dos conversores de entrada e saída, o atraso dos circuitos que implementam a soma BSD nos somadores BSDAs realmente é constante, isto é, a soma BSD é livre da propagação do *carry*, conforme explicado no capítulo 3. Desta forma, desconsiderando o atraso introduzido pelos conversores, a soma BSD de todos os somadores BSDAs apresentam um atraso consideravelmente menor que os somadores que utilizam o RCA. A diferença entre os atrasos desses somadores é grande porque o atraso da soma BSD é constante independentemente do número de bits que o somador possuir, como mostra a Figura 4.4.

Assim o RCA passa a apresentar um atraso, em média, 241% e 355% maior que os somadores BSDA_BSD e BSDA_C2, respectivamente. As versões protegidas destes somadores apresentam o mesmo comportamento, isto é, os somadores RCAIOI e RCATMR apresentam um atraso crítico consideravelmente maior que os somadores BSDAs protegidos com codificação 1 de 3 e verificação de paridade. Logo o RCATMR é, em média, 270% e 318% mais lento que os somadores BSDA1-3_BSD e BSDAPar_BSD, respectivamente. Já o RCAIOI apresenta um atraso, em média, 383% maior que atraso do BSDA1-3_BSD e 445% maior que o atraso do BSDAPar_BSD (considerando que nenhuma falha transiente tenha ocorrido no RCAIOI). Vale ressaltar que o atraso do circuito Verificador de Erros do BSDA1-3_BSD e dos circuitos Indicador de Erros *Pw* e Preditor de Paridade do BSDAPar_BSD não exercem influencia sobre o atraso destes somadores dado que estes executam suas funções em paralelo com a execução da soma BSD.

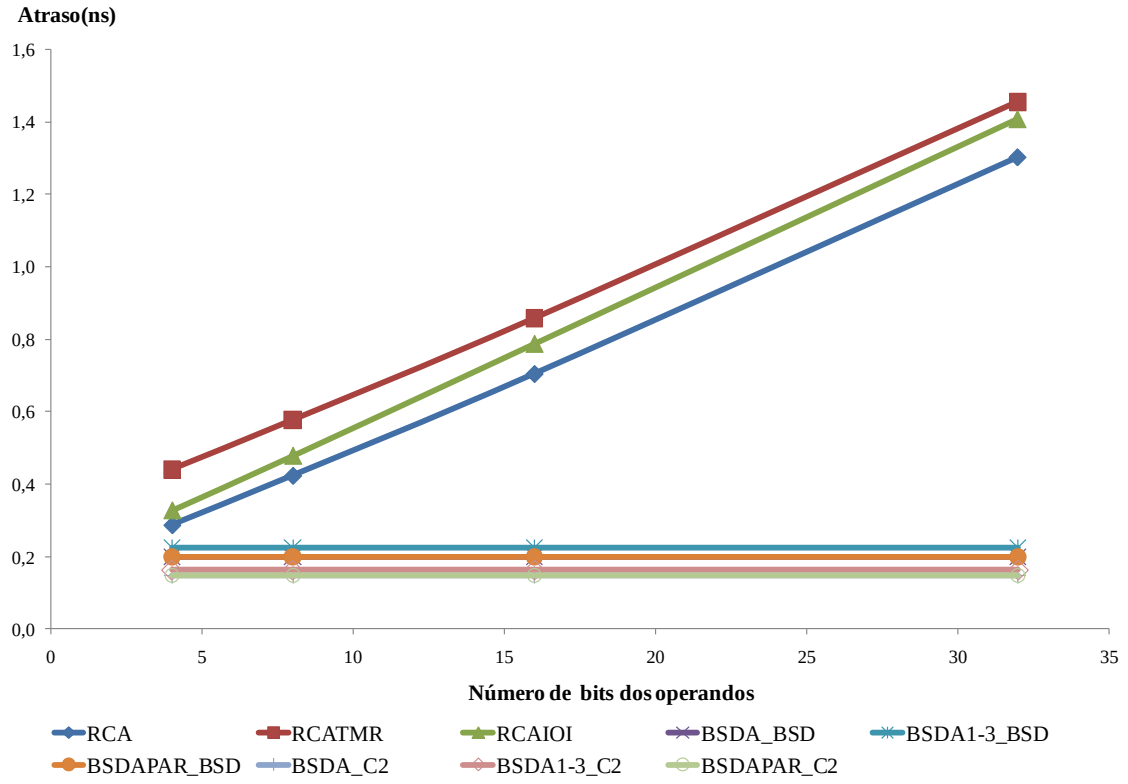


Figura 4.4: Atraso crítico das arquiteturas sem o uso de conversores

A Figura 4.4 mostra também que os somadores BSDAs com entradas representadas em complemento de 2 apresentam os menores atrasos críticos dentre todas as arquiteturas estudadas. Isso porque os operandos representados em complemento de 2 permitem uma maior simplificação nos circuitos que executam a soma BSD (circuitos ADD1 e ADD2 apresentados no capítulo 3). Desta forma, o RCATMR apresenta um atraso, em média, 410% maior que o BSDA1-3_C2 e 458% maior que o BSDAPar_C2. Já o RCAIOI é, em média, 359% mais lento que o BSDA1-3_C2 e 402% mais lento que o BSDAPar_C2.

Comparando as arquiteturas que tem atraso constante, pode ser visto através dos dados da Tabela 4.3 que o BSDA_BSD é, em média, 33% mais lento que o BSDA_C2. E considerando os somadores protegidos, o BSDA1-3_BSD é, em média, 38% mais lento que o BSDA1-3_C2 e o BSDAPar_BSD apresenta um atraso, em média, 33% maior que o BSDAPar_C2.

Quando o número de transistores utilizados pelas arquiteturas é analisado, pode ser visto na Tabela 4.3 e na Figura 4.5 que, mesmo desconsiderando as etapas de conversão dos somadores BSDAs, o BSDA_BSD necessita de mais transistores para ser projetado que o RCA. Desta forma, a soma BSD do BSDA_BSD é, em média, 253% maior que o RCA. Além disso, pode ser observado na Figura 4.5 que o BSDA_BSD utiliza praticamente a mesma quantidade de transistores que o RCATMR, mesmo sem os conversores, o que ressalta a grande quantidade de transistores utilizado pelos circuitos que implementam a soma BSD do BSDA_BSD já que o RCATMR triplica o RCA e ainda utiliza um circuito votador, ou seja, a técnica TMR necessita de uma quantidade considerável de transistores para ser projetada. No entanto, quando o BSDA_C2 é considerado ocorre contrário, ou seja, o RCA utiliza, em média, 25% mais transistores que a soma BSD do BSDA_C2, ressaltando neste caso a simplificação nos circuitos que

executam a soma BSD no BSDA_C2 permitida pelo uso de operandos representados em complemento de 2.

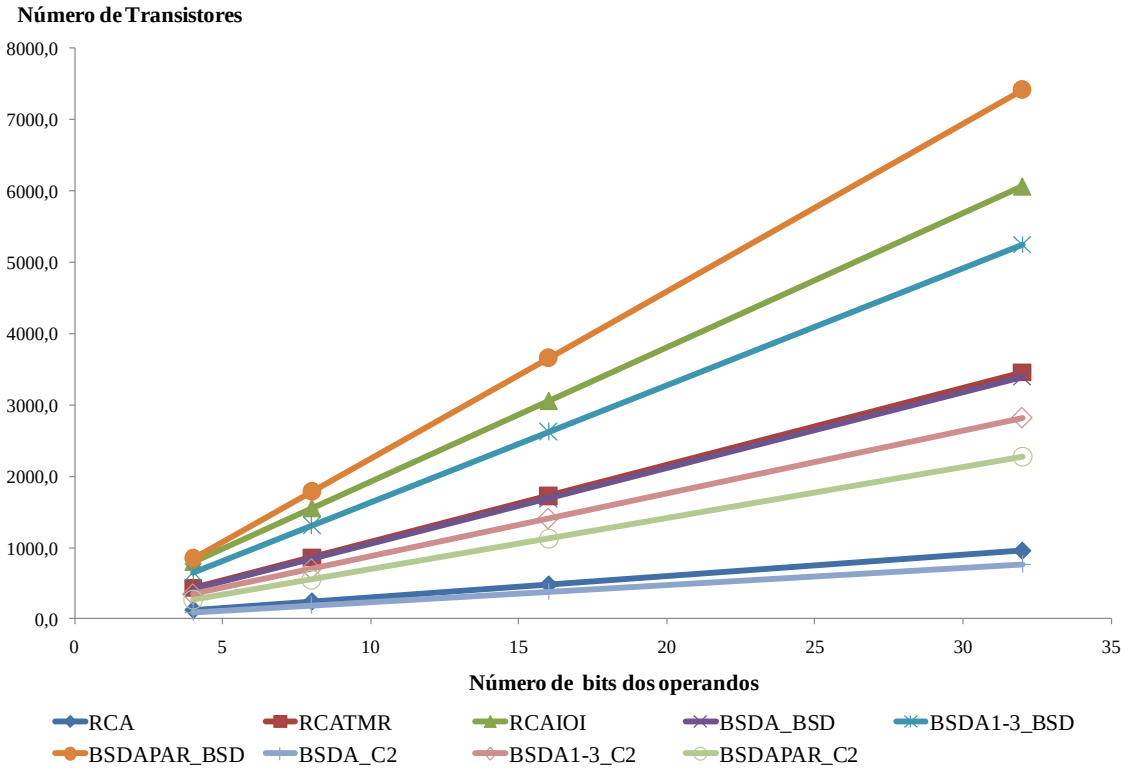


Figura 4.5: Números de transistores utilizados pelas arquiteturas sem o uso de conversores

Este comportamento se mantém quando a técnica de proteção TMR é aplicada ao RCA, ou seja, o RCATMR utiliza menos transistor que os somadores BSDA1-3_BSD e BSDAPar_BSD e utiliza mais transistores que os somadores BSDA1-3_C2 e BSDAPar_C2. Assim o BSDA1-3_BSD utiliza, em média, 52% mais transistores e o BSDAPar_BSD utiliza, em média, 107% mais transistores que o RCATMR. E o RCATMR utiliza, em média, 23% e 56% mais transistores que os somadores BSDA1-3_C2 e BSDAPar_C2, respectivamente. Já o RCAIOI usa, em média, 18% mais transistores que o BSDA1-3_BSD, sendo menor apenas que o BSDAPar_BSD que é, em média, 16% maior que o RCAIOI, como mostra a Figura 4.5, onde pode ser visto também que o BSDAPar_BSD utiliza mais transistores que todas as demais arquiteturas. Também pode ser visto que o RCAIOI utiliza mais transistores que os somadores BSDAs com entrada em complemento de 2, sendo, em média, 120% maior que o BSDA1-3_C2 e 180% maior que o BSDAPar_C2.

Considerando apenas as arquiteturas que executam a soma BSD, os somadores que possuem as entradas representadas em complemento de 2 utilizam uma quantidade consideravelmente menor de transistores que os somadores que possuem operandos representados em BSD NAF. Assim, o BSDA_BSD utiliza, em média, 342% mais transistores que o BSDA_C2, ao passo que o BSDA1-3_BSD necessita, em média, de 86% mais transistores que o BSDA1-3_C2 e o BSDAPar_C2 é, em média, 224% maior que o BSDAPar_C2.

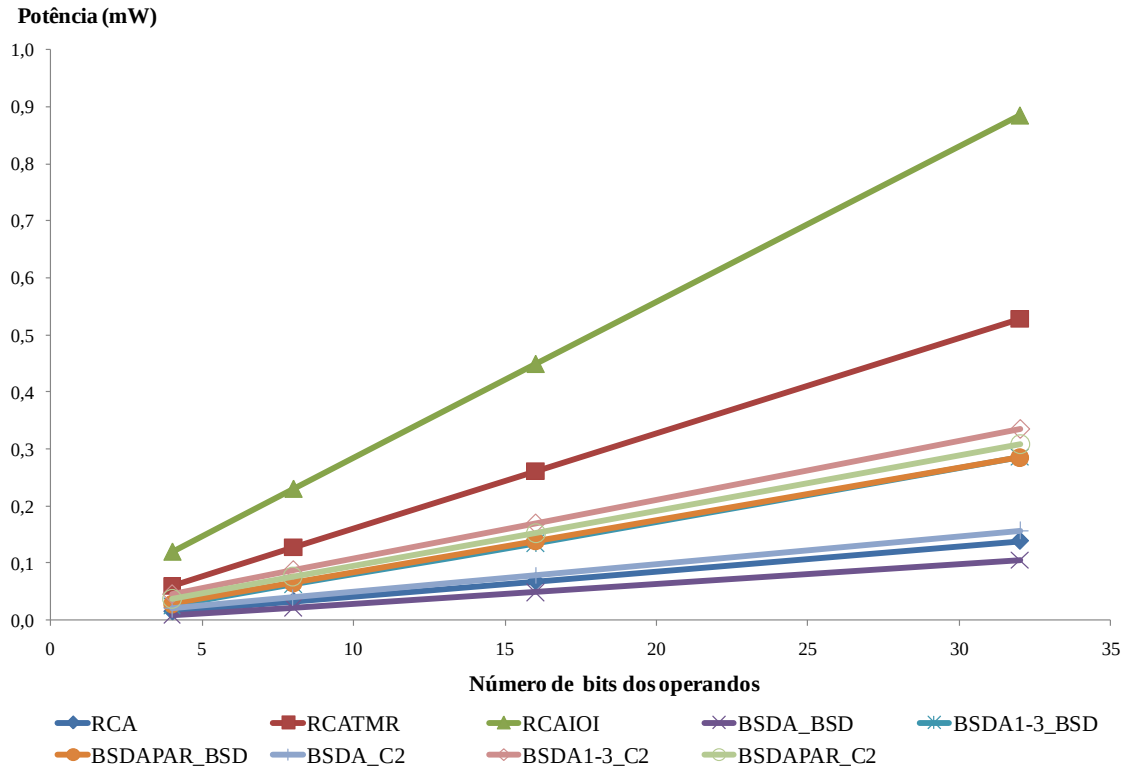


Figura 4.6: Potência consumida pelas arquiteturas sem o uso de conversores

A influência no consumo de potência, da utilização de operandos representados em BSD NAF, é vista nitidamente na Figura 4.6, onde nota-se que os somadores que utilizam a codificação BSD NAF, responsável por diminuir a atividade de chaveamento dos circuitos, apresentam o menor consumo de potência. Assim, dentre os somadores não-protegido o BSDA_BSD apresenta o menor consumo de potência. Desta forma, de acordo com os dados da Tabela 4.3, o RCA consome, em média, 53% mais potência que o BSDA_BSD e o BSDA_C2 apresenta um consumo de potência, em média, 83% maior que o consumo do BSDA_BSD (maiores detalhes podem ser encontrados no Apêndice C). Também pode ser visto no gráfico da Figura 4.6 que, ao desconsiderar as etapas de conversão, apesar do BSDA_C2 utilizar menos transistores, o que poderia resultar em um menor consumo de potência, este não utiliza a codificação BSD NAF, logo consome, em média, 19% mais potência que o RCA.

Dentre as arquiteturas protegidas também pode ser visto que somadores que utilizam a codificação BSD NAF consomem menos potência que os demais. Desta forma o BSDA1-3_BSD e o BSDAPar_BSD são os somadores protegidos que apresentam o menor consumo de potência. Assim, o BSDA1-3_C2 consome, em média, 41% mais potência que o BSDA1-3_BSDA e o BSDAPar_C2 apresenta um consumo, em média, 16% maior que o consumo de potência do BSDAPar_BSD.

A Figura 4.6 mostra que o RCA ao ser protegido pelas técnicas RESI e TMR consome mais potência dentre todos os somadores. O RCATMR consome, em média, 107% mais potência que o BSDA1-3_BSD e, em média, 95% mais potência que o BSDAPar_BSD. Já quando são considerados os somadores BSDAs com entradas representadas em complemento de 2 o RCATMR consome em média 48% e 68% mais potência que os somadores BSDA1-3_C2 e BSDAPar_C2. Já o RCAIOI consome, em média, 274% mais potência que o BSDA1-3_BSD e, em média, 252% mais potência

que o BSDAPar_BSD. Em relação aos somadores BSDAs com entradas em complemento de 2 o RCAIOI consome, em média, 165% e 220% mais potência que os somadores BSDA1-3_C2 e BSDAPar_C2.

As comparações apresentadas nesta sessão foram obtidas através de cálculos realizados a partir dos dados contidos na Tabela 4.3. Maiores explicações e mais análises sobre os resultados de atraso crítico, número de transistores e potência desconsiderando as etapas de conversão das arquiteturas BSDAs podem ser encontradas no Apêndice C.

4.3 Análise da Tolerância a Falhas dos Somadores

A fim de avaliar a eficiência das técnicas de proteção aplicadas aos somadores RCA e BSDA, campanhas de injeção de falhas foram realizadas, para todos os somadores projetados, por meio de simulações no nível elétrico usando o simulador Spice. (HSPICE).

A injeção de falhas e o método de simulação consistem em injetar na lógica combinacional uma falha simples por vez e analisar sua propagação usando simulação no nível elétrico simulando a falha. Uma vez que a propagação é função da lógica do circuito, a simulação deve levar em consideração cada possível vetor de entrada, a fim de determinar se o SET atinge uma das saídas primárias do circuito. A falha transiente é modelada no simulador elétrico através de uma fonte de corrente, usando a dupla exponencial de Messenger (1982). Para usar tal modelagem, um algoritmo de injeção de falhas pode ser usado, conforme ilustrado na Figura 4.7.

```
1   para cada falha
2     para cada vetor de entrada
3       simular o circuito com falha
4       comparar resultado c/ o resultado do circuito s/ falha
5       se (falha propagada)
6         incrementa lista de falhas
7     novo vetor de entrada
8   nova falha
```

Figura 4.7: Algoritmo utilizado para a Injeção de Falhas

Para tornar possível a simulação de um número significativo de falhas para todas as arquiteturas de RCAs e BSDAs analisadas, foi necessário desenvolver um programa em C++ para automatizar o controle do processo de injeção e simulação de falhas. Ao final de cada simulação executada, um arquivo de resultados é gerado com as falhas que se propagaram até as saídas primárias e dentre estas, as falhas que foram detectadas pela técnica de proteção.

A Tabela 4.4 apresenta os resultados obtidos através da injeção de falhas. As falhas foram injetadas somente para somadores de 4 bits.

É importante notar que o número de falhas injetadas nas versões protegidas do RCA, do BSDA_BSD e do BSDA_C2 é maior que o número de falhas injetadas nas versões não-protegidas devido ao acréscimo do número de transistores requeridos para realizar a proteção. Além disso, a Tabela 4.4 mostra que o RCATMR teve falhas injetadas apenas

no circuito votador, pois este é considerado o ponto único de falhas (*single-point of failure*) da técnica TMR, sendo que 30,55% das falhas injetadas no votador propagam-se até as saídas primárias do mesmo.

Tabela 4.4: Resultados da Injeção de Falhas

Arquiteturas	**Falhas injetadas (1)**	**Falhas propagadas (2)**	**Falhas detectadas (3)**	**(2)/(1)*100 [%]**	**((2)-(3))/1*100 [%]**	**((2)-(3))/2*100 [%]**
RCA	152	41	-	26,97	-	-
RCATMR	72	22	-	30,56	-	-
RCAIOI	602	89	52	14,78	6,15	41,57
BSDA_BSD	824	286	-	34,71	-	-
BSDA1-3_BSD	892	224	195	25,11	3,25	12,95
BSDAPar_BSD	1410	350	350	24,82	0,00	0,00
BSDA_C2	312	118	-	37,82	-	-
BSDA1-3_C2	568	202	202	35,56	0,00	0,00
BSDAPar_C2	580	181	41	31,21	24,14	77,35

Também pode-se observar que o RCAIOI e o BSDAPar_BSD apresentam a menor quantidade de falhas propagadas até as saídas. No RCAIOI, em média, 14,78% das falhas injetadas propagam-se até suas saídas, sendo que 41,57% destas não são identificadas. No caso do BSDAPar_BSD 24,82% das falhas injetadas propagam-se até as saídas deste, porém o BSDAPar_BSD detecta todas as falhas que se propagaram até as saídas.

Já a maior quantidade de falhas propagadas dentre os somadores protegidos é apresentada pelo BSDA1-3_C2, onde 35,56% das falhas injetadas propagam-se até as saídas do mesmo. No entanto a Tabela 4.4 mostra que o BSDA1-3_C2 identifica todas as falhas que se propagaram até as suas saídas.

O pior grau de proteção é apresentado pelo BSDAPar_C2, onde das 31,21% falhas que propagam-se até suas saídas, 77,35% não são detectadas.

5 CONCLUSÕES E TRABALHOS FUTUROS

Este trabalho avaliou o comportamento de somadores do tipo *Ripple Carry Adder* (RCA) e *Binary Signed Digit Adder* (BSDA) não-protegidos e protegidos contra falhas transientes. Duas versões de RCA protegido foram analisadas: uma com TMR (*Triple Modular Redundancy*) e outra com RESI (Recomputação com Entradas e Saídas Invertidas). As versões protegidas de BSDA utilizaram redundância de informação, por meio de código 1 de 3 ou de verificação da paridade. Resultados de número de transistores foram obtidos por meio do mapeamento manual dos somadores para uma biblioteca de células contendo portas lógicas CMOS. Resultados de atraso crítico e consumo médio de potência foram obtidos através de simulações no nível elétrico utilizando o simulador HSpice, e considerando o modelo PTM de 45nm. Também foram realizadas campanhas de injeção de falhas, por meio de simulação no nível elétrico. Os resultados permitiram estimar o grau de proteção das arquiteturas de somadores consideradas.

Baseado nos resultados obtidos conclui-se que as arquiteturas dos somadores BSDAs com entradas representadas em BSD NAF obtiveram resultados inferiores aos resultados obtidos pelas arquiteturas dos somadores BSDAs com entradas representadas em complemento de 2 e pelos somadores baseados no RCA. No entanto deve-se ressaltar a importância destas arquiteturas para a comprovação e quantificação da redução de potência gerada pela codificação BSD NAF, que pode ser de grande interesse para aplicações que exijam um consumo reduzido.

Desta forma, os resultados onde os conversores utilizados pelos somadores BSDAs com entradas representadas em BSD NAF foram desconsiderados mostraram que a codificação BSD NAF reduz o consumo de potência do circuito que executa a soma BSD do BSDA_BSD em 72% ao passo que a soma BSD do BSDA1-3_BSD apresenta uma redução de 75% na potência consumida. Já o conversor BSD/BIN utilizado por ambos, consome 59% menos potência quando operando sobre dados codificados em BSD NAF.

Com estes resultados conclui-se que o uso de codificação BSD NAF pode ser mais eficiente do que o sistema binário tradicional se o SOC precisar executar muitas adições (sobretudo com operandos de muitos bits), a título de cálculos intermediários, de modo que apenas alguns poucos valores precisem ser enviados para a saída. Tal tipo de comportamento pode ser encontrado em multiplicadores em geral, bem como em algoritmos de DSP (*Digital Signal Processing*).

Os somadores BSDAs com entradas representadas em complemento de 2 obtiveram resultados semelhantes ao RCATMR, pois apesar de não utilizarem conversores de entrada como os somadores BSDAs com entradas representadas em BSD NAF, eles necessitam de um conversor de saída, que equivale praticamente a um RCA. Assim os

somadores BSDA1-3_C2 e BSDAPar_C2 apresentam um atraso crítico semelhante ao atraso crítico do RCATMR e do RCAIOI. No entanto, no caso do RCAIOI, é importante ressaltar que o atraso apresentado não leva em consideração a ocorrência de uma falha, caso em que o RCAIOI deverá executar novamente a soma.

Em relação à quantidade de transistores utilizados por estes somadores, o BSDAPar_C2 utiliza menos transistor que o RCATMR, sendo que o BSDA1-3_C2 utiliza uma quantidade maior de transistores que o RCATMR, no entanto esta diferença não chega a ser muito representativa. Apesar dos somadores BSDAPar_C2 e BSDA1-3_C2 utilizarem praticamente a mesma quantidade de transistores que RCATMR, o consumo de potência destes somadores foi consideravelmente inferior ao consumo de potência do RCATMR(e também do consumo de potência do RCAIOI). Além disto, o grau de proteção dos somadores BSDA1-3_C2 e BSDAPar_C2 é no pior caso igual ao grau de poteção do RCATMR. Desta forma, conclui-se que os somadores BSDA1-3_C2 e BSDAPar_C2 podem ser uma alternativa ao RCATMR quanto a aplicação necessitar de um consumo de potência reduzido.

5.1 Trabalhos Futuros

Durante a realização deste trabalho, observou-se que algumas melhorias e experimentos poderiam ser realizados para otimizar os resultados obtidos. Tais melhorias e experimentos são listados a seguir:

- implementação de sistemas que necessitem executar muitas adições, como os multiplicadores e algoritmos de DSP, onde as perdas com as etapas de conversão dos somadores BSDAs poderão tornar-se insignificantes diante do ganho em redução de potência e melhoria do atraso crítico;
- implementar os somadores apresentados utilizando portas complexas;
- retirar a lógica compartilhada dos somadores BSDA_BSD, BSDA1-3_BSD e BSDAPar_BSD a fim de otimizar a tolerância a falha dos mesmos.

REFERÊNCIAS

ABRAMOVICI, M.; BREUER, M.; FRIEDMAN, A. **Digital Systems Testing and Testable Design.** [S.l.]: IEEE Press, 1990. 652p.

ALEXANDRESCU, D.; ANGHEL, L.; NICOLAIDIS M. New Methods for Evaluating the Impact of Single-Event Transients in VDSM ICs. In: IEEE INTERNATIONAL SYMPOSIUM ON DEFECT AND FAULT TOLERANCE IN VLSI SYSTEMS, 17. Vancouver, Canada, 2002, pp. 99-107.

ALEXANDRESCU, D.; ANGHEL, L.; NICOLAIDIS M. Simulating Single Event Transients in VDSM ICs for Ground Level Radiation. **Journal of Electronic Testing: theory and applications**, v. 20, p.413-421, 2004.

ANGHEL, L.; ALEXANDRESCU, D.; NICOLAIDIS, M. Evaluation of a Soft Error Tolerance Technique Based on Time and/or Space Redundancy. In: SYMPOSIUM ON INTEGRATED CIRCUITS AND SYSTEMS DESIGN, SBCCI, 13., 2000. **Anais…** Los Alamitos : IEEE Computer Society, 2000. p. 237-242.

ANGHEL, L.; LEVEUGLE, R.; VANHAUWAERT, P. Evaluation of SET and SEU Effects at Multiple Abstraction Levels. In: IEEE INTERNATIONAL ON-LINE TESTING SYMPOSIUM, 11 (IOLTS'05). Saint Raphaël (França), Jul 6-8, 2005. **Anais…** Los Alamitos (California), IEEE Computer Society, 2005. p. 309-312.

ASSIS, T. R. de. **Analysis of Transistor Sizing and Folding Effectiveness to Mitigate Soft Errors**. 2009. Dissertação (Mestrado em Ciência da Computação) — UFRGS.

BARTH, J. Applying Computer Simulation Tools to Radiation Effects Problems. In: IEEE NUCLEAR SPACE RADIATION EFFECTS CONFERENCE, NSREC, 1997. **Anais. . .** [S.l.: s.n.], 1997.

BASTOS, R. P. **Design of a Soft-Error Robust Microprocessor.** 2006. Dissertação (Mestrado em Ciência da Computação) — UFRGS.

BAUMANN, R. C. Soft Errors in Advanced Semiconductor Devices - Parte I: The Three Radiation Sources. **IEEE Transactions on Device and Materials Reliability**, v. 1, n. 1, Mar 2001.

BAUMANN, R. The Impact of Technology Scaling on Soft Error Rate Performance and Limits to the Efficacy of Error Correction. In: INTERNATIONAL ELECTRON DEVICES MEETING, 2002. **Anais. . .** [S.l.: s.n.], 2002. p.329–332.

BAUMANN, R. C. Radiation-Induced Soft Errors in Advanced Semiconductor Technologies. **IEEE Transactions on Devices and Materials Reliability**, New York, v.5, n.3, p.305-316, Set, 2005.

BAZE, M.; BUCHNER, S. Attenuation of Single Event Induced Pulses in CMOS Combinational Logic. **IEEE Transactions on Nuclear Science**, Vol. 44, No. 6, December 1997.

BUSHNELL, M. L.; AGRAWAL, V. D. **Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits.** [S.l.]: Boston, EUA : Kluwer, 2000.

CARDARILLI, G.; OTTAVI, M.; PONTARELLI, S.; RE, M.; SALSANO, A. Error Detection in Signed Digit Arithmetic Circuit with Parity Checker. **Defect and Fault-Tolerance in VLSI Systems, IEEE International Symposium on**, Los Alamitos, CA, USA, v.0, p.401–408, 2003.

CHA, H.; RUDNICK, E.; PATEL, J.; IYER, R.; CHOI, G. A gate-level simulation environment for alpha-particle-induced transient faults. **Computers, IEEE Transactions on,** [S.l.], v.45, n.11, p.1248 –1256, nov. 1996.

CHRISTOPHE, E.; DUHAMEL, P.; MAILHES, C. Adaptation of zerotrees using signed binary digit representations for 3D image coding. **J. Image Video Process.**, New York, NY, USA, v.2007, n.1, p.7, 2007.

COHEN, N.; SRIRAM, T.; LELAND, N.; MOYER, D.; BUTLER, S.; FLATLEY, R. Soft Error Considerations for Deep-Submicron CMOS Circuit Applications. **INTERNATIONAL ELECTRON DEVICES MEETING**, [S.l.], p.315–318, 1999.

DAHLGREN, P.; LIDEN, P. A Switch-Level Algorithm for Simulation of Transients in Combinational Logic. **Fault-Tolerant Computing, International Symposium on**, Los Alamitos, CA, USA, v.0, p.0207, 1995.

DODD, P.; MASSENGILL, L. Basic mechanisms and modeling of single-event upset in digital microelectronics. **Nuclear Science, IEEE Transactions on,** [S.l.], v.50, n.3, p.583 – 602, jun. 2003.

DUPONT, E.; NICOLAIDIS, M.; ROHR, P. Embedded Robustness IPs for Transient-Error-Free ICs. **IEEE Des. Test,** Los Alamitos, CA, USA, v.19, n.3, p.56–70, 2002.

EBEID, N.; HASAN, M. A. On binary signed digit representations of integers. **Des. Codes Cryptography,** Norwell, MA, USA, v.42, n.1, p.43–65, 2007.

FECHNER, B. Analysis of checksum-based execution schemes for pipelined processors. In: 2006. **Anais. . .** [S.l.: s.n.], 2006. p.8 pp.

FECHNER, B.; KELLER, J. Compression-free Checksum-based Fault-Detection Shemes for Pipelined Processors. In: ARCS '07 - 20th International Conference on Architecture of Computing Systems, 2007, Zurich, Switzerland. **Proceedings…** ARCS '07

FRANCO, D. T. **Signal Reliability of Combinational Logic Circuits Under Multiple Simultaneous Faults.** Tese (Doutorado em Ciência da Computação) — ENST-Paris.

GAJSKI, D. D. **Principles of digital design.** Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1996.

GILL, B.; PAPACHRISTOU, C.; WOLFF, F.; SEIFERT, N. Node sensitivity analysis for soft errors in CMOS logic. In: TEST CONFERENCE, 2005. PROCEEDINGS. ITC 2005. IEEE INTERNATIONAL, 2005. **Anais. . .** [S.l.: s.n.], 2005. p.9 pp. –972.

GOLDSTEIN, L. Controllability/observability analysis of digital circuits. **Circuits and Systems, IEEE Transactions on,** [S.l.], v.26, n.9, p.685 – 693, sep. 1979.

GORDON, D. M. A Survey of Fast Exponentiation Methods. **Journal of Algorithms,** [S.l.], v.27, n.1, p.129–146, 1998.

HARATA, Y.; NAKAMURA, Y.; NAGASE, H.; TAKIGAWA, M.; TAKAGI, N. A highspeed multiplier using a redundant binary adder tree. **Solid-State Circuits, IEEE Journal of,** [S.l.], v.22, n.1, p.28–34, feb 1987.

HEIJMEN, T.; NIEUWLAND, A. Soft-Error Rate Testing of Deep-Submicron Integrated Circuits. In: TEST SYMPOSIUM, 2006. ETS '06. ELEVENTH IEEE EUROPEAN, 2006. **Anais. . .** [S.l.: s.n.], 2006. p.247 –252.

HOUGHTON, A. D. **The Engineer's Error Coding Handbook.** London: Chapman & Hall, 1997.

HUANG, C.-T.; GOUDA, M. G. State Checksum and Its Role in System Stabilization. In: ICDCSW '05: PROCEEDINGS OF THE FOURTH INTERNATIONAL WORKSHOP ON ASSURANCE IN DISTRIBUTED SYSTEMS AND NETWORKS (ADSN) (ICDCSW'05), 2005, Washington, DC, USA. **Anais. . .** IEEE Computer Society, 2005. p.29–34.

HWANG, K. **Computer Arithmetic: principles, architecture and design.** New York, NY, USA: John Wiley & Sons, Inc., 1979.

IROM, F.; SWIFT, G.; FARMANESH, F.; JOHNSTON, A. Single-event upset in commercial silicon-on-insulator PowerPC microprocessors. In: IEEE COMPUTER SOCIETY, 2002. **Anais. . .** [S.l.: s.n.], 2002. p.203 – 204.

JOHNSTON, A. Scaling and Technology Issues for Soft Error Rates. In: RESEARCH CONFERENCE ON RELIABILITY, 2000. **Anais. . .** [S.l.: s.n.], 2000. v.4.

JOYE, M.; YEN, S.-M. Optimal left-to-right binary signed-digit recoding**. Computers, IEEE Transactions on,** [S.l.], v.49, n.7, p.740–748, jul 2000.

KASTENSMIDT, F. G. d. L. **Designing single event upset mitigation techniques for large SRAM-Based FPGA components.** 2003. Tese (Doutorado em Ciência da Computação) — UFRGS.

KASTENSMIDT, F. L.; NEUBERGER, G.; CARRO, L.; REIS, R. Designing and testing fault-tolerant techniques for SRAM-based FPGAs. In: COMPUTING

FRONTIERS, 1., 2004, New York, NY, USA. **Anais. . .** ACM, 2004. p.419–432. (CF ’04).

KEATING, M.; FLYNN, D.; AITKEN, R.; GIBBONS, A.; SHI, K. **Low Power Methodology Manual for System-on-Chip Design**. New York, NY: Springer, 2007.

KIM, N. S. et al. Leakage current: Moore’s law meets static power. **IEEE Computer**, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 36, p. 68–75, December 2003.

KOREN, I. **Computer Arithmetic Algorithms,** Second Edition, A. K. Peters, Natick, MA, 2002.

KOREN, I; KRISHNA, C. M. **Fault-Tolerant Systems,** Morgan-Kaufman, San Francisco, CA, 2007.

KUNINOBU, S.; NISHIYAMA, T.; EDAMATSU, H.; TANIGUCHI., T.; TAKAGI, N. Design of high speed MOS multiplier and divider using redundant binary representation. In: SYMP. COMPUTER ARITHMETIC, 8., 1987. **Anais. . .** [S.l.: s.n.], 1987. p.80–86.

LALA, P. K. **Self-Checking and Fault-Tolerant Digital Design.** [S.l.: s.n.], 2001.

LALA, P. K.; WALKER, A. On-Line Error Detectable Carry-Free Adder Design. In: DFT ’01: PROCEEDINGS OF THE IEEE INTERNATIONAL SYMPOSIUM ON EFECT AND FAULT TOLERANCE IN VLSI SYSTEMS, 2001, Washington, DC, USA. **Anais. . .** IEEE Computer Society, 2001. p.66.

LEE, H. K.; HA, D. S.; KIM, K. Test generation of stuck-open faults using stuck-at test sets in CMOS combinational circuits. In: ACM/IEEE DESIGN AUTOMATION CONFERENCE, 26., 1989, New York, NY, USA. **Proceedings. . .** ACM, 1989. p.345–350. (DAC ’89).

LISBOA, C. A. L**. Dealing with radiation induced long duration transient faults in future technologies. 2009.** Tese (Doutorado em Ciência da Computação) — UFRGS.

LO, J.-C.; THANAWASTIEN, S.; RAO, T. Concurrent error detection in arithmetic and logical operations using Berger codes. In: IEEE COMPUTER SOCIETY, 1989. **Anais. . .** [S.l.: s.n.], 1989. p.233 –240.

MAVIS, D.; EATON, P. Soft error rate mitigation techniques for modern microcircuits. In: INTERNATIONAL RELIABILITY PHYSICS SYMPOSIUM, 2002. **Anais. . .** [S.l.: s.n.], 2002. p.216 – 225.

MESSENGER, G. C. Collection of Charge on Junction Nodes from Ion Tracks. **IEEE Transactions on Nuclear Science,** [S.l.], v.29, p.2024–2031, 1982.

NANGATE. Nangate 45nm Open Cell Library. < http://www.nangate.com/ > Acesso em 08/01/2011.

NAJM, F. N. A survey of power estimation techniques in VLSI circuits. **IEEE Trans. Very Large Scale Integr. Syst.,** Piscataway, NJ, USA, v.2, n.4, p.446–455, 1994.

NEUMANN, J. V. Probabilistic logics and the synthesis of reliable organisms from unreliable components. Automata Studies, Ann. of Math. Studies, (34):43–98, 1956.

NICOLAIDIS, M. Design for soft error mitigation. **Device and Materials Reliability, IEEE Transactions on,** [S.l.], v.5, n.3, p.405 – 418, sep. 2005.

NICOLAIDIS, M. Carry Checking/Parity Prediction Adders and ALUs. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, New York, v.11, n.1, p. 121-128, Feb. 2003.

NICOLAIDIS, M.; DUARTE, R.; MANICH, S.; FIGUERAS, J. Fault-secure parity prediction arithmetic operators. **Design Test of Computers, IEEE,** [S.l.], v.14, n.2, p.60 –71, apr. 1997.

NORMAND, E. Single event upset at ground level. **Nuclear Science, IEEE Transactions on,** [S.l.], v.43, n.6, p.2742 –2750, dec. 1996.

NORMAND, E.; BAKER, T. Altitude and latitude variations in avionics SEU and atmospheric neutron flux. **Nuclear Science, IEEE Transactions on,** [S.l.], v.40, n.6, p.1484 –1490, dec. 1993.

OIKONOMAKOS, P.; FOX, P. Error Correction in Arithmetic Operations by I/O Inversion. In: IOLTS ’06: PROCEEDINGS OF THE 12TH IEEE INTERNATIONAL SYMPOSIUM ON ON-LINE TESTING, 2006, Washington, DC, USA. **Anais. . .** IEEE Computer Society, 2006. p.287–292.

OKEYA, K.; SCHMIDT-SAMOA, K.; SPAHN, C.; TAKAGI, T. Signed Binary Representations Revisited. In: FRANKLIN, M. (Ed.). **Advances in Cryptology – CRYPTO 2004.** [S.l.]: Springer Berlin / Heidelberg, 2004. p.119–142. (Lecture Notes in Computer Science, v.3152).

OMAÑA, M.; PAPASSO, G.; ROSSI, D.; METRA, C. A model for transient fault propagation in combinatorial logic. **On-Line Testing Symposium, 2003. IOLTS 2003. 9th IEEE,** [S.l.], p.111 – 115, jul. 2003.

ORAILOGLU, A.; KARRI, R. Automatic synthesis of self-recovering vlsi systems. **IEEE Transactions on Computers**, 45(2):131–142, February 1996.

OSSI, E. J.; LIMBRICK, D. B.; ROBINSON,W. H.; BHUVA, B. L. Soft-error mitigation at the architecture-level using Berger codes and instruction repetition. In: IEEE WORKSHOP ON SILICON ERRORS IN LOGIC-SYSTEM EFFECTS (SELSE 2009), 2009. Anais. . . [S.l.: s.n.], 2009.

PARHAMI, B. **Computer arithmetic:** algorithms and hardware designs. Oxford, UK: Oxford University Press, 2000.

PARHAMI, B. Approach to the design of parity-checked arithmetic circuits. In: SIGNALS, SYSTEMS AND COMPUTERS, 2002. CONFERENCE RECORD OF THE THIRTY-SIXTH ASILOMAR CONFERENCE ON, 2002. **Anais. . .** [S.l.: s.n.], 2002. v.2, p.1084 – 1088 vol.2.

PETERSON, W.; BROWN, D. Cyclic Codes for Error Detection. **Proceedings of the IRE,** [S.l.], v.49, n.1, p.228 –235, jan. 1961.

PETERSON, W. Wesley. **Error-correcting codes.** 2nd. Edition. Cambridge, EUA: The Mit Press, 1980. 560p.

PRADHAN, Dhiraj K.. **An Introduction To Desigin and Analysis of Fault-Tolerant Systems**. In: Faut-Tolerant Computer System Design. New Jersey: Prentice Hall PTR, 1996. p. 1-84.

RABAEY, J; CHANDRAKASAN, A.; NIKOLIC, B. **Digital Integrated Circuits:** a design perspective. 2nd Edition. Upper Saddle River, N.J.: Prentice Hall, 2003.

RABAEY, J. **Low Power Design Essentials**. New York, NY: Springer, 2009.

RAO, T. R. N. **Error Coding for Arithmetic Processors.** Orlando, FL, USA: Academic Press, Inc., 1974.

SHIVAKUMAR, P.; KISTLER, M.; KECKLER, S.; BURGER, D.; ALVISI, L. Modeling the effect of technology trends on the soft error rate of combinational logic. In: DEPENDABLE SYSTEMS AND NETWORKS, 2002. DSN 2002. PROCEEDINGS. INTERNATIONAL CONFERENCE ON, 2002. **Anais. . .** [S.l.: s.n.], 2002. p.389 – 398.

SYNOPSYS. The Gold Standard for Accurate Circuit Simulation. HSPICE Homepage. <http://www.synopsys.com/Tools/Verification/AMSVerification/CircuitSimulation/HSPICE/Pages/default.aspx > Acesso em 07/01/2011

TAKAGI, N.; YASUURA, H.; YAJIMA, S. High-Speed VLSI Multiplication Algorithm with a Redundant Binary Addition Tree. **Computers, IEEE Transactions on,** [S.l.], v.C- 34, n.9, p.789 –796, sept. 1985.

TAKAHASHI, Y.; KONTA, K. ichi; CMOS, D.; TECHNOLOGY, L. C.; KAF, T.; KARR, T.; SHOUNO, K.; PROPAVVZQH, C. Carry Propagation Free Adder/Subtracter Using Adiabatic Dynamic CMOS Logic Circuit Technology. **IEICE Trans. Fundamentals,** [S.l.], v.E86-A, n.6, p.437–1444, junho 2003.

THORNTON, M. A. Signed Binary Addition Circuitry with Inherent Even Parity Outputs. **IEEE Trans. Comput.,** Washington, DC, USA, v.46, n.7, p.811–816, 1997.

TOWNSEND, W. J.; ABRAHAM, J. A.; LALA, P. K. On-Line Error Detecting Constant Delay Adder. In: IN 9TH IEEE INTL. ON-LINE TESTING SYMP, 2003. **Anais. . .** [S.l.: s.n.], 2003.

TOWNSEND, W. J.; THORNTON, M. A.; LALA, P. K. On-line error detection in a carry-free adder. In: IN 11TH IEEE/ACM INTERNATIONALWORKSHOP ON LOGIC & SYNTHESIS, 2002. **Anais. . .** [S.l.: s.n.], 2002. p.251–254.

VIOLANTE, M. Accurate single-event-transient analysis via zero-delay logic simulation. **Nuclear Science, IEEE Transactions on,** [S.l.], v.50, n.6, p.2113 – 2118, dec. 2003.

WIRTH, G.; VIEIRA, M.; NETO, E.; KASTENSMIDT, F. Single Event Transients in Combinatorial Circuits. In: SYMPOSIUM ON INTEGRATED CIRCUITS AND SYSTEMS DESIGN, 18., 2005. **Anais. . .** [S.l.: s.n.], 2005. p.121 –126.

YEN, S.-M.; LAIH, C.-S.; CHEN, C.-H.; LEE, J.-Y. An efficient redundant-binary number to binary number converter. **Solid-State Circuits, IEEE Journal of**, [S.l.], v.27, n.1, p.109 –112, jan 1992.

ZHAO, W.; CAO, Y. New generation of Predictive Technology Model for sub-45nm early design exploration **IEEE Transactions on Electron Devices**, vol. 53, no. 11, pp. 2816-2823, November 2006.

APÊNDICE A SISTÊMAS NUMÉRICOS CONVENCIONAIS

O sistema de representação numérica posicional foi usado pela primeira vez pelos chineses. Neste sistema o valor representado por cada símbolo não depende somente de sua forma, mas também da posição de tal símbolo em relação aos demais símbolos, ou seja, cada símbolo possui um peso em relação aos demais símbolos que compõem o número.

A equação 1 mostra como é obtido um número *X* no sistema numérico posicional

$$X = \sum_{i=0}^{n-1} x_i \cdot w_i \tag{1}$$

onde *n* representa a quantidade de símbolos e w_i o peso do dígito x_i.

O **sistema decimal** usado nos dias de hoje para representar números é baseado no sistema posicional, e surgiu por volta do século 9. Neste sistema, a posição do símbolo determina seu valor, ou seja, um mesmo símbolo possui diferentes valores de acordo com a posição em que se encontra em um número. Isto é realizado através da multiplicação do símbolo de uma determinada posição pelo seu peso, w_i, que em sistemas numéricos convencionais é determinado pela i-ésima potência da base utilizada. Por exemplo, considerando o sistema decimal, no número 555 cada símbolo 5 representa um valor diferente: o 5 mais à esquerda representa 5 centenas, ao passo que os símbolo do meio e da direita representam respectivamente, 5 dezenas e 5 unidades, ou seja, $5*10^2 + 5*10^1 + 5*10^0 = (555)_{10}$. Outro exemplo é o número 6446, onde o valor do primeiro símbolo 6 é diferente do último símbolo 6. O primeiro indica 6 mil e o último indica 6 unidades. Isso também serve para os símbolos 4. O primeiro indica 4 centenas enquanto que o segundo indica 4 dezenas. Mais informações a respeito deste assunto podem ser encontradas em (PARHAMI, 2000).

Já quando o sistema numérico utiliza números romanos, isso não é válido. Considerando, por exemplo, o número XXVIII, cada X vale 10, independente de sua posição, sendo que o mesmo acontece com o V e I, ou seja, os símbolos que compõem um número não são multiplicados por um peso.

Por exemplo, VI $(5 + 1)_{10} = (6)_{10}$ e IV $(5 – 1) = 4$ demonstram que neste sistema, a posição dos símbolos apenas indica se o símbolo inferior deve ser somado (se estiver à direita), ou subtraído (se estiver à esquerda) do símbolo superior.

Em um sistema posicional, se a unidade correspondente para cada posição é um múltiplo constante da unidade para a posição à direita de seus vizinhos obtém-se o sistema convencional base fixa posicional.

O sistema numérico base fixa posicional possui uma base *r* inteira e positiva, e representa cada número sem sinal por um vetor de dígitos que poderão assumir *r* diferentes valores do conjunto de dígitos {0, 1, ..., *r-1*}.

Tal vetor de dígitos possui o tamanho de *k* + *l*, onde *k* é a quantidade de dígitos da parte inteira e *l* representa a quantidade de dígitos da parte fracionária do número. O valor do número $X = (x_{k-1}x_{k-2}...x_1x_2 \ . \ x_{-1}x_{-2}...x_{-l})_r$ é representado pela equação 2

$$X = \sum_{i=-l}^{k-1} x_i \cdot r^i \qquad (2)$$

O **sistema decimal** é um exemplo de sistema base fixa posicional com base fixa *r* = 10. Mais informações a respeito deste assunto podem ser encontradas em (PARHAMI, 2000) (KOREN, 2002) (HWANG, 1979).

Em um sistema posicional base mista a base é um vetor composto de mais de um valor, ou seja, a base não é constante. Em uma representação de intervalos de tempo, por exemplo, a base é o vetor T = (24, 60, 60), ou seja, T = (horas, minutos, segundo). Sistemas de numeração com base mista estão fora do escopo deste trabalho. Maiores detalhes sobre o sistema posicional base mista podem ser encontrados em (PARHAMI, 2000) e (HWANG, 1979).

A.1 Bases

Em sistemas numéricos posicionais, base ou *radix* é usualmente igual ao tamanho do conjunto de dígitos, ou seja, o número de dígitos únicos, incluindo o zero, que o sistema numérico posicional usa para representar números. Portanto um sistema numérico base *r* apresenta o conjunto implícito de dígitos {0, ..., *r*-1}.

Os sistemas numéricos mais comuns são:

- Base 10 – sistema numérico decimal. É utilizado em aritmética e apresenta o conjunto de dígitos {0, ..., 9}.
- Base 2 - sistema numérico binário. É usado em circuitos digitais e apresenta o conjunto de dígitos {0, 1}.
- Base 8 – sistema numérico octal. Apresenta o conjunto de dígitos {0, ..., 7}.
- Base 16 – sistema numérico hexadecimal. Apresenta o conjunto de dígitos {0, ..., 9, A, ..., F}.

Em computação normalmente esses quatro sistemas são utilizados: o decimal para a entrada e saída de dados, pois é o sistema mais utilizado pelos usuários diariamente; o binário é utilizado para realização interna dos cálculos pelo computador e os sistemas octal e hexadecimal são utilizados como forma compacta de representação interna, pois requerem menos espaço para representar os dados.

A.2 Sistema Numérico Binário

É um sistema numérico posicional com base 2 ($r = 2$) utilizado para a representação de inteiros. Em um número binário de comprimento n, cada dígito (chamado de bit) pode apresentar apenas os dois valores, 0 ou 1, isto é, o sistema binário possui o conjunto de dígitos {0, 1}.

Um número binário de comprimento n representa o valor inteiro

$$X = \sum_{i=0}^{n-1} x_i 2^i \tag{3}$$

Na equação 3 o peso do x_i é a i-ésima potência de 2.

Como o princípio de operação dos sistemas digitais baseia-se em um modelo de dois estados (mapeados para dois níveis de tensão distintos), o sistema binário é ideal para ser utilizado nos cálculos. A cada estado atribui-se um dos dígitos do conjunto {0,1}, ou seja, dígito 0 para um nível de tensão (geralmente, 0V) e dígito 1 para o outro nível de tensão (geralmente, Vdd, que é a tensão de alimentação).

Já o comprimento n de um número binário determina o intervalo de representação dos valores.

A.2.1 Representações de Números Negativos

Existem muitas formas para representar números negativos, porém as três formas mais utilizadas são **Sinal e Magnitude**, também chamado de Método **Sinal-Magnitude**, **Complemento**, e **Excesso** (PARHAMI, 2000) (KOREN, 2002) (GAJSKI, 1997).

Estas representações podem ser utilizadas em qualquer base. Neste trabalho, será detalhada aplicação dos mesmos no sistema binário, isto é, $r = 2$, como especificado na figura 2.1, apresentada no capítulo 2.

Na representação em **Sinal-Magnitude**, um número contém duas partes: o sinal e a magnitude, $X = \langle s, m \rangle$. Ambas são representadas separadamente, pois o primeiro dígito do número representa o sinal e os (*n-1*) dígitos restantes representam a magnitude do número. O sinal + ou – indica o valor positivo ou negativo da magnitude.

No caso dos números binários, o sinal é representado por um bit adicional, sendo que um bit 0 indica um valor positivo, enquanto um bit 1 indica um valor negativo. Normalmente o bit mais significativo (*Most Significant Bit* – MSB) é utilizado para representar o sinal e os demais bits menos significantes representam a magnitude. Desta forma os números deste sistema diferem apenas pelo seu bit mais significativo. Por exemplo, os números + 40_{10} e - 40_{10} são respectivamente **0**0101000_2 e **1**0101000_2.

A representação em sinal-magnitude possui a mesma quantidade de números negativos e positivos, sendo que qualquer número de n bits esta dentro de uma faixa de representação de $[-2^{n-1}+1,\ +2^{n-1}-1]$. Assim para um número X de $n = 8$, a faixa de representação é $-127 \le X \le +127$. Isso faz com que sinal-magnitude apresente a vantagem de possuir uma faixa de representação simétrica. Além disso, esta representação possui outras vantagens: é intuitiva, apresenta simplicidade conceitual e a negação, ou seja, a mudança de sinal é simples e ocorre através da inversão do bit de sinal.

Porém, a representação em sinal-magnitude apresenta duas desvantagens. A primeira reside no fato de haver duas representações possíveis para o zero, conforme ilustrado na Figura A.1. A segunda se refere ao custo do *hardware* necessário para realizar a adição, o qual tende a ser maior do que o observado em outras representações. Para uma simples adição, é necessário comparar os sinais dos dois operandos: caso os sinais sejam diferentes, a adição se transforma em uma subtração, e o sinal do resultado será o sinal do operando com maior magnitude. Caso os sinais dos operandos sejam iguais, basta somar as magnitudes, sendo que o resultado herdará o sinal dos operandos (PARHAMI, 2000). A Figura A.2 apresenta o circuito que executa a adição de números representados em sinal-magnitude usando pré e pós-complementação.

00000000(+0)	10000000(-0)
(a)	(b)

Figura A.1: Representações do valor 0 em sinal-magnitude, com n = 8 bits

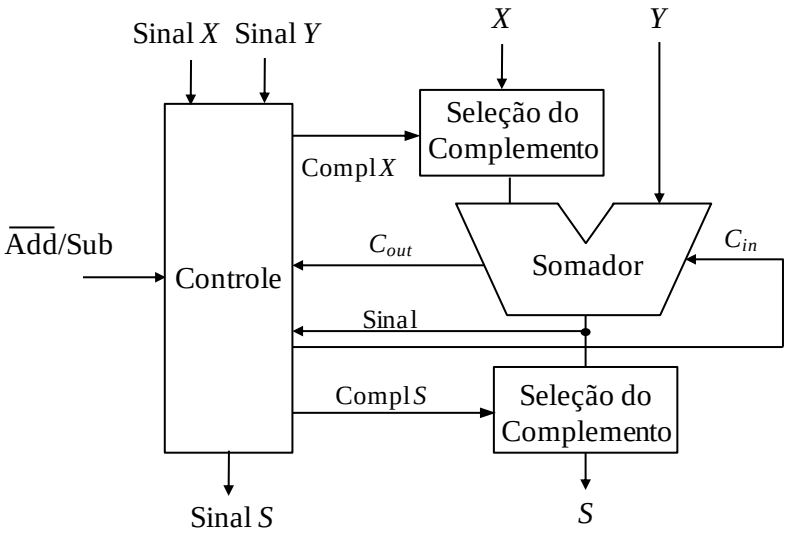


Figura A.2: Adição usando pré-complementação e pós-complementação de números no sistema sinal-magnitude.

A representação em **complemento** foi criada para diminuir o atraso das operações de adição e subtração e facilitar a implementação das mesmas através da eliminação da comparação entre magnitudes, necessária na representação em sinal-magnitude.

Para cada sistema de base r existem dois tipos de complemento: **complemento da base** (*radix complement* ou *r's complement*) e **complemento de dígito** (*digit, diminished-radix complement,* ou *(r-1)'s complement*).

Em ambos os tipos de complemento, os números positivos são representados da mesma forma que em sinal-magnitude. Porém, os números negativos são representados diferentemente.

Considerando somente números inteiros, isto é, $k = n$ e $l = 0$, um número Y pode ser representado como

$$Y = \sum_{i=0}^{k-1} y_i r^i \qquad (4)$$

Onde *k* é a quantidade de dígitos da parte inteira e *l* é a quantidade de dígitos da parte fracionária do número.

Nessa definição se assume que a quantidade de dígitos de um número é exatamente *k*, e se o resultado de qualquer operação gerar mais de *k* dígitos, somente os *k* dígitos menos significativos serão considerados.

Nas representações em complemento (complemento de base e de dígito), o complemento $\overline{Y}$ de um número *Y* de *k* dígitos é obtido através da equação 5.

$$\overline{Y} = C - Y \tag{5}$$

Caso seja realizado o complemento de base $C = r^k$. Já se o complemento a ser realizado for o complemento de dígito $C = r^k - ulp$, onde *ulp* significa uma unidade na última posição (*unit in the last position – ulp*) e é um conceito utilizado quando não existe distinção entre as partes inteira e fracionária de um número. Logo, *ulp* é o peso do dígito menos significativo: $ulp = r^{-l}$. Para números inteiros $ulp = 1$.

Portanto, o complemento de base e o complemento de dígito são obtidos através das equações 6 e 7 , respectivamente.

$$\overline{Y} = r^k - Y \tag{6}$$

$$Y' = (r^k - ulp) - Y \tag{7}$$

onde o complemento de base de *Y* será representado como $\overline{Y}$, ou seja, barrado para diferenciar do complemento de dígito que será representado por Y'.

Considerando, por exemplo, os números 987 e 123, ambos com 3 dígitos, o complemento de base é obtido pela subtração de $10^3 = 1000$. Neste caso o complemento de base é chamado de complemento de 10 e resulta em 13 e 877 para os números 987 e 123, respectivamente.

Uma forma simplificada de obter-se o complemento de dígito de um número *Y* é através da complementação em paralelo de todos os dígitos que formam o número. O complemento de um único dígito y_i é dado como

$$y_i' = (r-1) - y_i \tag{8}$$

Deste modo, o complemento de dígito Y' pode ser definido como

$$Y' = \sum_{i=0}^{k-1} y_i' \tag{9}$$

Já o complemento de base também pode ser obtido de maneira simplificada reescrevendo a equação 5 como

$$\overline{Y} = C - Y = (r^k - 1) - Y + 1 \tag{10}$$

Sabendo que $ulp = 1$, logo, $(r^k - 1) = (r^k - ulp)$ e o complemento de dígito $Y' = (r^k - ulp) - Y = (r^k - 1) - Y$, desta forma substituindo na equação 10 Y' obtém-se

$$\overline{Y} = Y' + 1 \tag{11}$$

Portanto, pode-se definir através da equação 11 que o complemento de base é o complemento de dígito adicionado de 1, ou seja, é obtido através da complementação de todos os dígitos que compõe o número e logo após adicionando-se 1 ao valor encontrado.

Considerando novamente o valor 123, o complemento de dígito encontrado é 876.

$$y_i' = (r-1) - y_i$$

1 $\quad y_1' = (10-1) - 1 = 8$

2 $\quad y_2' = (10-1) - 2 = 7$

3 $\quad y_3' = (10-1) - 3 = 6$

Desta forma, para obter-se o complemento de base do valor 123, basta somar 1 ao valor 876 encontrado no complemento de dígito. Logo, o complemento de base de 123 é 877.

Como o interesse deste trabalho é o sistema numérico binário, ou seja, $r = 2$ com o conjunto de dígitos {0, 1}, apenas a aplicação da representação em complemento à aritmética binária será detalhada a seguir.

A aplicação do complemento de dígito (*(r-1)'s complement*) no sistema binário é chamado de **complemento de 1** (*one's complement*), uma vez que em binário, $r = 2$.

A representação em **complemento de 1** possui uma faixa de representação simétrica para números de $k = n$ bits de $[-2^{n-1}+1,\ 2^{n-1}-1]$, o que resulta em duas representações para zero, uma positiva, representada por 000...0, e uma negativa, representada por 111...1, o que caracteriza uma desvantagem desta representação.

O valor numérico de um número Y representado em complemento de 1 é definido como

$$Y = -y_{n-1}(2^{n-1}-1) + \sum_{i=0}^{n-2} y_i 2^i \qquad (12)$$

A representação em complemento de 1 é obtida através da determinação paralela do complemento y_i' de todos os bits i de um determinado número Y , ou seja, é obtida pela negação de todos seus bits,. Importante observar que todos os n bits do número devem ser negados, inclusive o bit mais significativo. Por exemplo, os números + 10_{10} e - 10_{10} são representados em complemento de 1 com $n = 8$ por 00001010_2 e 11110101_2, respectivamente.

Diferentemente da representação em sinal-magnitude, na adição e na subtração de números em complemento de 1, todos os dígitos que compõem os operandos participam da operação, incluindo o dígito mais significativo. Por outro lado, a presença de um *carry* de saída não significa necessariamente que ocorreu um *overflow*, mas representa a necessidade de uma etapa de correção.

Para adicionar um número positivo X com um número negativo Y, se $|X| > |Y|$ o resultado será positivo e o *carry* de saída não faz parte da soma final, sendo porém uma indicação de que o valor 1 deve ser adicionado à posição menos significativa do resultado. A Figura A.3 ilustra um exemplo de adição em complemento de 1 onde $|X| > |Y|$. Caso $|X| < |Y|$, o resultado obtido será negativo e não haverá necessidade de correção.

carries	1	1	1	0	-	
		0	0	1	1	$(3)_{10}$
+		1	1	1	0	$(-1)_{10}$
		0	0	0	1	$1 \neq 2$
		+			1	
		0	0	1	0	$(2)_{10}$

Figura A.3: Adição de dois operandos representados em complemento de 1, com *n*=4

A aplicação do complemento de base (*r's complement*) no sistema binário é chamado de **complemento de 2** (*two's complement*), uma vez que em binário, *r* = 2.

O valor numérico de um número *Y* representado em **complemento de 2** é dado por

$$Y = \sum_{i=0}^{n-1} y_i 2^i \tag{13}$$

no caso em que o bit mais significativo é 0, o que significa que o número é positivo. Já para o caso onde o bit mais significativo for 1, significando que o número é negativo, seu valor numérico é obtido calculando-se a sua representação em complemento de 2 e logo após, aplicando-se a equação 13.

O complemento de 2 de um número binário *Y* é obtido em dois passos. No primeiro passo, calcula-se o complemento de 1 para todos os bits do número, isto é, inverte-se *Y*, bit a bit, incluindo o bit mais significativo. No segundo passo, adiciona-se 1 ao complemento de 1 obtido no primeiro passo, desprezando o *carry* de saída, caso este seja gerado. Este procedimento é ilustrado na Figura A.4.

1^o passo:		0	0	0	0	$(+0)_{10}$
		1	1	1	1	$(-0)_{10}$
2^o passo:						
carries	1	1	1	1	-	
		1	1	1	1	
		+			1	
		0	0	0	0	

Figura A.4: Representação do zero em complemento de 2, com *n*=4

A faixa de representação para números de *n* bits em complemento de 2 é $[-2^{n-1}, 2^{n-1}-1]$, com uma única representação para o zero. Em função disto, há uma representação a mais para números negativos. Assim, o número binário -2^{n-1} não tem um número positivo equivalente e, se o complemento deste número for calculado, um sinal de *overflow* deve ser gerado, o qual é calculado por meio de uma operação XOR entre os dois últimos *carries*.

A principal vantagem do complemento de 2 reside no fato de que tanto os números positivos como os números negativos podem ser somados pelo mesmo circuito (uma vez que não é necessário conferir os sinais dos números para realizar as operações de soma e de subtração), resultando em circuitos de menor custo, em comparação com aqueles utilizados nas representações anteriores,

É importante observar que a aplicação do procedimento ilustrado na Figura A.4 em um número que já se encontra em complemento de 2, transforma-o em um número

positivo. Desta forma, tal procedimento pode ser utilizado para trocar o sinal de um número.

A Figura A.5 mostra um exemplo de circuito capaz de realizar as operações de adição, de subtração e de troca de sinal (do operando *Y*), quando os operandos *X* e *Y* estão representados em complemento de 2.

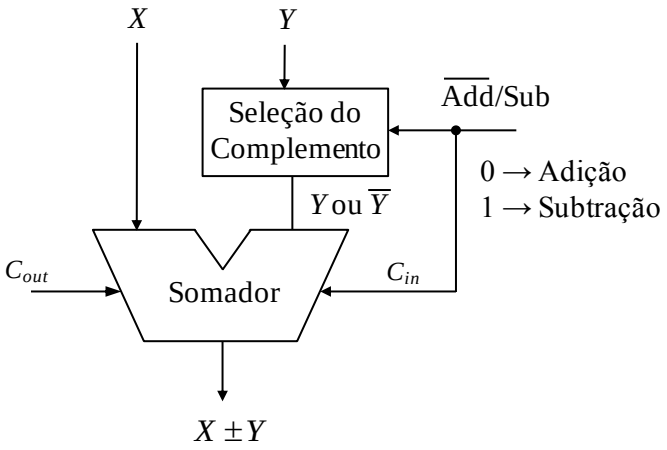


Figura A.5: Arquitetura de um Somador/Subtrator para números representados por complemento de 2.

Na realização de operações aritméticas com números em complemento de 2 é preciso verificar se o resultado gerado está correto ou se ocorreu *overflow*, pois a geração de *carry* de saída não indica necessariamente a ocorrência de *overflow*.

Na adição de dois números *X* e *Y* representados em complemento de dois não ocorre *overflow* se eles possuírem sinais diferentes, independentemente da existência ou não de um *carry* de saída. Porém, se *X* e *Y* possuírem o mesmo sinal e o resultado apresentar um sinal diferente, então ocorre *overflow*, independente da geração de um *carry* de saída. Conforme já mencionado anteriormente, na adição de números em complemento de 2 o *overflow* pode ser calculado por meio de uma operação XOR entre os dois últimos *carries*.

Ao contrário dos sistemas sinal-magnitude e complemento, onde o único parâmetro importante é a quantidade de dígitos que compõem o número, na **representação em excesso**, além da quantidade de dígitos, também é necessário especificar um valor do excesso, *B* (PARHAMI, 2000).

O valor numérico de um número inteiro de *n* bits representado em excesso de B na base *r* = 2 é dado por

$$X = \sum_{i=0}^{n-1} x_i 2^i - B \qquad (14)$$

Com um valor de *B* é possível representar inteiros em excesso de –*B* até $2^n - (B+1)$, usando *n* bits. Geralmente os valores atribuídos à *B* são 2^{n-1} ou $2^{n-1} - 1$, o que resulta em uma faixa de representação quase simétrica.

Por exemplo, a Tabela A.1 mostra como números inteiros com sinal na faixa de [-4, 3] podem ser representados como valores sem sinal de 0 até 7 através de uma representação em excesso de $B = 2^{3-1} = 4$.

Tabela A.1: Exemplo representação em excesso de $B = 4$

Binário Padrão	*Números sem sinal*	*Excesso de 4*
000	0	- 4
001	1	- 3
010	2	- 2
011	3	-1
100	4	0
101	5	1
110	6	2
111	7	3

Com representações de n bits e excesso de 2^{n-1}, o bit mais significativo indica o sinal do valor representado, como nos sistemas sinal-magnitude e complemento. Porém, ao contrário destes, o valor do bit igual a 0 indica um número negativo, ao passo que o valor 1 indica um número positivo.

O objetivo da representação em excesso é manter a ordem relativa dos números, de modo que 000...0 representa o menor valor e 111...1 representa o maior valor. Por exemplo, na Tabela A.2 os números representados possuem $n = 4$ e $B = 8$, o que resulta 0000 = - 8 e 1111 = +7.

A representação em excesso apresenta algumas diferenças em relação às representações em sinal-magnitude e em complemento:

- Mediante a escolha de um B adequado, a representação em excesso permite se representar mais números positivos do que negativos.
- A representação em excesso pode ter no máximo uma representação para zero, como na representação complemento de 2. Porém, é possível escolher B de forma que não exista zero.
- As operações aritméticas tornam-se ligeiramente mais complexas para números representados em excesso. Logo, é necessário um *hardware* específico, que considere não apenas a quantidade de bits n dos operandos, mas também o valor de B. Na adição e subtração de números em excesso deve ser somado ou subtraído o valor B no resultado destas operações.

A Tabela A.2 reúne os sistemas de representação de números negativos sinal-magnitude, complemento de 1, complemento de 2 e excesso.

Tabela A.2: Sistemas de representação sinal-magnitude, complemento de 1, complemento de 2 e excesso

Decimal	*Binário sem sinal*	*Sinal-Magnitude*	*Complemento de 1*	*Complemento de 2*	*Excesso de 8*
+8	1000	---	----	---	----
+7	0111	0111	0111	0111	1111
+6	0110	0110	0110	0110	1110
+5	0101	0101	0101	0101	1101
+4	0100	0100	0100	0100	1100
+3	0011	0011	0011	0011	1011
+2	0010	0010	0010	0010	1010
+1	0001	0001	0001	0001	1001
0	0000	0000	0000	0000	1000
-0	---	1000	1111	----	----
-1	----	1001	1110	1111	0111
-2	----	1010	1101	1110	0110
-3	----	1011	1100	1101	0101
-4	----	1100	1011	1100	0100
-5	----	1101	1010	1011	0011
-6	----	1110	1001	1010	0010
-7	----	1111	1000	1001	0001
-8	----	----	----	1000	0000

APÊNDICE B COMANDOS HSPICE PARA MEDIR POTÊNCIA DINÂMICA

Para executar a medida da potência média consumida pelos somadores, este trabalho utilizou a função *Pseudo Random Bit Generator Source* (PRBS) do HSpice, a qual faz uso de um *Linear Feedback Shift Register* (LFSR) para gerar uma sequência de bits pseudo-aleatória (SYNOPSYS, 2011). A Figura B.1 apresenta a sintaxe da fonte de tensão LFSR usada para gerar os vetores pseudoaleatórios de entrada. A Tabela B.1 descreve os parâmetros usados na definição de uma LFSR.

```
Vxxx n+ n- LFSR <(> vlow vhigh tdelay trise tfall
rate seed <[>+ taps <]><)>
```

Figura B.1: Sintaxe da fonte de tensão LFSR

Tabela B.1: Parâmetros usados na definição de uma LFSR.

Parâmetros	**Descrição**
LFSR	Especifica a fonte de tensão como PRBS
vlow	O nível mínimo de tensão
vhigh	O nível máximo de tensão
tdelay	Especifica o tempo de atraso inicial para a primeira transição
trise	Especifica a duração da rampa de subida (em segundos), a partir do valor inicial para o valor máximo do pulso
tfall	Especifica a duração da rampa de descida (em segundos) a partir do valor máximo do pulso até a volta ao seu valor inicial
rate	A taxa de bits
seed	O valor inicial carregado no registrador de deslocamento.
taps	Os bits usados para a realimentação.

A Figura B.2 apresenta um exemplo de fonte de tensão LFSR utilizadas nas medidas de potência média das arquiteturas de somadores.

```
Vin0 A0 0 LFSR (0 vdd 0.1u 1n 1n 333meg 612 [25, 22])
```

Figura B.2: Exemplo de fonte de tensão LFSR

Com relação aos parâmetros da LFSR utilizados no presente trabalho, vale observer o seguinte:

- A tensão da fonte varia de 0 até vdd.
- O atraso inicial para a primeira transição é 0.1u.
- Os atrasos de subida e descida são 1n.
- A freqüência é 333meg bits/s.
- A semente é 612.
- O polinômio do LFSR é [25, 22].

Dados os parâmetros para geração dos vetores pseudoaleatórios, os comandos HSpice utilizados para medir a potência média foram os seguintes.

```
.tran 0.1n 3u
.meas tran avgpow avg power from=0.1u to 2.9u
```

Figura B.3: Comandos HSpice utilizados para medir potência média

APÊNDICE C RESULTADOS EXPERIMENTAIS DETALHADOS

C.1 Comparação entre somadores não-protegidos: RCA e BSDA_BSD

Antes de realizar a análise dos somadores protegidos, é conveniente que se analisem os somadores não-protegidos RCA e BSDA_BSD, quando descritos no nível de transistor e simulados utilizando o modelo PTM 45nm. A Figura B.1 mostra as curvas do atraso crítico dos somadores RCA e BSDA_BSD de 4, 8, 16 e 32 bits.

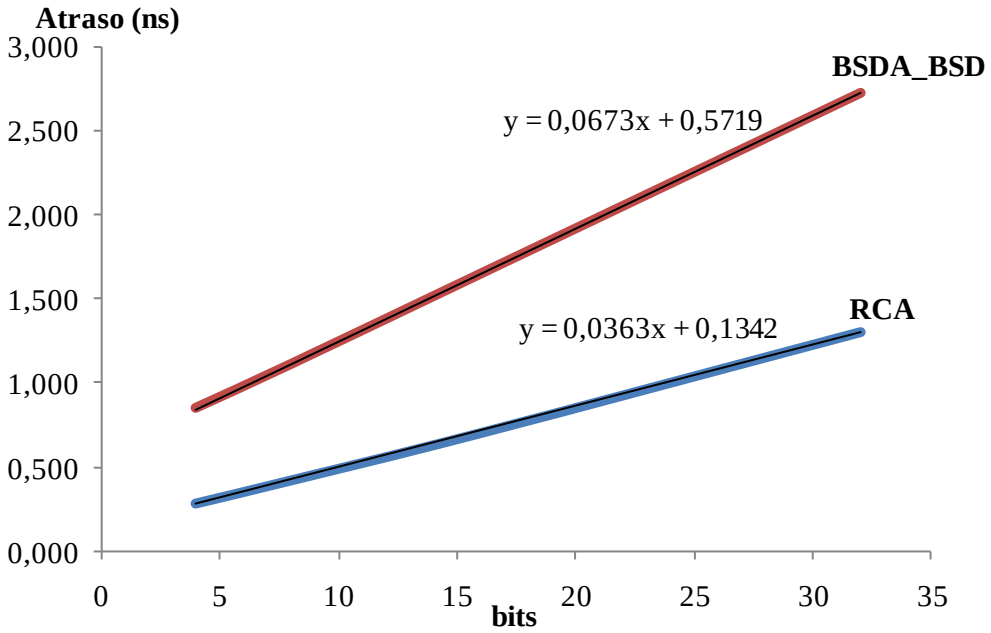


Figura C.1: Atraso crítico dos somadores RCA e BSDA_BSD

Apesar de saber-se que a adição BSD é *carry-free* e portanto, possui atraso constante em relação ao comprimento dos operandos, nota-se nitidamente que o BSDA_BSD apresenta um atraso consideravelmente maior que o RCA. Isso se deve ao uso de conversores para converter o sistema de representação binário convencional para representação redundante BSD NAF. Vale lembrar que há um conversor para cada operando de entrada e um conversor para o resultado.

Os conversores utilizados pelo BSDA_BSD também resultam em um número consideravelmente maior de transistores, em relação ao RCA, conforme pode ser visto no gráfico da Figura C.2.

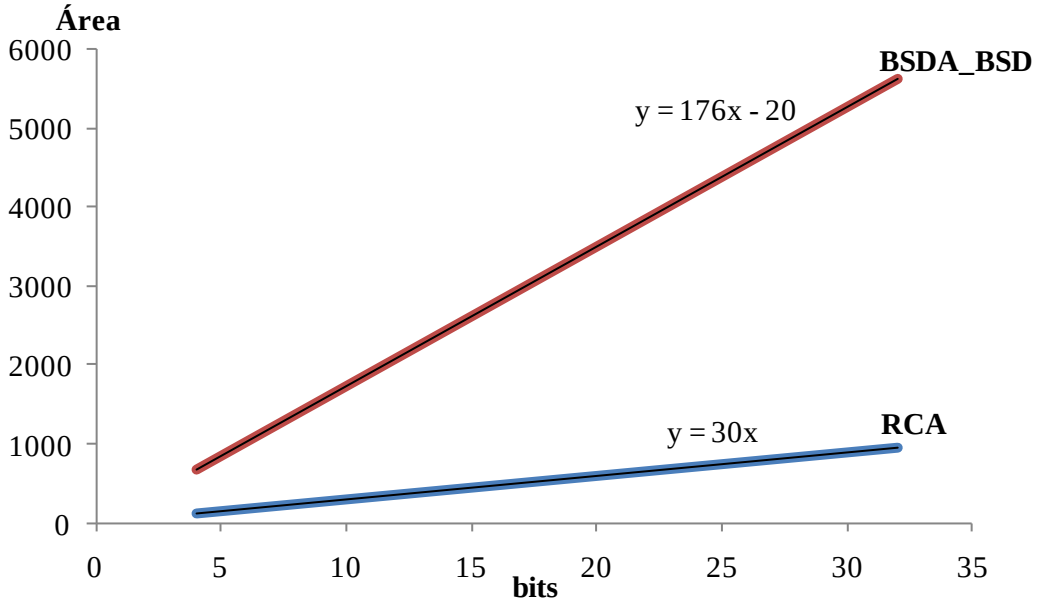


Figura C.2: Número de transistores utilizados pelos somadores RCA e BSDA_BSD

O gráfico da Figura C.3 mostra que o BSDA_BSD consome bem mais potência que o RCA, o que também pode ser explicado devido aos conversores do BSDA_BSD.

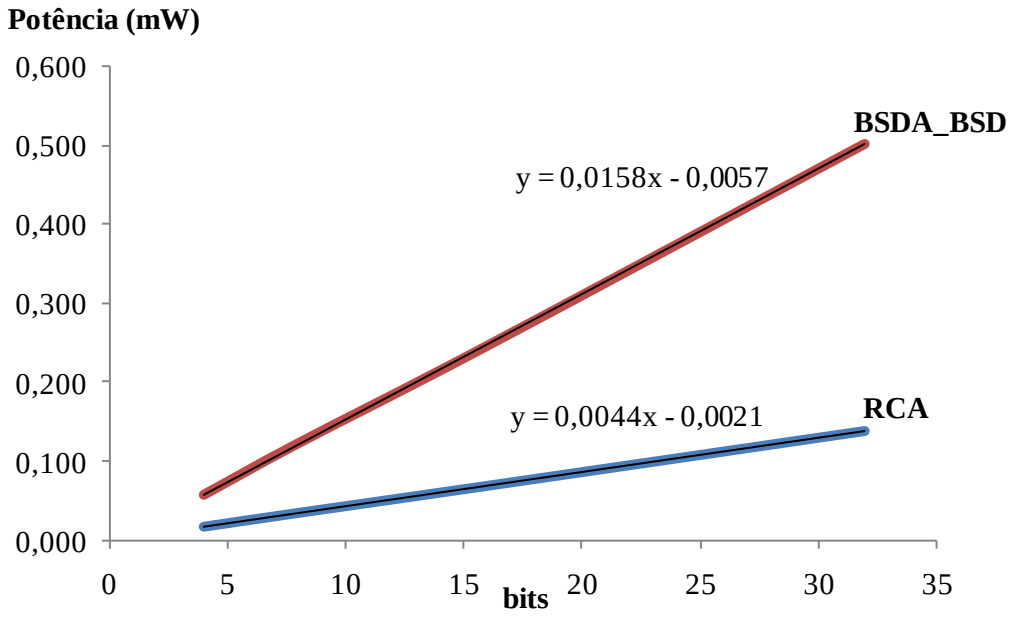


Figura C.3: Potência consumida pelos somadores RCA e BSDA_BSD

A Tabela C.1 concentra os resultados de atraso crítico, potência e número de transistores para os somadores RCA e BSDA_BSD. Também são mostrados os valores de acréscimo de atraso crítico, de potência e de transistores, tomando-se o RCA como referência. Note-se que o atraso e a potência evoluem de maneira quase linear com relação ao número de bits dos operandos.

Tabela C.1: Comparação do atraso crítico, número de transistores e potência consumida entre os somadores RCA e BSDA_BSD.

	Atraso (ns)			***Potência (mW)***			***Nº de Transistores***		
bits	**RCA (1)**	**BSDA_BSD (2)**	**Acrésc. [(2)/(1)-1*100]**	**RCA (1)**	**BSDA_BSD (2)**	**Acrésc. [(2)/(1)-1*100]**	**RCA (1)**	**BSDA_BSD (2)**	**Acrésc. [(2)/(1)-1*100]**
4	0,287	0,845	194%	0,016	0,057	269%	120	684	470%
8	0,423	1,107	161%	0,033	0,123	268%	240	1388	478%
16	0,704	1,648	134%	0,068	0,246	261%	480	2796	483%
32	1,302	2,728	110%	0,139	0,501	261%	960	5612	485%

Analisando-se a Tabela C.1, conclui-se que o BSDA_BSD apresenta um atraso crítico, em média, 150% maior que o atraso crítico do RCA. Porém, conforme aumenta o número de bits dos operandos, a diferença entre os atrasos críticos do RCA e do BSDA_BSD diminui. Isso se deve ao fato de que o atraso inserido pelos conversores de entrada e saída utilizados no BSDA_BSD tornar-se dominante em relação ao atraso do bloco da soma BSD propriamente. Assim, para somadores de 4 bits, o atraso crítico do BSDA_BSD é 194% maior do que o atraso crítico do RCA, sendo a maior diferença encontrada. Já para somadores de 32 bits, o atraso crítico do BSDA_BSD é 110% maior do que o atraso crítico do RCA, sendo esta a menor diferença observada. Estes resultados nos permitem especular que, para um comprimento de dados muito grande, o acréscimo de atraso do BSDA_BSD ficará próximo de 100%, uma vez que o atraso do BSDA_BSD será dominado pela soma entre o atraso de um conversor de entrada e o atraso do conversor de saída, sendo ambos atrasos semelhantes ao atraso de um RCA de igual comprimento.

Em relação ao número de transistores, pode-se observar que o BSDA_BSD utiliza, em média, 479% mais transistores do que o RCA. Isso se deve ao fato de cada conversor utilizado pelo BSDA_BSD praticamente equivaler a um RCA, sendo que o BSDA_BSD utiliza três conversores (dois para os operandos de entrada e um para a saída) e ainda possui a lógica que implementa a soma propriamente dita.

Apesar de utilizar , em média, 479% mais transistores que o RCA, o BSDA_BSD consome "apenas" 265% mais potência que o RCA. Isso acontece devido à codificação BSD NAF gerada pelos blocos conversores BIN/BSD (que converte operandos em complemento de dois para BSD NAF), a qual reduz o consumo dos blocos ADD1 e ADD2 dos BSDA_BSDs que executam a soma propriamente dita e também reduz a potência consumida pelo bloco conversor BSD/BIN (que converte operandos em BSD NAF para complemento de dois). Tal redução de potência tem como origem o menor número de chaveamentos de sinais que ocorre quando BSD NAF é adotada.

A fim de se examinar em detalhes como o uso de BSD NAF influencia o consumo de potência do BSDA_BSD, a Tabela C.2 mostra os valores de potência consumida pelos blocos que compõem o BSDA_BSD. Estes blocos foram simulados separadamente com vetores de entrada aleatórios, isto é, sem a codificação BSD NAF gerada pelas saídas do conversor BIN/BSD. Eles também foram simulados conectados, de modo que o somador BSD recebe como entrada a codificação BSD NAF da saída do conversor BIN/BSD. A Tabela C.2 também mostra os valores de redução do consumo

de potência obtidos com a codificação BSD NAF, tomando os blocos sem codificação NAF como referência.

Tabela C.2: Consumo de potência dos blocos que compõem o BSDA_BSD

	Com Codificação BSD NAF		Sem Codificação BSD NAF		Redução	
bits	**Soma (1)**	**Conv Bsd/Bin (2)**	**Soma (3)**	**Conv Bsd/Bin (4)**	**[(1)/(3)-1*100]**	**[(2)/(4)-1*100]**
4	0,008	0,014	0,040	0,025	-79%	-44%
8	0,021	0,019	0,080	0,044	-73%	-57%
16	0,049	0,028	0,162	0,083	-70%	-66%
32	0,105	0,048	0,323	0,160	-68%	-70%

Pela análise da Tabela C.2, conclui-se que a soma BSD propriamente dita, quando operando sobre codificação BSD NAF, consome, em média 72% menos potência do que quando operando sobre dados sem codificação BSD NAF. Entretanto, observa-se que esta redução de potência é ligeiramente menor para somadores com maior número de bits.

Analisando-se ainda a Tabela C.2, observa-se que o conversor BSD/BIN quando operando com codificação BSD NAF, consome, em média, 59% menos potência que quando operando sem codificação BSD NAF. Porém, ao contrário da soma, a redução de potência é maior para somadores com mais bits.

O BSDA_BSD apresenta vantagem em relação ao RCA nos quesitos avaliados quando se considera somente os blocos que executam a soma propriamente dita. O gráfico da Figura C.4 apresenta o atraso crítico dos somadores RCA e BSDA_BSD sem levar em conta o atraso inserido nas etapas de conversão.

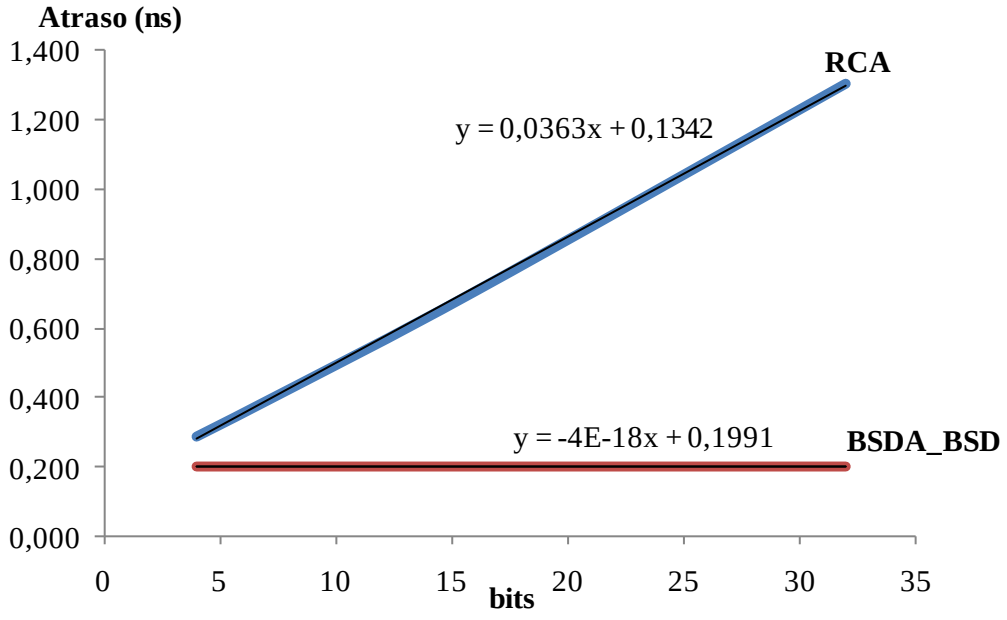


Figura C.4: Atraso crítico dos somadores RCA e soma BSD sem conversores.

Como pode ser visto na Figura C.4, o atraso crítico da soma propriamente dita no BSDA_BSD é constante e menor que o atraso crítico do RCA, pois como mencionado

anteriormente, a soma BSD é *carry-free*, ou seja, com o aumento do número de bits dos operandos, o atraso crítico não aumenta, pois não há propagação de *carry*.

O gráfico da Figura C.5 apresenta o número de transistores utilizados pelo RCA e pela soma BSD, desconsiderando os conversores de entrada e saída do BSDA_BSD.

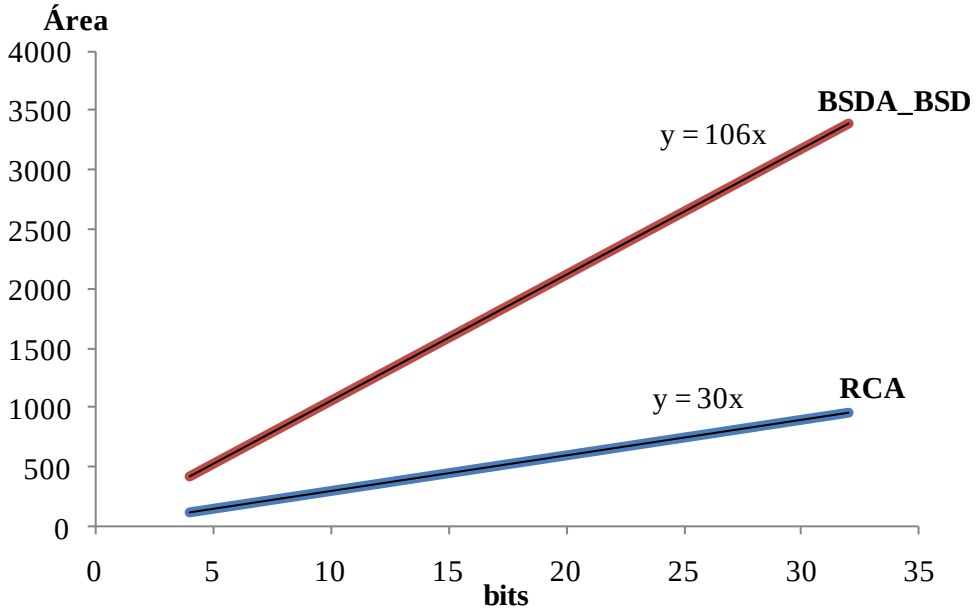


Figura C.5: Número de transistores dos somadores RCA e soma BSD sem conversores.

A Figura C.5 evidencia que, mesmo desconsiderando-se os conversores, a soma BSD utiliza mais transistores que o RCA.

O gráfico apresentado na Figura C.6 mostra a potência consumida pelo RCA e pela soma BSD, desconsiderando os conversores do BSDA_BSD, bem como a potência que seria consumida pela soma BSD caso esta fosse realizada com entradas aleatórias, isto é, sem usar representação BSD NAF.

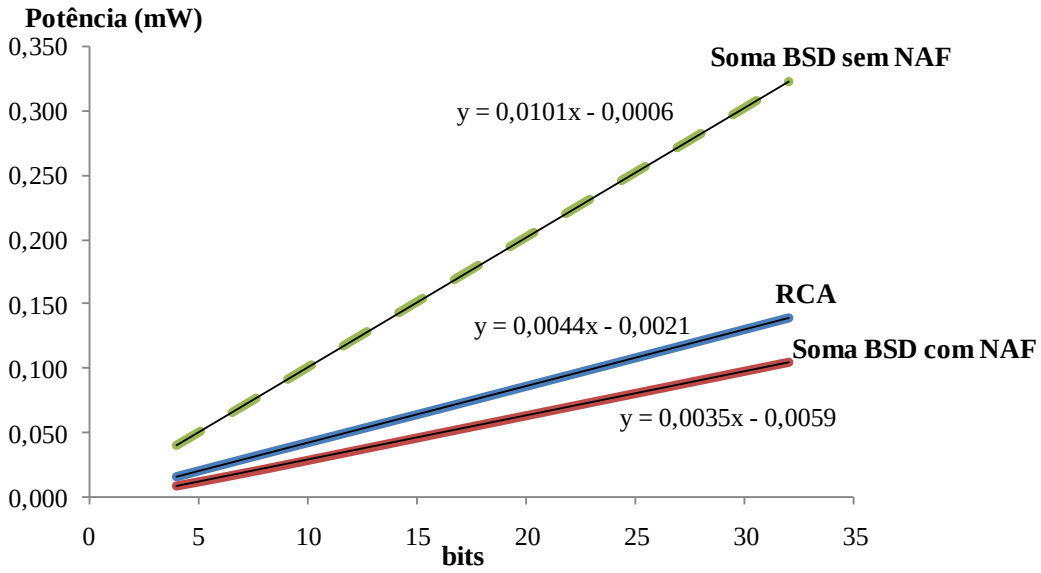


Figura C.6: Potência consumida pelos somadores RCA e soma BSD sem conversores.

A Figura C.6 mostra que, apesar da soma BSD utilizar muito mais transistores que o RCA, a potência consumida pela soma BSD é menor que a potência consumida pelo RCA quando a codificação BSD NAF é adotada. O gráfico da Figura C.6 também mostra quanto a soma BSD consome, caso as entradas fossem aleatórias (isto é, sem o efeito da codificação BSD NAF).

A Tabela C.3 mostra os resultados de atraso crítico, número de transistores e consumo de potência para os somadores RCA e soma BSD. Também são mostrados os valores de acréscimo do atraso crítico, de transistores utilizados e de potência consumida, tomando-se o RCA como referência.

Tabela C.3: Comparação do atraso crítico, número de transistores e potência consumida entre os somadores RCA e soma BSD sem conversores.

	Atraso (ns)			***Potência (mW)***			***Nº de Transistores***		
bits	**RCA (1)**	**BSD (2)**	**Acréscimo [(1)/(2)-1*100**	**RCA (1)**	**BSD (2)**	**Acréscimo [(1)/(2)-1*100**	**RCA (1)**	**BSD (2)**	**Acréscimo [(2)/(1)-1*100**
4	0,287	0,199	44%	0,016	0,008	86%	120	424	253%
8	0,423	0,199	113%	0,033	0,021	56%	240	848	253%
16	0,704	0,199	254%	0,068	0,049	38%	480	1696	253%
32	1,302	0,199	554%	0,139	0,105	32%	960	3392	253%

Como pode ser visto na Tabela C.3, conforme aumenta o comprimento dos operandos, a diferença entre o atraso crítico do RCA e da soma BSD aumenta de maneira praticamente constante. Isso se deve ao fato de que o atraso crítico da soma BSD é constante mesmo com o aumento do comprimento dos operandos. Desta forma, o RCA apresentou um atraso crítico, em média, 241% maior que o atraso crítico da soma BSD.

Em relação ao consumo de potência, o RCA consome, em média, 53% mais potência que a soma BSD. Porém, conforme o número de bits dos somadores aumenta, a diferença entre o consumo de potência dos somadores diminui. Assim, o consumo de potência do RCA de 4 bits apresenta a maior diferença em relação à soma BSD de 4 bits, sendo 86% maior. Já o RCA de 32 bits apresenta a menor diferença em relação à soma BSD de 32 bits, consumindo 32% mais potência.

Em se tratando de transistores, a soma BSD necessita, em média, de 253% mais transistores do que o RCA, sendo que para operandos de maior comprimento, a diferença entre o número de transistores usados pela soma BSD e o número de transistores usados pelo RCA mantém-se constante.

C.2 Comparação entre o RCA (não-protegido) e os RCAs protegidos (RCATMR e RCAIOI)

O gráfico da Figura C.7 e a Tabela C.4 apresentam os atrasos críticos dos somadores RCA, RCATMR e RCAIOI. A Tabela C.4 também apresenta os acréscimos de atrasos críticos do RCATMR e do RCAIOI em relação ao RCA.

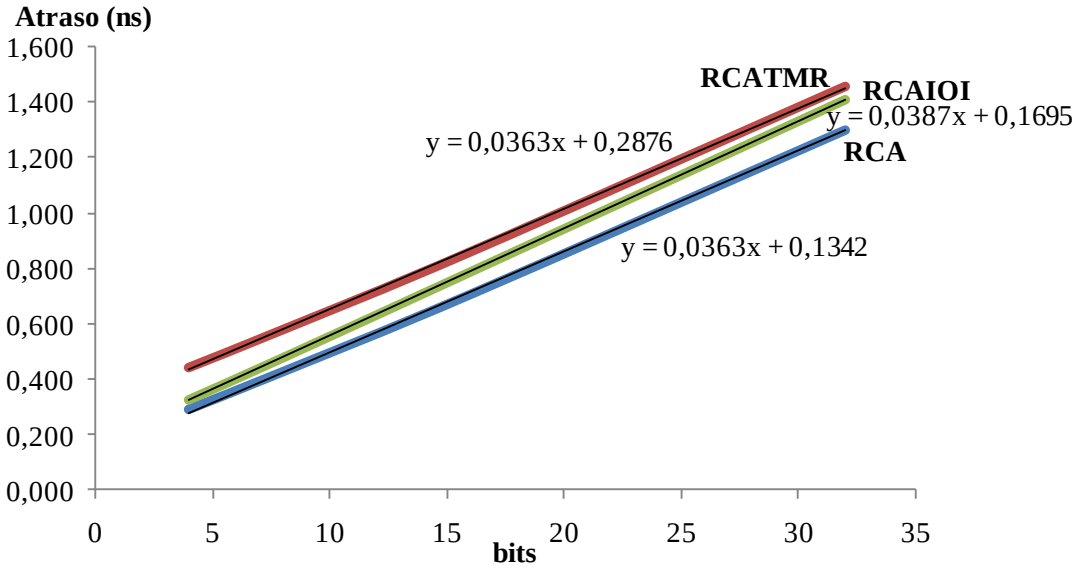


Figura C.7: Atraso crítico dos somadores RCA

Como era esperado, os somadores RCATMR e RCAIOI apresentam atraso crítico maior que o RCA. Pode ser visto no gráfico da Figura C.7 e também na Tabela C.4 que o acréscimo no atraso crítico ocasionado pela aplicação da técnica RESI ao RCA é praticamente constante, ao passo que o acréscimo de atraso ocasionado pela aplicação da técnica TMR ao RCA diminui com o aumento do comprimento dos operandos.

Também pode ser observado que o RCAIOI apresentou um atraso menor que o RCATMR. Isso se deve ao fato de que os SCCRs usados no RCAIOI não utilizam portas XORs, mas somente portas NAND. Porém, com o aumento do número de bits dos somadores, o atraso do RCAIOI aproxima-se do atraso do RCATMR, pois o atraso inserido pela técnica de inversão das entradas não é constante, ao contrário do atraso inserido pelo votador, no caso do RCATMR.

Tabela C.4: Comparação entre o atraso crítico (em ns) dos somadores RCA, RCAIOI e RCATMR

bits	**RCA (1)**	**RCATMR (2)**	**RCAIOI (3)**	**Acréscimo [(2)/(1)-1*100]%**	**Acréscimo [(3)/(1)-1*100]%**
4	0,287	0,441	0,326	53	14
8	0,423	0,577	0,477	36	13
16	0,704	0,858	0,786	22	12
32	1,302	1,455	1,407	12	8

O atraso do RCATMR é, em média, 31% maior do que o atraso do RCA. Mas para o RCATMR de 32 bits, o acréscimo de atraso é de apenas 12%. Isso demonstra que a contribuição do votador ao atraso crítico do RCATMR tende a cair significativamente com o aumento do comprimento dos operandos.

Já o atraso do RCAIOI é, em média, 12% maior do que o atraso do RCA. Observa-se, porém, que o acréscimo de atraso varia pouco, caindo para 8%, no caso do RCAIOI de 32 bits. O acréscimo no atraso do RCAIOI se deve ao fato deste possuir multiplexadores de entrada e saída e um registrador de saída. (Note-se que o circuito verificador de paridade dos *carries* redundantes existente no RCAIOI opera em paralelo com o registrador de saída e portanto, não influencia no atraso crítico.)

O gráfico da Figura C.8 apresenta o número de transistores utilizados pelos somadores RCA, RCATMR e RCAIOI.

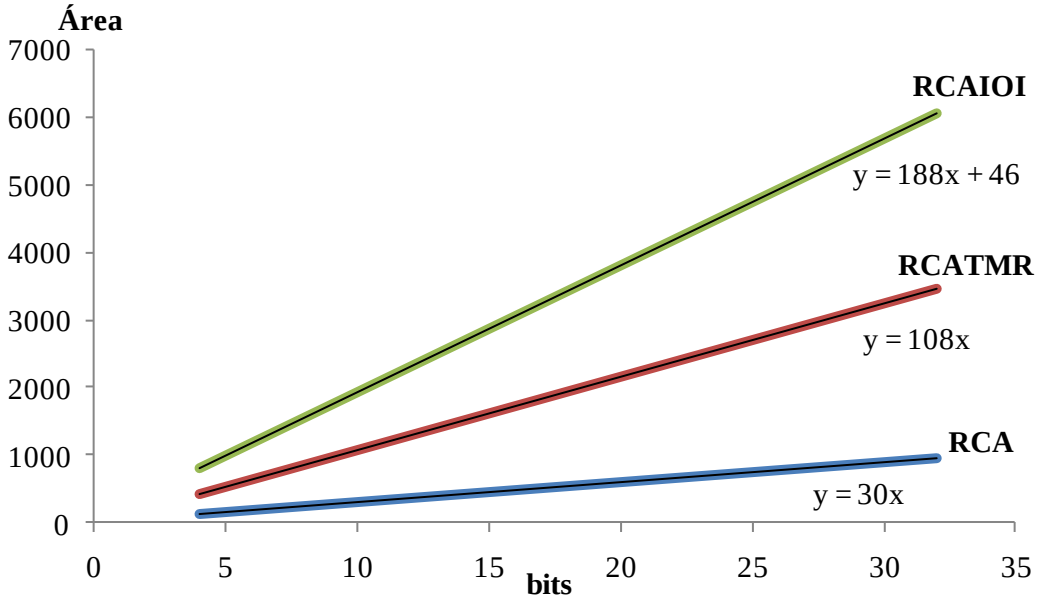


Figura C.8: Número de transistores utilizados pelos somadores RCA

O RCATMR apresenta um acréscimo de transistores significativo em relação ao RCA, o que era esperado, já que a técnica TMR consiste na triplicação do bloco que se deseja proteger, neste caso, o próprio somador RCA. Também pode ser visto que o RCAIOI utilizou uma quantidade de transistores ainda maior do que o RCATMR. Isso se deve ao fato do RCAIOI possuir um bloco verificador de paridade, multiplexadores de entrada para cada operando, registrador e multiplexador de saída, e portas XORs para a verificação da paridade dos operandos e dos *carries* redundantes.

A Tabela C.5 mostra os resultados referentes ao número de transistores utilizados pelo RCA, pelo RCATMR e pelo RCAIOI, bem como o acréscimo de transistores do RCATMR e RCAIOI, em relação ao RCA.

Tabela C.5: Comparação entre o número de transistores utilizados pelo RCA, RCATMR e RCAIOI

bits	**RCA (1)**	**RCATMR (2)**	**RCAIOI (3)**	**Acréscimo [(2)/(1)-1*100]%**	**Acréscimo [(3)/(1)-1*100]%**
4	120	432	798	260	565
8	240	864	1550	260	546
16	480	1728	3054	260	536
32	960	3456	6062	260	531

Analisando-se a Tabela C.5, verifica-se que o RCATMR necessita de 260% mais transistores do que o RCA, independentemente do comprimento dos operandos. Isso porque o número de bits do votador é igual ao comprimento dos operandos. Já o RCAIOI utiliza, em média, 545% mais transistores que o RCA, sendo 565% mais transistores para operandos de 4 bits e 531% mais transistores no caso de operandos de 32 bits. Tal variação deve-se ao fato da lógica extra inserida pela aplicação da técnica

RESI tornar-se menos significativa diante dos transistores utilizados pelo RCA de mais bits.

Considerando-se apenas as arquiteturas protegidas, percebe-se que o RCAIOI necessita, em média, 79% mais transistores do que o RCATMR.

O gráfico da Figura C.9 apresenta a potência consumida pelos RCA RCATMR e RCAIOI.

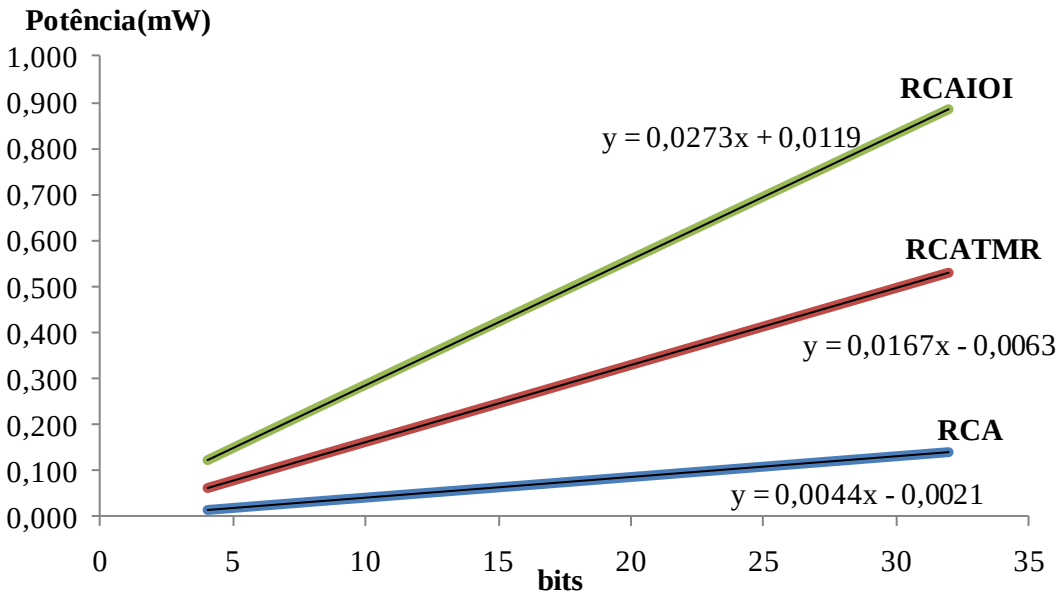


Figura C.9: Consumo de potência dos somadores RCA

Observando-se a Figura C.9, nota-se que tanto o RCATMR quanto o RCAIOI possuem um consumo de potência significativamente maior do que o RCA, sendo que o RCAIOI é a arquitetura RCA protegida que mais consome, dentre as duas analisadas. Este comportamento é previsível e pode ser explicado pela relação entre o número de transistores destes somadores.

A Tabela C.6 detalha os resultados referentes ao consumo de potência dos somadores RCA e RCA, RCATMR e RCAIOI, bem como os resultados da comparação entre estas arquiteturas.

Tabela C.6: Comparação entre o consumo de potência (em mW) dos somadores RCA, RCATMR e RCAIOI.

bits	**RCA (1)**	**RCATMR (2)**	**RCAIOI (3)**	**Acréscimo [(2)/(1)-1*100]%**	**Acréscimo [(3)/(1)-1*100]%**
4	0,016	0,060	0,120	288	674
8	0,033	0,128	0,230	283	592
16	0,068	0,261	0,449	283	559
32	0,139	0,528	0,884	280	537

Como pode ser visto na Tabela C.6, o RCATMR apresentou um acréscimo na potência consumida praticamente constante de, em média, 284%, como era esperado, visto que utilizou 260% mais transistores que o RCA. Já o acréscimo no consumo de potência do RCAIOI variou, reduzindo-se com o aumento do comprimento dos somadores. Isso ocorre porque a potência consumida pela lógica extra usada na técnica

RESI torna-se menos significativa diante da potência consumida pelo RCA. Assim, o consumo de potência do RCAIOI de 4 bits também apresentou a maior diferença em relação ao RCA de 4 bits, ficando 674% maior, ao passo que o RCAIOI de 32 bits apresentou um acréscimo de 537% em relação ao RCA de 32 bits.

Portanto, o RCAIOI consome, em média, 590% mais potência do que o RCA, o que era esperado, visto que a aplicação da técnica RESI adicionou uma quantidade considerável de lógica extra ao RCA (em média 545% a mais de transistores).

Ao se comparar o RCATMR com o RCAIOI, nota-se que o RCAIOI consome, em média, 80% mais potência do que o RCATMR.

C.3 Comparação entre BSDA_BSD e BSDA_BSDs protegidos (BSD1-3_BSD e BSDAPar_BSD)

O gráfico da Figura C.10 apresenta o atraso crítico do BSDA_BSD não-protegido, juntamente com o atraso crítico que este apresentou, quando protegido com codificação 1 de 3 e verificação de paridade.

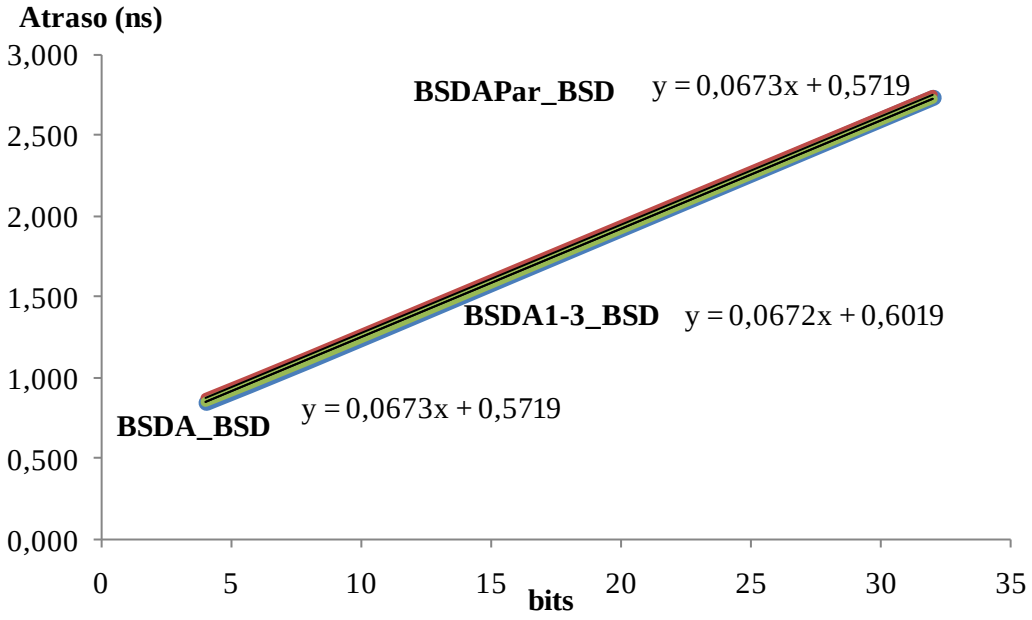


Figura C.10: Atraso crítico dos somadores BSDA_BSD

Como pode ser visto na Figura C.10, o atraso crítico dos somadores BSDA_BSD, BSDA1-3_BSD e BSDAPar_BSD apresentou um comportamento praticamente linear, ou seja, com o aumento do número de bits, o atraso crítico de ambos os somadores tem um acréscimo aproximadamente constante. Também pode ser visto que as três arquiteturas BSDA_BSD obtiveram praticamente o mesmo atraso. Isso se deve ao uso de conversores para as conversões entre o sistema de representação binário convencional e o sistema de representação redundante BSD NAF.

A Tabela C.7 apresenta, de maneira detalhada, os resultados referentes ao atraso crítico dos somadores BSDA_BSD, BSDA1-3_BSD e BSDAPar_BSD. Também são mostrados os resultados da comparação entre a versão não protegida com as versões protegidas.

Tabela C.7: Comparação entre o atraso crítico (em ns) dos somadores BSDA_BSD, BSDA1-3_BSD e BSDAPar_BSD.

bits	BSDA_BSD (1)	BSDA 1-3_BSD (2)	BSDA Par_BSD(3)	Acréscimo [(2)/(1)-1*100]	Acréscimo [(3)/(1)-1*100]
4	0,845	0,873	0,845	3%	0%
8	1,107	1,137	1,107	3%	0%
16	1,648	1,677	1,648	2%	0%
32	2,728	2,754	2,728	1%	0%

Analisando a Tabela C.7, percebe-se que a aplicação da Codificação 1 de 3 gera um acréscimo de, em média, 2%, no atraso crítico do BSDA_BSD. Também pode-se observar que o acréscimo do atraso não foi constante, isto é, conforme aumenta o número de bits das arquiteturas, a diferença entre os atrasos críticos do BSDA_BSD e BSDA1-3_BSD diminui. Assim, o atraso crítico do BSDA1-3_BSD de 4 bits apresenta a maior diferença em relação BSDA_BSD de 4 bits, ficando 3% maior e o BSDA1-3_BSD de 32 bits apresenta a menor diferença em relação ao BSDA_BSD, ficando apenas 1% maior.

Isso deve-se ao fato dos atrasos críticos dos conversores de entrada e saída serem mais significativos que o atraso do bloco que executa a soma propriamente dita. Os atrasos críticos das somas BSD dos somadores BSDA_BSD e BSDA1-3_BSD 3 são, respectivamente, 0,199ns e 0,225ns para todos os comprimentos dos operandos de entrada. Portanto, os blocos que executam a soma apresentaram um acréscimo no atraso crítico de, em média, 13% com a aplicação da codificação 1 de 3.

Já o BSDAPar_BSD não apresenta aumento no atraso crítico em relação ao BSDA_BSD não-protegido, visto que a verificação da paridade executada pela técnica ocorre em paralelo ao cálculo da soma. A Tabela C.8 discrimina os atrasos críticos (em ns) dos blocos conversores de entrada e saída dos somadores BSDA_BSD. Também são apresentados os valores de acréscimo do atraso crítico inserido pela aplicação da codificação 1 de 3, tomando-se o conversor BIN/BSD e o conversor BSD /BIN como refêrencia.

Considerando os somadores protegidos, o BSDA_BSD apresenta praticamente o mesmo atraso crítico quando protegido com código 1 de 3 e verificação de paridade, ficando o BSDA1-3_BSD apenas 2%, em média, mais lento que o BSDAPar_BSD.

Tabela C.8: Atraso crítico dos conversores de entrada e saída dos BSDA_BSDs.

bits	Conv BIN/BSD (1)	Conv BIN/BSD 1de3 (2)	Acréscimo [(2)/(1)-*100]	Conv BSD/BIN (3)	Conv BSD/BIN 1de3 (4)	Acréscimo [(4)/(3)-*100]
4	0,310	0,313	1**%**	0,34	0,34	0**%**
8	0,437	0,442	1**%**	0,47	0,47	0**%**
16	0,695	0,699	0**%**	0,75	0,75	0**%**
32	1,209	1,209	0**%**	1,32	1,32	0**%**

A aplicação da codificação 1 de 3 não resulta em um acréscimo significativo nos conversores BIN/BSD, ficando, em média 1% maior, pois as mudanças na lógica dos mesmos são mínimas, e conforme o número de bits das arquiteturas aumenta, a diferença entre os atrasos críticos diminui chegando a 0. Isso se deve ao fato da cadeia

de *carry* existente no conversor BIN/BSD tornar-se mais significativa diante do atraso inserido pelas mudanças na lógica do conversor para a aplicação da codificação para somadores de maior comprimento. Já o conversor BSD/BIN não apresenta acréscimo de atraso, pois devido a codificação escolhida para representar os dígitos BSD ser a mesma para todos os somadores descritos, em todas as arquiteturas foi utilizado o mesmo circuito para a realização da conversão do resultado calculado em BSD NAF para complemento de 2.

A aplicação da técnica de verificação de paridade não necessita de mudanças na lógica dos conversores de entrada e saída, portanto não altera o atraso crítico dos mesmos.

O gráfico da Figura C.11 apresenta o número de transistores utilizados pelo BSDA_BSD, juntamente com o número de transistores que este utilizou, quando protegido com codificação 1 de 3 e verificação de paridade.

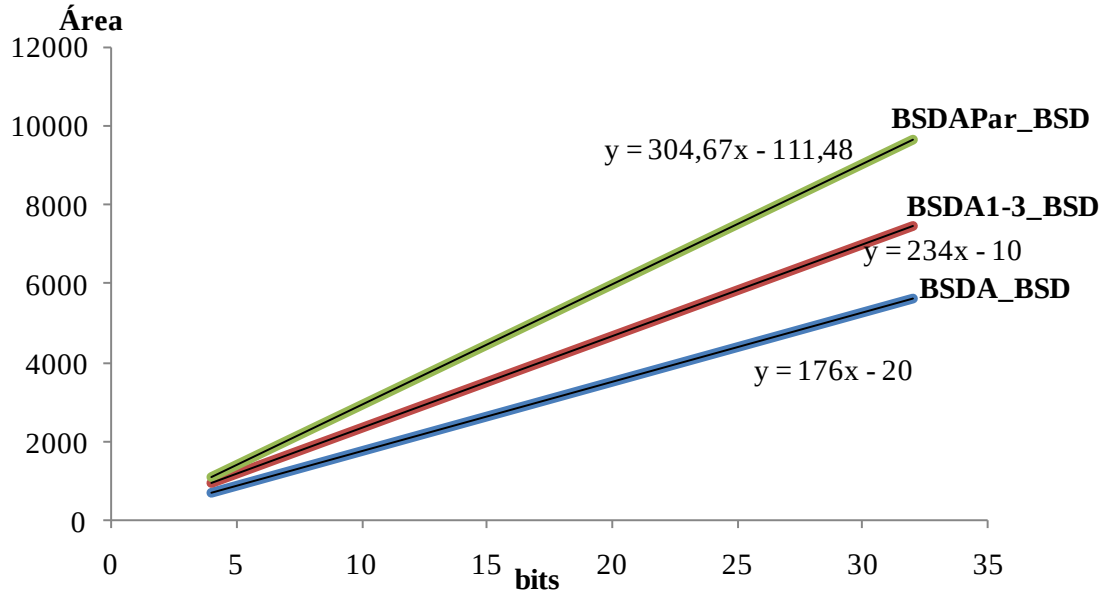


Figura C.11: Número de transistores utilizados pelos somadores BSDA_BSD, BSDA1-3_BSD e BSDPar_BSD

Pode ser observado na Figura C.11 que as versões protegidas utilizam um número maior de transistores que o BSDA_BSD,o que era esperado devido a lógica extra necessária para implementar as técnicas de proteção, e que o BSDAPar_BSD utiliza uma quantidade maior de transistores que o BSDA1-3_BSD. Isso se deve ao fato do BSDAPar_BSD utilizar árvores de portas CMOS XORs para implementar os circuitos que verificam a paridade

A Tabela C.9 apresenta os resultados referentes ao número de transistores utilizados pelos BSDA_BSD, BSDA1-3_BSD e BSDPar_BSD, bem como os resultados da comparação entre a versão não protegida com as versões protegidas.

Tabela C.9: Comparação entre o número de transistores dos somadores BSDA_BSD, BSDA1-3_BSD e BSDPar_BSD.

bits	BSDA_ BSD (1)	BSDA 1-3_BSD (2)	BSDAPar_ BSD (3)	Acréscimo [(2)/(1)-1*100]	Acréscimo [(3)/(1)-1*100]
4	684	926	1110	35%	62%
8	1388	1862	2326	34%	68%
16	2796	3734	4758	34%	70%
32	5612	7478	9640	33%	72%

A aplicação da codificação 1 de 3 não resulta em um acréscimo tão significativo na utilização de transistores em relação ao número de transistores utilizados pelo BSDA_BSD, ficando o BSDA1-3_BSD, em média, 34% maior. Também pode-se observar que conforme aumenta o número de bits das arquiteturas, a diferença entre o número de transistores utilizados pelo BSDA1-3_BSD e pelo BSDA_BSD manteve-se praticamente constante.

Já o BSDAPar_BSD utiliza, em média, 68% mais transistores que o somador BSDA_BSD. Porém, ao contrário do BSDA1-3_BSD, com o aumento do número de bits dos somadores, a diferença entre o número de transistores utilizados pelo BSDAPar_BSD e pelo BSDA_BSD apresenta um ligeiro aumento. Isso ocorre porque a aplicação da técnica de verificação de paridade torna o somador BSDA_BSD mais irregular do que a aplicação da codificação 1 de 3.

Desta forma ao serem considerados os somadores protegidos, observa-se que o BSDAPar_BSD utiliza, em média, 25% mais transistores que o BSDA1-3_BSD.

A Tabela C.10 discrimina o número de transistores utilizados por um dos blocos conversores de entrada e o conversor de saída dos somadores BSDA_BSD e BSDA1-3_BSD. Também são mostrados os valores de acréscimo do número de transistores gerado pela aplicação da codificação 1 de 3, tomando-se o conversor BIN/BSD e o conversor BSD /BIN como refêrencias.

Tabela C.10: Número de transistores dos conversores de entrada e saída dos somadores BSDA_BSD e BSDA1-3_BSD.

bits	Conv BIN/BSD (1)	Conv BIN/BSD 1de3 (2)	Acréscimo [(2)/(1)-1*100]	Conv BSD/BIN (3)	Conv BSD/BIN 1de3 (4)	Acréscimo [(4)/(3)-*100]
4	100	110	10%	160	160	0%
8	252	262	4%	288	288	0%
16	556	566	2%	544	544	0%
32	1164	1174	1%	1056	1056	0%

Como pode ser visto na Tabela C.10, a aplicação da codificação 1 de 3 não resulta em um acréscimo significativo no número de transistores utilizados pelos conversores BIN/BSD, ficando, em média 4% maior, pois as mudanças na lógica do circuito, necessárias para a aplicação da codificação, são mínimas e conforme aumenta o número de bits dos operandos de entrada a diferença entre o número de transistores utilizados pelo conversor BIN/BSD 1 de 3 em relação ao conversor BIN/BSD diminui.

O conversor BSD/BIN não apresenta acréscimo de transistores pois, como mencionado anteriormente, o circuito utilizado para realizar a conversão do resultado em BSD para complemento de dois é o mesmo para ambos os somadores. Já a aplicação da técnica de verificação de paridade não necessita de mudanças na lógica dos conversores de entrada e saída portanto não gera aumento no número de transistores utilizados pelos conversores.

A Tabela C.11 discrimina o número de transistores utilizados pelos blocos responsáveis pela soma BSD, propriamente dita, dos somadores BSDA_BSD e BSDA1-3_BSD. Também são mostrados os valores de acréscimo do número de transistores utilizados, gerado pela aplicação do Código 1 de 3, tomando-se a soma BSD como refêrencia.

Tabela C.11: Número de transistores dos blocos da soma BSD dos somadores BSDA_BSD e BSDA1-3_BSD.

bits	**Soma BSD (1)**	**Soma BSD 1de3 (2)**	**Acréscimo [(2)/(1)-*100]**
4	424	560	32%
8	848	1120	32%
16	1696	2240	32%
32	3392	4480	32%

Como pode ser visto na Tabela C.11, a aplicação da codificação 1 de 3 resulta em um aumento, em média, de 32% na utilização de transistores em relação ao BSDA_BSD. Também pode ser observado que conforme o número de bits dos somadores aumenta, a diferença entre o número de transistores utilizados pela soma BSD 1 de 3 em relação a soma BSD manteve-se constante. Isso se deve ao fato de que os bloco que executam a soma BSD possuem um comportamento linear, ou seja, ao dobrar o comprimento dos operandos de entrada, o número dos transistores utilizados por ambos os somadores também dobra.

Já a aplicação da técnica de verificação de paridade utiliza o mesmo bloco da soma BSD utilizado pelo BSDA_BSD e portanto não gera aumento no número de transistores utilizados pela soma BSD. Porém o BSDAPar_BSD utiliza blocos extras que verificam a paridade dos operandos de entrada, da soma intermediária e da soma final, um bloco preditor da paridade do *carry* intermediário, e dois blocos indicadores de erros que verificam a paridade, aumentando assim o número de transistores com a aplicação da técnica em relação ao BSDA_BSD.

O gráfico da

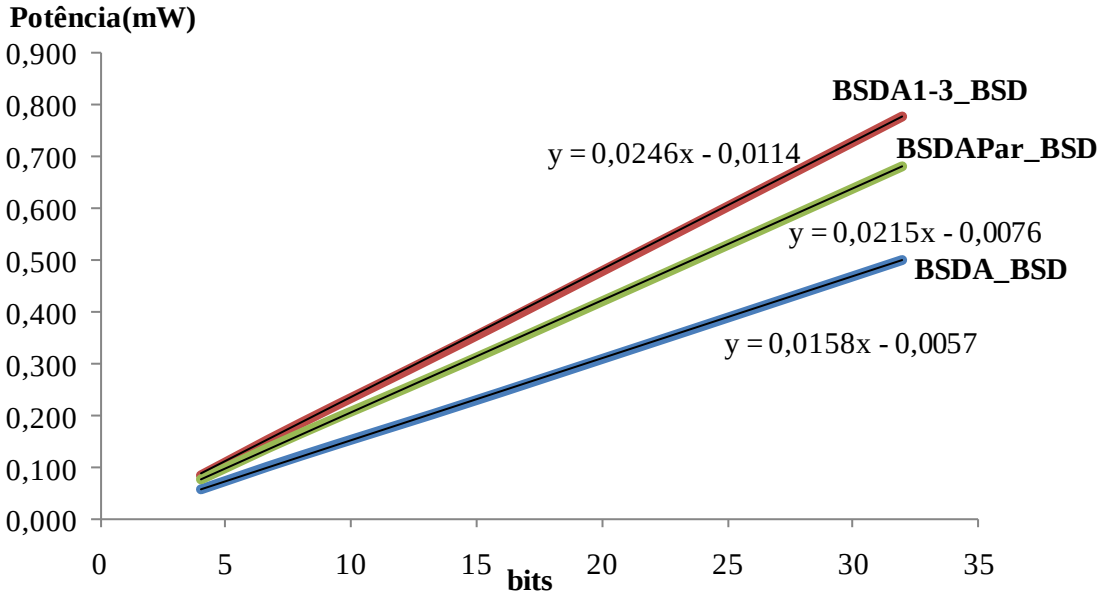


Figura C.12 apresenta o consumo de potência do BSDA_BSD, juntamente com a potência que este consome, quando protegido com codificação 1 de 3 e verificação de paridade.

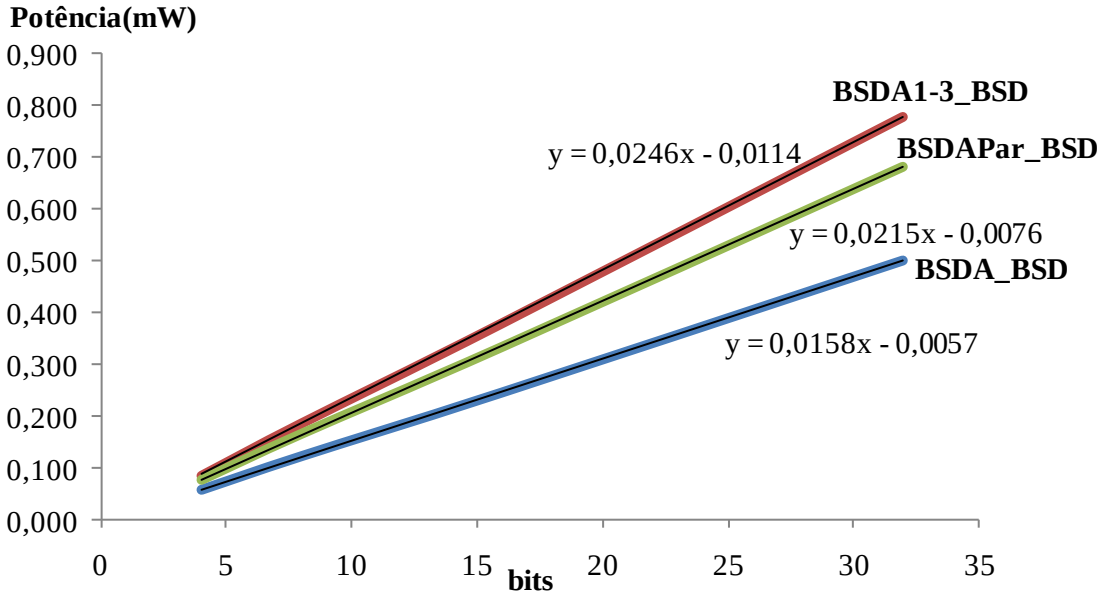
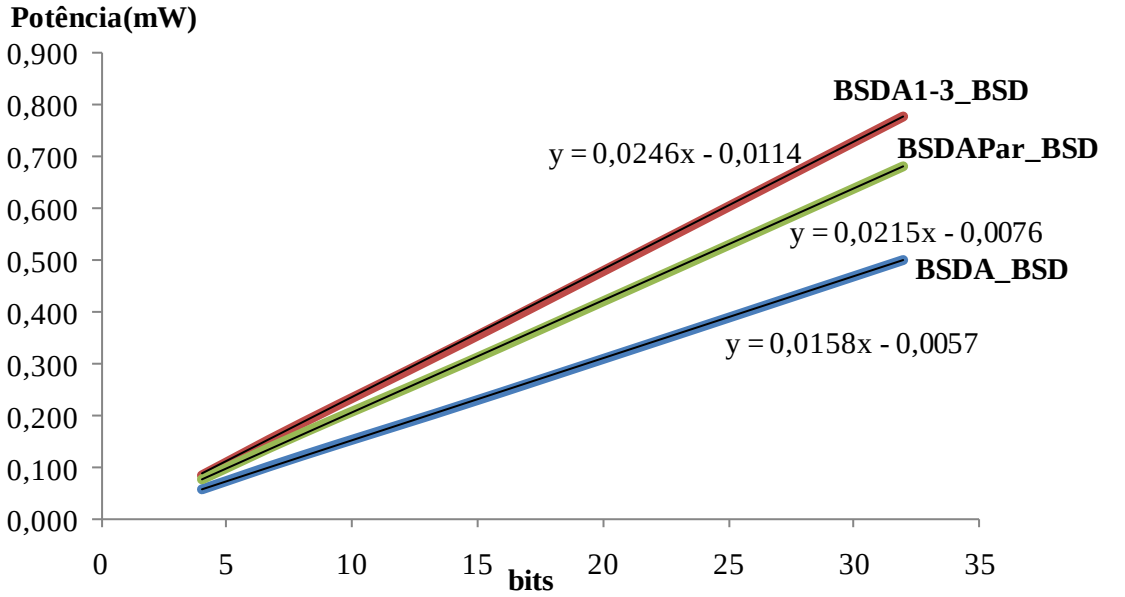


Figura C.12: Consumo de potência dos somadores BSDA_BSD, BSDA1-3_BSD e BSDAPar_BSD



Pode ser observado na Figura C.12 que as versões protegidas consomem mais potência que o somador BSDA_BSD, o que era esperado visto que a aplicação das técnicas de proteção resulta em uma maior quantidade de transistores aumentando assim a atividade de chaveamento do somador. Além disso, nota-se que apesar de utilizar mais transistores que o BSDA1-3_BSD, o somador BSDAPar_BSD consome menos potência que este. Isso se deve ao modo como foi implementada a verificação de paridade dos operandos de entrada, da soma intermediária, da soma final e do *carry* intermediário gerado pelo preditor de paridade. Ao invés de verificar-se a paridade de cada operando individualmente e da soma intermediária para posteriormente verificar se $PW = PX \oplus PY$, verificou-se as paridades $P(x_i)$ e $P(y_i)$ dos operandos em cada posição i, na saída dos conversores BIN/BSD e comparou-se a paridade da soma intermediária, $P(w_i)$, gerada nas saídas dos blocos ADD1. Desta forma $P(w_i) = P(x_i) \oplus P(y_i)$ resultará 1 somente se ocorrer uma falha.

Da mesma forma, ao invés de verificar individualmente a paridade da soma final, da soma intermediária e do *carry* intermediário para posteriormente verificar se $PS = PW \oplus PC$, verificou-se as paridades de $P(w_i)$ e $P(c_{i-1})$ nas saídas de cada bloco ADD1 e do preditor de paridade PC e comparou-se com a paridade da soma final, $P(s_i)$, gerada nas saídas dos blocos ADD2. Assim, $P(s_i) = P(w_i) \oplus P(c_{i-1})$ também resultará em 1 somente na presença de uma falha. Desta forma os blocos Indicador de Erro 1 e Indicador de Erro 2 não possuem um consumo de potência significativo, pois suas entradas serão praticamente sempre alimentadas com 0.

A Tabela C.12 apresenta os resultados referentes ao consumo de potência dos somadores BSDA_BSD, BSDA1-3_BSD e BSDPar_BSD, bem como os resultados da comparação entre a versão não protegida com as versões protegidas.

Tabela C.12: Comparação entre a potência consumida pelos somadores BSDA_BSD, BSDA1-3_BSD e BSDPar_BSD.

bits	BSDA_ BSD (1)	BSDA 1-3_BSD (2)	BSDAPar_ BSD(3)	Acréscimo [(2)/(1)-1*100]	Acréscimo de [(3)/(1)-1*100]
4	0,057	0,087	0,078	51%	36%
8	0,123	0,188	0,167	53%	36%
16	0,246	0,380	0,335	55%	36%
32	0,501	0,778	0,682	55%	36%

Como pode ser visto na Tabela C.12, a aplicação da codificação 1 de 3 resulta em um aumento, em média, de 54% no consumo de potência em relação ao BSDA_BSD. Já o BSDAPar_BSD apresenta um aumento constante, em média, de 36% em relação ao BSDA_BSD. Considerando os somadores protegidos observa-se que o BSDA1-3_BSD consome, em média, 13% mais potência que o BSDAPar_BSD.

A Tabela C.13 discrimina o consumo de potência de um dos conversores de entrada dos somadores BSDA_BSD e BSDA1-3_BSD e para o conversor de saída usado por ambos. Também são mostrados os valores de acréscimo de consumo de potência gerado pela aplicação da codificação 1 de 3, tomando-se o conversor BIN/BSD e o conversor BSD /BIN como referência.

Tabela C.13: Consumo de potência dos blocos conversores de entrada e saída dos somadores BSD e BSD 1 de 3.

bits	Conv BIN/BSD (1)	Conv BIN/BSD 1de3 (2)	Acréscimo [(2)/(1)-1*100	Conv BSD/BIN (3)	Conv BSD/BIN 1de3 (4)	Acréscimo [(4)/(3)-1*100
4	0,017	0,025	47%	0,014	0,014	0%
8	0,039	0,054	38%	0,019	0,019	0%
16	0,084	0,110	30%	0,028	0,028	0%
32	0,176	0,222	26%	0,048	0,048	0%

Como pode ser visto na Tabela C.13, a aplicação da codificação 1 de 3 resulta em um aumento, em média, de 35% no consumo de potência do bloco conversor BIN/BSD. Também pode ser observado que conforme aumenta o número de bits dos operandos, a diferença entre o consumo de potência do conversor BIN/BSD 1 de 3 em relação ao conversor BIN/BSD diminui. Desta forma, o conversor BIN/BSD 1 de 3 de 4 bits apresenta a maior diferença em relação ao conversor BIN/BSD de 4 bits, consumindo 47% mais potência que o conversor BIN/BSD de 4 bits. A menor diferença é apresentada pelos conversores de 32 bits, ficando o conversor BIN/BSD 1 de 3 26% maior que o conversor BIN/BSD.

A Tabela C.14 discrimina o consumo de potência dos blocos que executam a soma propriamente dita dos somadores BSDA_BSD e BSDA1-3_BSD. Também são mostrados os valores de acréscimo do consumo de potência gerado pela aplicação da codificação 1 de 3, tomando-se a soma BSD do BSDA_BSD como refêrencia. Além disso, mostra a redução, causada pela codificação BSD NAF, no consumo de potência dos blocos que executam a soma BSD dos somadores BSDA_BSD e BSDA1-3_BSD.

Tabela C.14: Consumo de potência dos blocos que executam a soma dos somadores BSDA_BSD e BSDA1-3_BSD.

Com Codificação BSD NAF				Sem Codificação BSD NAF			Redução	
bits	**BSD (1)**	**BSD 1de3(2)**	**Acréscimo [(2)/(1)-1*100]**	**BSDA (3)**	**BSD 1-3(4)**	**Acréscimo [(4)/(3)-1*100]**	**[(1)/(3)-1*100]**	**[(2)/(4)- 1*100]**
4	0,008	0,010	16%	0,040	0,059	48%	-79%	-84%
8	0,021	0,029	35%	0,080	0,120	50%	-73%	-76%
16	0,049	0,066	33%	0,162	0,236	46%	-70%	-72%
32	0,105	0,145	38%	0,323	0,477	48%	-68%	-70%

Como pode ser visto, a soma BSD do BSDA1-3_BSD, simulada com codificação BSD NAF, apresenta um acréscimo de 30% no consumo de potência em relação à soma BSD do BSDA_BSD. Ao simular tais blocos com entradas aleatórias é gerado um acréscimo de 48% no consumo de potência da soma BSD do BSDA1-3_BSD em relação ao consumo da soma BSD do BSDA_BSD.

O gráfico apresentado na Figura C.13 mostra a potência consumida pela soma BSD dos somadores BSDA_BSD e BSDA1-3_BSD, desconsiderando os conversores de entrada e saída dos BSDA_BSDs, bem como a potência consumida por ambas, quando simuladas com entradas aleatórias, isto é, sem a codificação BSD NAF.

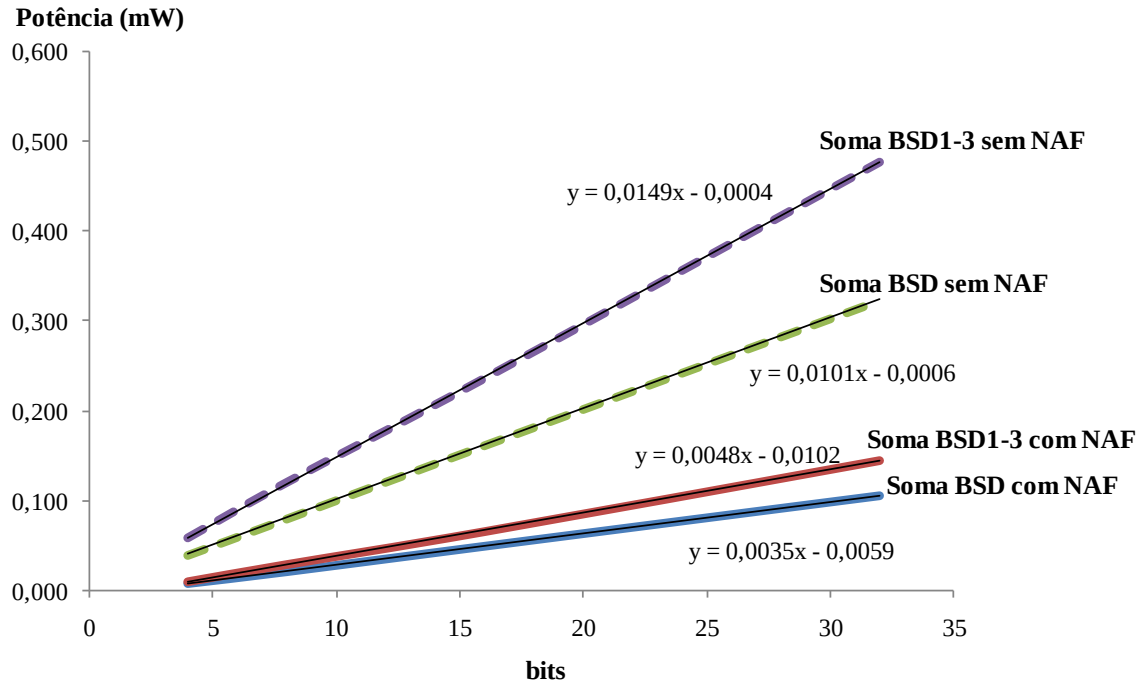


Figura C.13: Consumo de potência das soma BSD e soma BSD 1 de 3 com e sem o efeito da codificação BSD NAF

O gráfico da Figura C.13 apresenta o consumo dos blocos que executam a soma BSD nos somadores BSDA_BSD e BSDA1-3_BSD com a aplicação da codificação BSD NAF. Também pode ser visto no gráfico o consumo destes mesmos blocos quando simulados com entradas aleatórias. Pode se observar na Figura C.13 que a aplicação da codificação BSD NAF gera uma redução significativa no consumo de potência dos blocos que executam a soma BSD em ambos os somadores.

O comportamento da soma BSD para o BSDAPar_BSD foi omitido por ser o mesmo que o consumo do BSDA_BSD não-protegido.

Além dos blocos apresentados acima, o BSDA1-3_BSD possui o bloco Verificador de Erros. A Tabela C.15 apresenta os valores de atraso crítico, número de transistores e consumo de potência do Verificador de Erros.

Tabela C.15: Atraso crítico, número de transistores e potência consumida pelo Verificador de Erros

bits	Atraso(ns)	Potência(mW)	Nº de transistores
4	0,143	0,015	96
8	0,143	0,034	192
16	0,143	0,069	384
32	0,143	0,141	768

Como pode ser visto na Tabela C.15, o atraso do Verificador de Erros é constante com o aumento de bits, pois a verificação é realizada simultaneamente para todas as posições do resultado. Logo, o atraso do verificador não influencia no atraso crítico do BSDA1-3_BSD pois a verificação de erro ocorre em paralelo com o estágio de conversão de saída.

No caso do somador BSDPar_BSD, os valores de atraso crítico, número de transistores e consumo de potência acrescido pela aplicação da técnica podem ser vistos na Tabela C.16.

Tabela C.16: Valores de atraso crítico, número de transistores e consumo de potência acrescidos ao BSDA_BSD para a aplicação da técnica de verificação de paridade.

bits	Atraso Indicador de Erro 1	Atraso Indicador de Erro 2	Potência	Nº de transistores
4	0,533	0,594	0,020	426
8	0,620	0,686	0,044	938
16	0,715	0,780	0,089	1962
32	0,815	0,888	0,180	4028

Como no caso do verificador de Erros do BSDA1-3_BSD, o atraso dos blocos indicadores de erros do BSDPar_BSD não influencia no atraso crítico do somador, pois são executados em paralelo com o cálculo da soma.

C.4 Comparação entre RCA protegidos (RCATMR e RCAIOI) e BSDA protegidos (BSDA1-3_BSD e BSDAPar_BSD)

A Figura C.14 mostra o gráfico que apresenta os resultados de atraso crítico, gerados pela aplicação das técnicas de proteção TMR, codificação 1 de 3 e verificação de paridade, nos somadores RCA e BSDA_BSD, respectivamente.

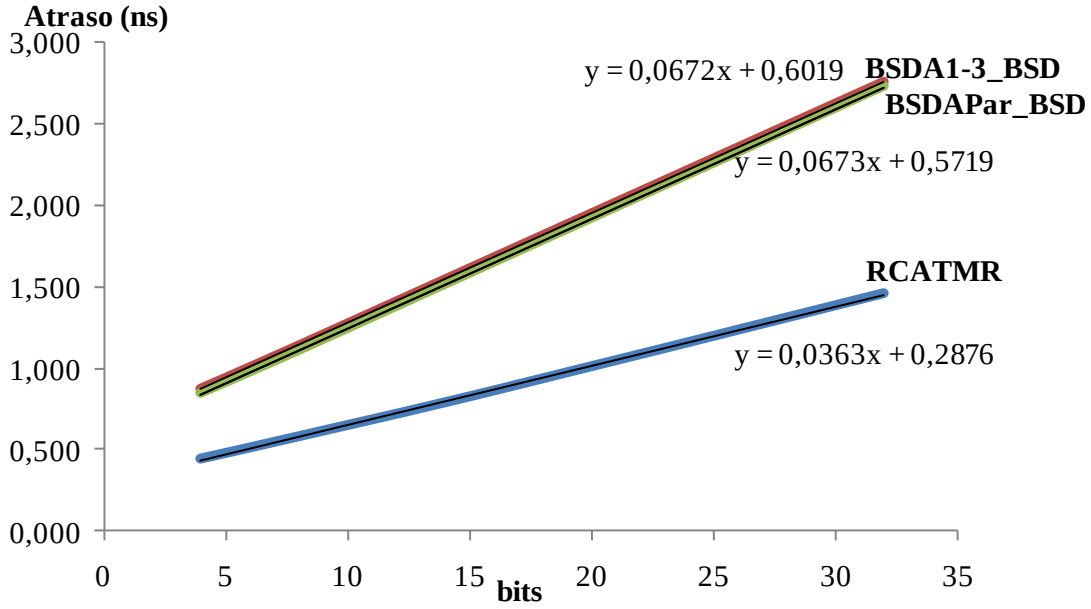


Figura C.14: Atraso crítico dos somadores RCATMR, BSDA1-3_BSD e BSDAPar_BSD

Também pode ser visto na Figura C.14, que o atraso crítico dos somadores BSDA1-3_BSD e BSDAPar_BSD fica consideravelmente maior que o atraso crítico do RCATMR. Com mencionando anteriormente, isso se deve ao uso dos conversores, pelos somadores do tipo BSDA_BSD.

A Tabela C.17 apresenta os resultados referentes ao atraso crítico do somador RCATMR e dos somadores BSDA1-3_BSD e BSDAPar_BSD. Também mostra os resultados da comparação entre as técnicas de proteção aplicadas ao somador BSDA_BSD com a técnica TMR aplicada ao RCA.

Tabela C.17: Comparação entre o atraso crítico do somador RCATMR e somadores BSDA1-3_BSD e BSDAPar_BSD

bits	RCATMR (1)	BSDA 1-3_BSD (2)	BSDAPar_ BSD (3)	Acréscimo [(2)/(1)-1*100]	Acréscimo [(3)/(1)-1*100]
4	0,441	0,873	0,845	98%	92%
8	0,577	1,137	1,107	97%	92%
16	0,858	1,677	1,648	96%	92%
32	1,455	2,754	2,728	89%	87%

Analisando-se a Tabela C.17 conclui-se que o BSDA1-3_BSD fica , em média, 95% mais lento do que o RCATMR. Já o BSDAPar_BSD fica, em média, 91% mais lento que o RCATMR.

No caso do RCATMR o atraso crítico é composto pelos atrasos de um RCA e do votador, que é constante independentemente do comprimento dos operandos de entrada das arquiteturas. Já o atraso dos somadores BSDA1-3_BSD e BSDAPar_BSD é composto pelo atraso dos blocos conversores de entrada e saída dos mesmos e pelo atraso do bloco que realiza a soma BSD. O atraso da soma BSD também é constante. Portanto, conforme o número de bits dos somadores aumenta, a diferença entre os atrasos críticos do BSDA1-3_BSD e BSDAPar_BSD em relação ao atraso crítico do

RCA diminui. Isso se deve ao fato do atraso inserido pelo votador e pela soma BSD tornarem-se menos importantes diante do acréscimo do atraso crítico para somadores de mais bits.

Assim, os somadores BSDA1-3_BSD e BSDAPar_BSD de 4 bits apresentaram a maior diferença em relação ao RCATMR de 4 bits, ficando, respecitavamente, 98% e 92% mais lentos. Já a menor diferença foi apresentada pelos somadores de 32 bits, onde os somadores BSDA1-3_BSD e BSDAPar_BSD, ficaram respectivamente, 89% e 87% mais lentos que o RCATMR.

O gráfico da Figura C.15 apresenta os resultados em termos de atraso crítico, gerados pela aplicação das técnicas de proteção TMR, codificação 1 de 3 e verificação de paridade, nos somadores RCA e BSDA_BSD, respectivamente, desconsiderando o atraso crítico dos blocos conversores dos somadores BSDA1-3_BSD e BSDAPar_BSD.

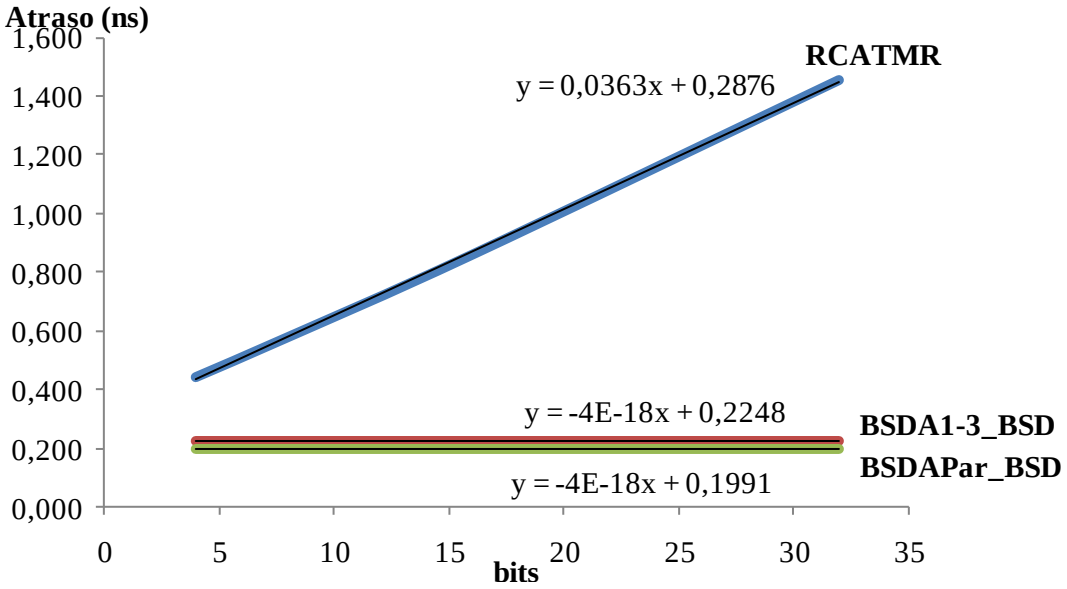


Figura C.15: Atraso crítico dos somadores RCATMR, BSDA1-3_BSD e BSDAPar_BSD desconsiderando as etapas de conversão dos BSDA_BSDs

Como pode ser visto na Figura C.15, o atraso da soma BSD utilizada nos somadores BSDA1-3_BSD e BSDAPar_BSD é constante para todos os comprimentos dos operandos de entrada dos somadores. Também pode ser observado que o atraso crítico da soma BSD utilizada pelo BSDA1-3_BSD é maior que o atraso da soma BSD utilizada pelo BSDAPar_BSD. Isso é devido as alterações realizadas no bloco que executa a soma BSD para a aplicação da Codificação 1 de 3, ao passo que o BSDAPar_BSD utiliza o mesmo bloco da soma BSD que o BSDA_BSD. Também pode ser visto que o atraso da soma BSD é consideravelmente menor que o atraso crítico do RCATMR.

A Tabela C.18 apresenta os resultados referentes ao atraso crítico do somador RCATMR e dos somadores BSDA1-3_BSD e BSDAPar_BSD, porém sem considerar o atraso inserido pelas etapas de conversão dos BSDA_BSDs. Também mostra os resultados da comparação entre as técnicas de proteção aplicadas ao BSDA_BSDs com a técnica TMR aplicada ao RCA.

Tabela C.18: Comparação entre o atraso crítico do somador RCATMR e somadores BSDA1-3_BSD e BSDAPar_BSD desconsiderando as etapas de conversão dos BSDA_BSDs

bits	RCATMR (1)	BSDA 1-3_BSD (2)	BSDAPar_ BSD (3)	Acréscimo [(1)/(2)-1*100]	Acréscimo [(1)/(3)-1*100]
4	0,441	0,225	0,199	96%	121%
8	0,577	0,225	0,199	157%	190%
16	0,858	0,225	0,199	282%	331%
32	1,455	0,225	0,199	547%	631%

Como pode ser observado na Tabela C.18, o RCATMR é , em média, 270% mais lento do que a soma BSD do BSDA1-3_BSD e, em média, 318% mais lento que do que a soma BSD do BSDAPar_BSD. Também pode ser visto que conforme o número de bits dos somadores aumenta, a diferença entre os atrasos críticos da soma BSD do BSDA1-3_BSD e da soma BSD do BSDAPar_BSD em relação ao atraso crítico do RCA aumenta. Isso é devido ao fato de o atraso crítico da soma BSD ser constante enquanto que o RCATMR apresenta a cadeia de *carry* que aumenta a medida que o número de bits dos somadores aumenta.

A Figura C.16 mostra as curvas do número de transistores dos somadores RCATMR, BSDA1-3_BSD e BSDAPar_BSD.

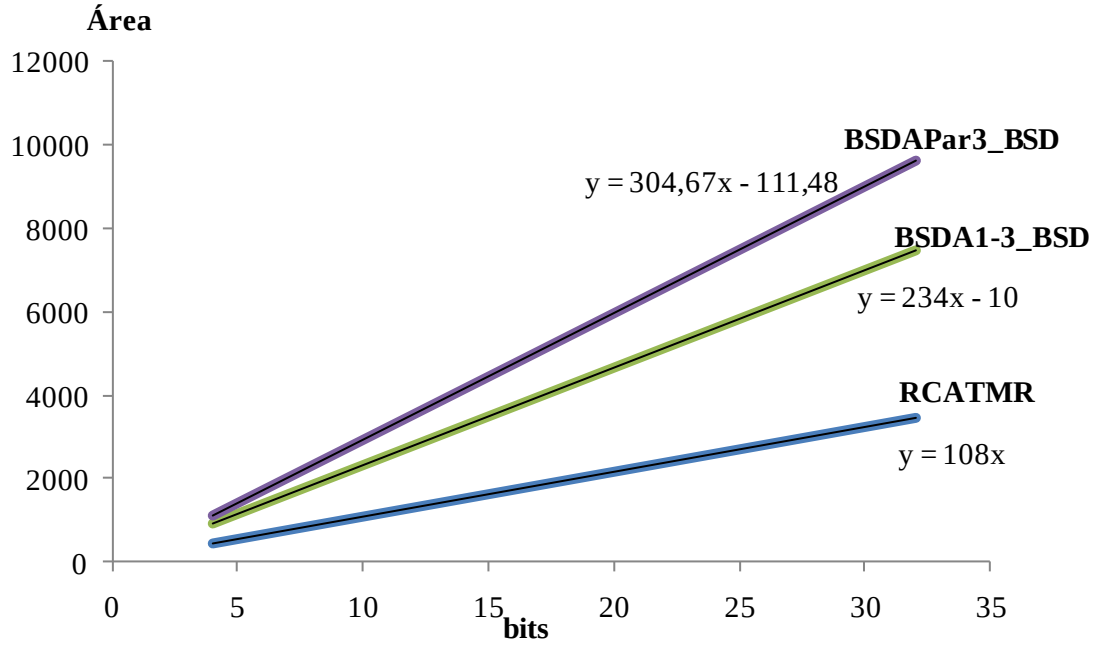


Figura C.16: Número de transistores dos somadores RCATMR, BSDA1-3_BSD e BSDAPar_BSD

A Figura C.16 mostra que apesar da técnica TMR triplicar o RCA, este utiliza menos transistores do que os somadores BSDA1-3_BSD e BSDAPar_BSD, isso é devido ao fato de que, além dos somadores BSDA1-3_BSD e BSDAPar_BSD utilizarem os conversores de entrada e saída, a aplicação da codificação 1 de 3 e verificação de paridade aumenta consideravelmente a quantidade de transistores necessários para a implementação das técnicas de proteção.

A Tabela C.19 apresenta os resultados referentes ao número de transistores do somador RCATMR e dos somadores BSDA1-3_BSD e BSDAPar_BSD. Também mostra os resultados da comparação entre as técnicas de proteção aplicadas ao BSDA_BSD com a técnica TMR aplicada ao RCA.

Tabela C.19: Comparação entre o número de transistores do somador RCATMR e somadores BSDA1-3_BSD e BSDAPar_BSD

bits	**RCATMR (1)**	**BSDA 1-3_BSD (2)**	**BSDAPar_ BSD (3)**	**Acréscimo [(2)/(1)-1*100]**	**Acréscimo [(3)/(1)-1*100]**
4	432	926	1110	114%	157%
8	864	1862	2326	116%	169%
16	1728	3734	4758	116%	175%
32	3456	7478	9640	116%	179%

Como pode ser visto na Tabela C.19, a diferença no número de transistores utilizados pelo BSDA1-3_BSD em relação ao RCATMR é praticamente constante com o aumento do número de bits dos somadores, ao passo que a diferença no número de transistores utilizados pelo BSDAPar_BSD em relação ao RCATMR aumenta conforme o número de bits dos somadores aumenta. Isso se deve ao fato de que conforme o número de bits do somador dobra, a quantidade de transistores utilizada pelas técnicas TMR e Codificação 1 de 3 praticamente duplica, ao passo que isso não ocorre no BSDAPar_BSD.

Assim, comparando a técnica TMR com a técnica de verificação de paridade, o BSDAPar_BSD de 4 bits apresenta a menor diferença em relação ao RCATMR de 4 bits, utilizando 157% mais transistores. Já a maior diferença é apresentada pelos somadores de 32 bits, onde o BSDAPar_BSD utiliza 179% mais transistores do que o RCATMR.

Além disso, a Tabela C.19 mostra que o BSDA1-3_BSD utiliza, em média, 116% mais transistores que o RCATMR. Já o BSDAPar_BSD utiliza, em média, 170% mais transistores que o RCATMR.

A Figura C.17 mostra o gráfico que apresenta a quantidade de transistores utilizados pelos somadores RCATMR, BSDA1-3_BSD e BSDAPar_BSD, desconsiderando os blocos dos conversores dos somadores BSDA1-3_BSD e BSDAPar_BSD.

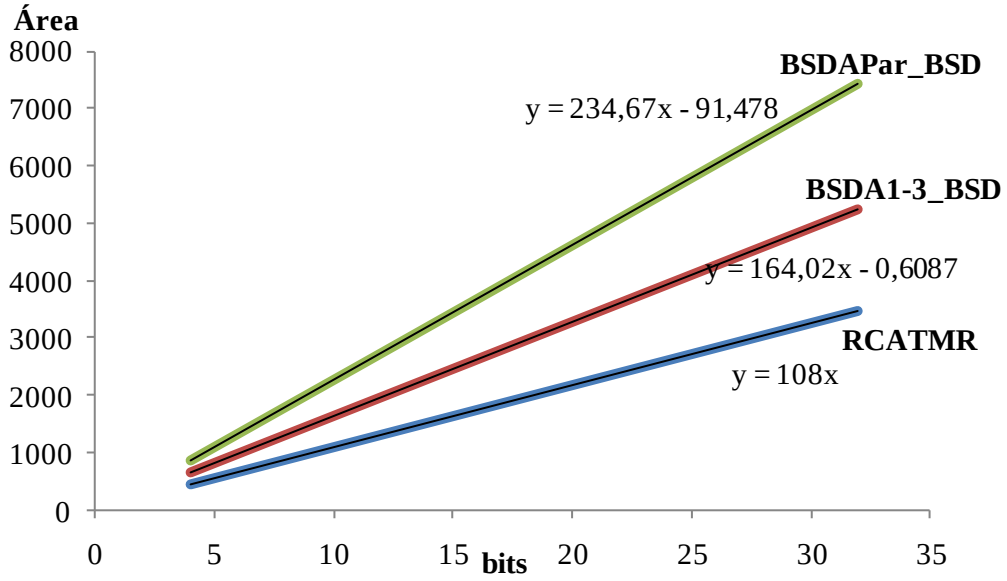


Figura C.17: Número de transistores dos somadores RCATMR, BSDA1-3_BSD e BSDAPar_BSD desconsiderando as etapas de conversão dos somadores BSDA1-3_BSD e BSDAPar_BSD

Pode ser visto na Figura C.17 que, mesmo sem considerar os transistores utilizados pelos blocos conversores, os somadores BSDA1-3_BSD e BSDAPar_BSD utilizam uma quantidade maior de transistores que o somador RCATMR. Isso se deve à lógica extra necessária para a implementação das técnicas de proteção e ao próprio circuito que implementa a soma BSD propriamente dita dos somadores BSDA1-3_BSD e BSDAPar_BSD.

A Tabela C.20 apresenta os resultados referentes ao número de transistores do somador RCATMR e dos somadores BSDA1-3_BSD e BSDAPar_BSD, sem considerar os transistores utilizados pelos blocos conversores dos mesmos. Também mostra os resultados da comparação entre as técnicas de proteção aplicadas ao BSDA com a técnica TMR aplicada ao RCA.

Tabela C.20: Comparação entre o número de transistores do somador RCATMR e somadores BSDA1-3_BSD e BSDAPar_BSD desconsiderando as etapas de conversão dos BSDA_BSDs

bits	RCATMR (1)	BSDA 1-3_BSD (2)	BSDAPar_ BSD (3)	Acréscimo [(2)/(1)-1*100]	Acréscimo [(3)/(1)-1*100]
4	432	655	850	52%	97%
8	864	1312	1786	52%	107%
16	1728	2624	3658	52%	112%
32	3456	5248	7420	52%	115%

Pode ser visto na Tabela C.20, que a soma BSD e o Verificador de Erros do BSDA1-3_BSD utilizam, em média, 52% mais transistores que o RCATMR. Já a soma BSD do BSDAPar_BSD e demais blocos necessários para implemantaçao da técnica de verificação de paridade, utilizam, em média, 107% mais transistores que o RCATMR.

Também pode ser visto que o acréscimo no número de transistores do BSDA1-3_BSD, em relação ao RCATMR, é constante visto que os circuitos que compõem o RCATMR e o BSDA1-3_BSD são praticamente regulares. Já o acréscimo no número de transistores utilizados pelo BSDAPar_BSD aumenta com o aumento do número de bits das arquiteturas,pois a lógica da técnica de verificação de paridade não é regular.

O gráfico apresentado na Figura C.18 mostra as curvas do consumo de potência dos somadores RCA TMR, BSDA1-3_BSD e BSDAPar_BSD

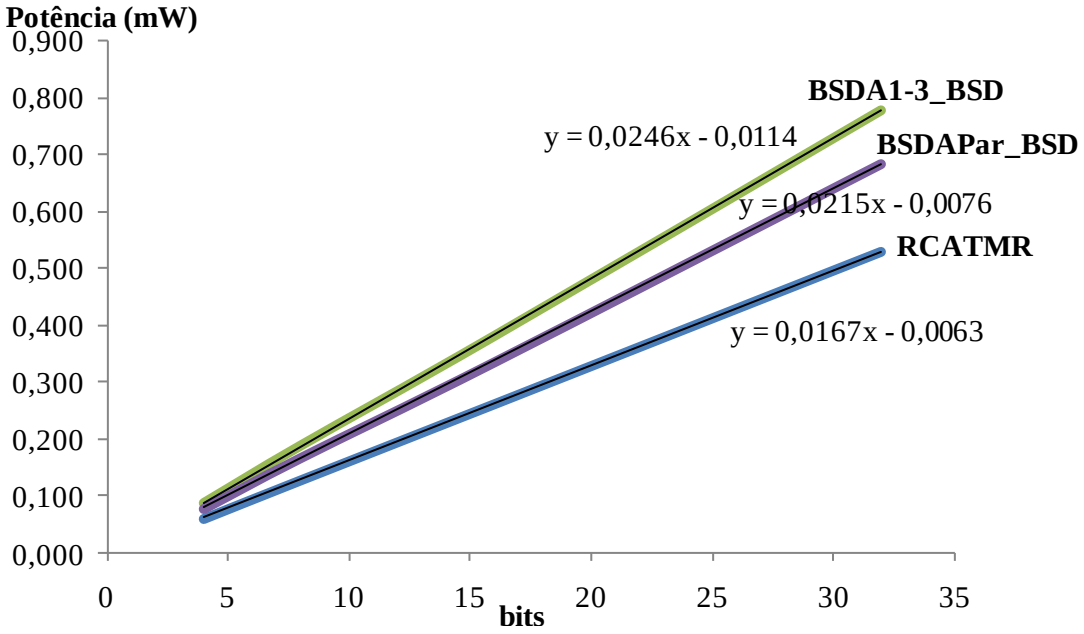


Figura C.18: Consumo de potência dos somadores RCA TMR, BSDA1-3_BSD e BSDAPar_BSD

A Figura C.18 mostra que os somadores BSDA1-3_BSD e BSDAPar_BSD apresentam um consumo de potência maior do que o somador RCA TMR, como era esperado, visto que estes utilizam uma maior quantidade de transistores e portanto, possuem uma maior atividade de chaveamento.

A Tabela C.21 apresenta os resultados referentes à potência consumida pelo RCATMR e pelos somadores BSD1-3_BSD e BSDAPar_BSD. Também mostra os resultados da comparação entre as técnicas de proteção aplicadas ao BSDA_BSD com a técnica TMR aplicada ao RCA.

Tabela C.21: Comparação entre o consumo de potência do somador RCATMR e somadores BSDA1-3_BSD e BSDAPar_BSD.

bits	**RCATMR (1)**	**BSDA 1-3_BSD (2)**	**BSDAPar_ BSD (3)**	**Acréscimo [(2)/(1)-1*100]**	**Acréscimo [(3)/(1)-1*100]**
4	0,060	0,087	0,078	44%	29%
8	0,128	0,188	0,167	47%	31%
16	0,261	0,380	0,335	46%	28%
32	0,528	0,778	0,682	47%	29%

Pode ser visto na Tabela C.21 que o BSDA1-3_BSD consome, em média, 46% mais potência que o RCATMR. Já o BSDAPar_BSD apresenta um consumo de potência, em média, 29% maior que o consumo de potência do RCATMR.

O gráfico apresentado na Figura C.19 mostra o consumo de potência gerado pela aplicação das técnicas de proteção TMR, codificação 1 de 3 e verificação de paridade, nos somadores RCA e BSDA_BSD, respectivamente, porém sem considerar a potência consumida pelos blocos conversores dos BSDA_BSDs.

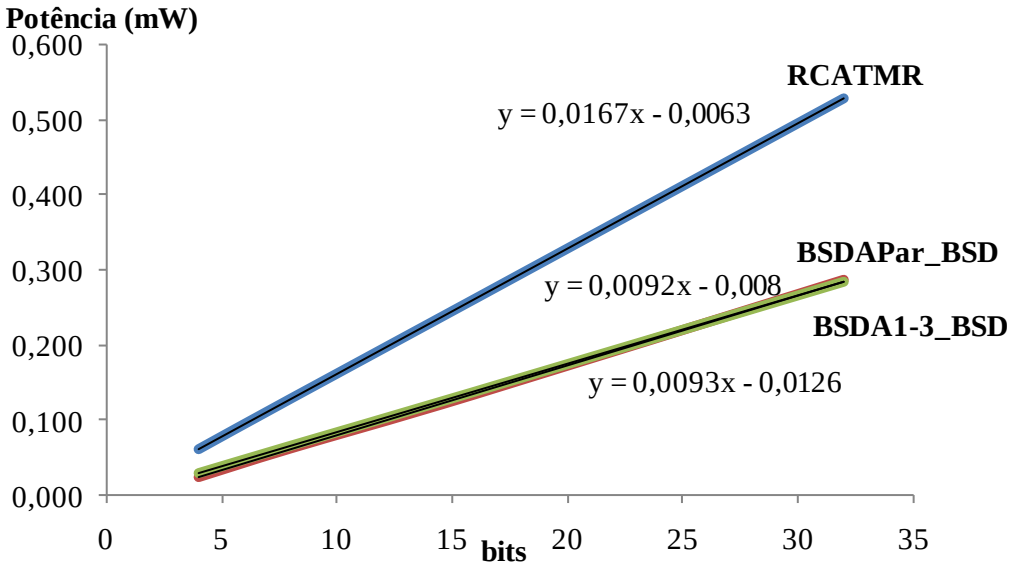


Figura C.19: Consumo de potência dos somadores RCATMR, BSDA1-3_BSD e BSDAPar_BSD desconsiderando as etapas de conversão dos BSDA_BSDs

Como pode ser visto na Figura C.19, a soma BSD juntamente com os blocos necessários para a implementação das técnicas de proteção dos BSDA_BSDs apresentou um consumo de potência inferior ao consumo de potência apresentado pelo somador RCATMR. Isso é resultado da utilização do sistema de representação redundante BSD NAF que reduz o consumo de potência dos blocos que implementam os BSDA_BSDs.

A Tabela C.22 apresenta os resultados referentes à potência consumida pelo RCATMR e pelos somadores BSDA1-3_BSD e BSDAPar_BSD, sem no entanto considerar o consumo de potência dos blocos conversores dos BSDA_BSDs. Também mostra os resultados da comparação entre as técnicas de proteção aplicadas ao BSDA_BSD com a técnica TMR aplicada ao RCA.

Tabela C.22: Comparação entre o consumo de potência do somador RCATMR e somadores BSDA1-3_BSD e BSDAPar_BSD desconsiderando as etapas de conversão dos BSDA_BSDs

bits	RCATMR (1)	BSDA 1-3_BSD (2)	BSDAPar_ BSD (3)	Acréscimo [(1)/(2)-1*100]	Acréscimo [(1)/(3)-1*100]
4	0,060	0,025	0,029	143%	110%
8	0,128	0,063	0,065	104%	95%
16	0,261	0,134	0,138	94%	89%
32	0,528	0,285	0,285	85%	85%

Pode ser visto na Tabela C.22 que o RCATMR consome, em média, 107% mais potência que a soma BSD e o Verificador de Erros do BSDA1-3_BSD. Já em relação à soma BSD e os blocos de verificação de paridade do BSDAPar_BSD, o RCATMR apresenta um consumo de potência, em média, 95% maior. Também pode ser visto que, com o aumento do número de bits dos somadores, a diferença entre o consumo de potência dos blocos dos BSDA_BSDs e o RCATMR diminui.

A Figura C.20 mostra o gráfico que apresenta os resultados de atraso crítico, gerados pela aplicação das técnicas de proteção RESI, codificação 1 de 3 e verificação de paridade, nos somadores RCA e BSDA_BSD, respectivamente.

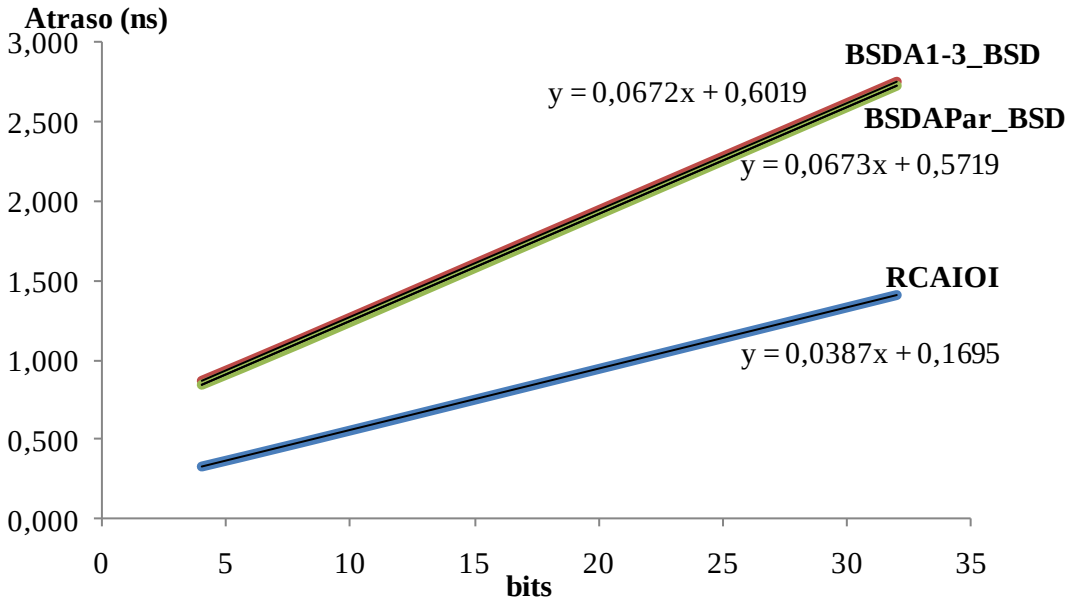


Figura C.20: Atraso crítico dos somadores RCAIOI, BSDA1-3_BSD e BSDAPar_BSD

Pode ser visto na Figura C.20 que os somadores BSDA1-3_BSD e BSDAPar_BSD apresentam um atraso crítico consideravelmente maior que o RCAIOI. Como mencionado anteriormente, isso se deve ao fato do uso de conversores de entrada e saída de ambos.

A Tabela C.23 concentra os resultados referentes ao atraso crítico do RCAIOI e dos somadores BSDA1-3_BSD e BSDAPar_BSD. Também mostra os resultados da comparação entre as técnicas de proteção aplicadas ao BSDA_BSD com a técnica RESI aplicada ao RCA.

Tabela C.23: Comparação entre o atraso crítico do RCAIOI e somadores BSDA1-3_BSD e BSDAPar_BSD

bits	RCAIOI (1)	BSDA 1-3_BSD (2)	BSDAPar_ BSD (3)	Acréscimo [(2)/(1)-1*100]	Acréscimo [(3)/(1)-1*100]
4	0,326	0,873	0,845	168%	159%
8	0,477	1,137	1,107	138%	132%
16	0,786	1,677	1,648	113%	110%
32	1,407	2,754	2,728	96%	94%

Pode ser observado na Tabela C.23 que o BSDA1-3_BSD é, em média, 129% mais lento que RCAIOI. Já o BSDAPar_BSD apresenta um atraso crítico,em média, 124% maior que o atraso crítico do RCAIOI.

Também pode ser visto que, com o aumento do número de bits das arquiteturas, a diferença entre o atraso crítico dos BSDA_BSDs e o atraso crítico do RCAIOI diminui. Isso se deve ao fato do atraso inserido pelos blocos que executam a soma BSD nos somadores BSDA1-3_BSD e BSDAPar_BSD e pelos blocos extras do RCAIOI tornar-se menos importante diante do acréscimo do atraso crítico para somadores de mais bits, onde a cadeia de *carry* dos conversores dos somadores BSDA1-3_BSD e BSDAPar_BSD e do RCA do RCAIOI torna-se mais significativa para o acréscimo do atraso crítico.

Assim a maior diferença entre os atrasos é apresentada pelas arquiteturas de 4bits, sendo que o BSDA1-3_BSD é 168% mais lento e o BSDAPar_BSD é 159% mais lento que o RCAIOI. A menor diferença é apresentada pelas arquiteturas de 32 bits onde o BSDA1-3_BSD possui o atraso 96% maior e o BSDAPar_BSD possui o atraso crítico 94% maior que o RCAIOI.

A Figura C.21 mostra o gráfico que apresenta os resultados de atraso crítico do RCAIOI e dos somadores BSDA1-3_BSD e BSDAPar_BSD, porém sem considerar o atraso crítico dos blocos conversores.

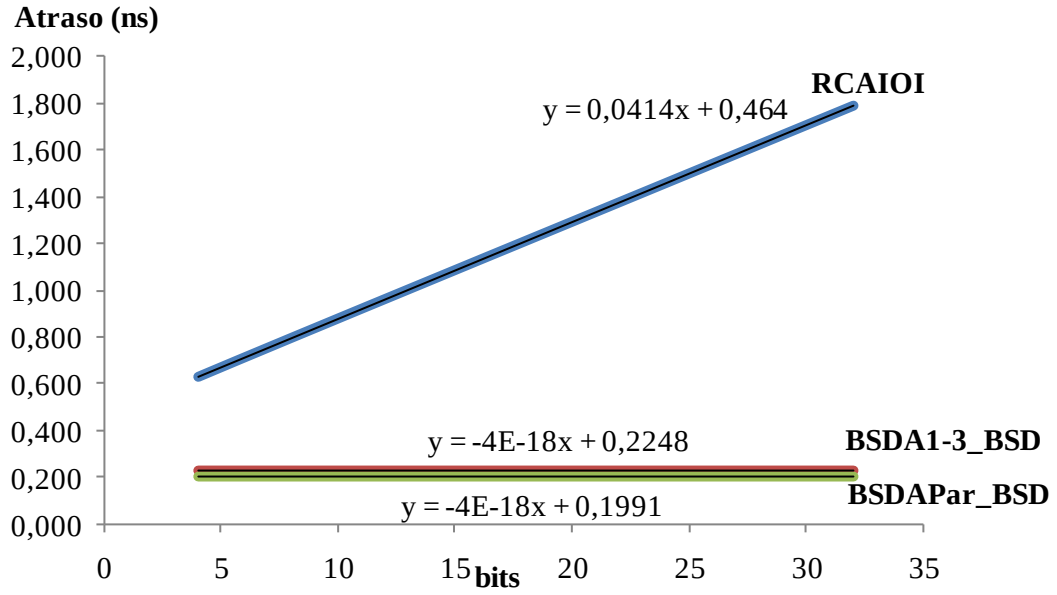


Figura C.21: Atraso crítico dos somadores RCAIO, BSDA1-3_BSD e BSDAPar_BSD desconsiderando as etapas de conversão

Pode ser observado no gráfico da Figura C.21, que sem considerar as etapas de conversão dos somadores BSDA1-3_BSD e BSDAPar_BSD, o atraso dos mesmos é consideravelmente menor que o atraso crítico do RCAIOI. E como mencionado anteriormente, o atraso crítico da soma BSD dos somadores BSDA1-3_BSD e BSDAPar_BSD é constante para todos os comprimentos dos operandos de entrada das arquiteturas.

A Tabela C.24 evidencia os resultados referentes ao atraso crítico do RCAIOI e dos somadores BSDA1-3_BSD e BSDAPar_BSD, sem considerar o atraso crítico dos

blocos conversores. Também mostra os resultados da comparação entre as técnicas de proteção aplicadas ao somador BSD com a técnica RESI aplicada ao RCA.

Tabela C.24: Comparação entre o atraso crítico do RCAIOI e somadores BSDA1-3_BSD e BSDAPar_BSD desconsiderando as etapas de conversão

bits	RCAIOI (1)	BSDA 1-3_BSD (2)	BSDAPar_ BSD (3)	Acréscimo [(1)/(2)-1*100]	Acréscimo [(1)/(3)-1*100]
4	0,326	0,225	0,199	180%	216%
8	0,477	0,225	0,199	254%	300%
16	0,786	0,225	0,199	402%	467%
32	1,407	0,225	0,199	696%	799%

A Tabela C.24 mostra que sem considerar as etapas de conversão dos somadores BSDA1-3_BSD e BSDAPar_BSD o RCAIOI apresenta um atraso maior que tais somadores, sendo em média, 383% e 445% maior que o atraso do BSDA1-3_BSD e do BSDAPar_BSD, respectivamente. Também pode-se observar que com o aumento do número de bits a diferença entre os atrasos críticos do RCAIOI, BSDA1-3_BSD e do BSDAPar_BSD aumenta consideravelmente, pois o atraso crítico destes é constante, ao passo que o atraso do RCAIOI aumenta conforme o aumento do número de bits dos somadores.

A Figura C.22 apresenta as curvas do número de transistores utilizados pelos somadores RCAIOI, BSDA1-3_BSD e BSDAPar_BSD.

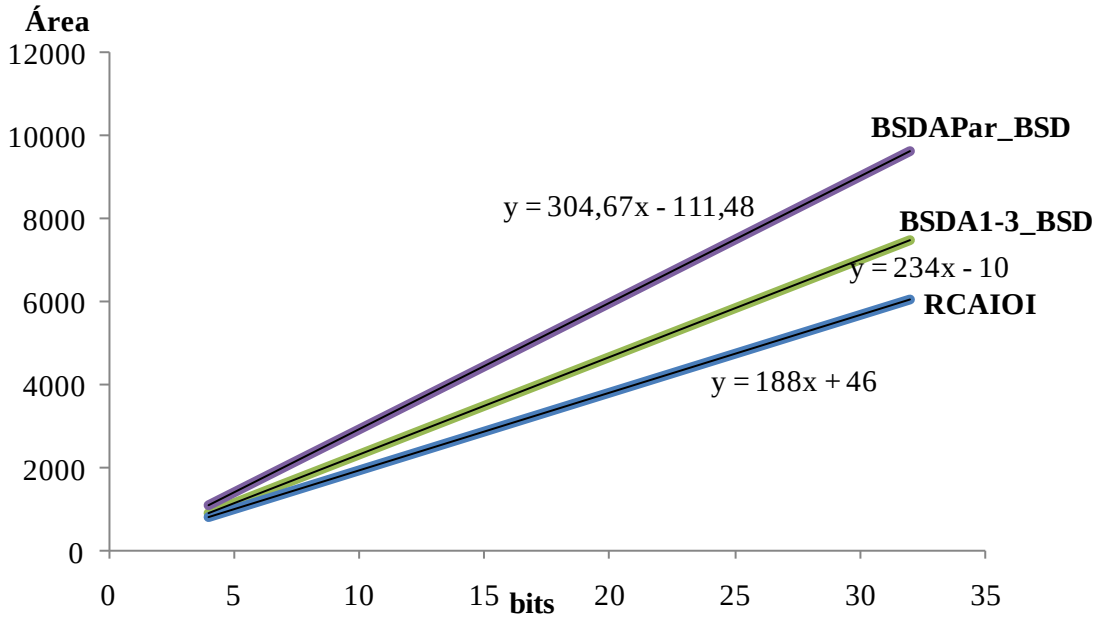


Figura C.22: Número de transistores dos somadores RCAIOI, BSDA1-3_BSD e BSDAPar_BSD.

O gráfico da Figura C.22 mostra que a quantidade de transistores utilizados pelo RCAIOI não é tão inferior à quantidade de transistores utilizados pelos somadores BSDA1-3_BSD e BSDAPar_BSD apesar destes utilizarem os conversores de entrada e saída. Isso se deve ao fato de que o RCAIOI utiliza redundância de *hardware* para a obtenção dos *carries* redundantes, blocos para a verificação de paridade, registrador e multiplexadores de entrada e saída.

A Tabela C.25 apresenta os resultados referentes ao número de transistores do RCAIOI e dos somadores BSDA1-3_BSD e BSDAPar_BSD, bem como os resultados da comparação entre as técnicas de proteção, tomando como referência RCAIOI.

Tabela C.25: Comparação entre o número de transistores do RCAIOI e somadores BSDA1-3_BSD e BSDAPar_BSD

bits	RCAIOI (1)	BSDA 1-3_BSD (2)	BSDAPar_ BSD (3)	Acréscimo [(2)/(1)-1*100]	Acréscimo [(3)/(1)-1*100]
4	798	926	1110	16%	39%
8	1550	1862	2326	20%	50%
16	3054	3734	4758	22%	56%
32	6062	7478	9640	23%	59%

Como pode ser observado na Tabela C.25, com o aumento do número de bits dos somadores a diferença entre o número de transistores utilizados pelos BSDA_BSDs e o número de transistores utilizados pelo RCAIOI aumenta. Assim a menor diferença entre o número de transistores dos somadores BSDA1-3_BSD e BSDAPar_BSD e RCAIOI é apresentado pelas arquiteturas de 4 bits, onde o BSDA1-3_BSD utiliza 16% mais transistores e o BSDAPar_BSD é 39% maior que o RCAIOI. Já a maior diferença entre o número de transistores dos BSDA_BSDs e RCAIOI é apresentada pelas arquiteturas de 32 bits, sendo que o BSDA1-3_BSD possui um acréscimo de 23% e o BSDAPar_BSD apresenta um acréscimo de 59% no número de transistores em relação ao RCAIOI. Também pode ser visto que o BSDA1-3_BSD utiliza, em média, 20% mais transistores que o RCAIOI. Já o BSDAPar_BSD, em média, 51% mais transistores que o RCAIOI.

O gráfico da Figura C.23 apresenta os resultados referentes ao número de transistores utilizados pelos somadores RCAIOI e BSDA1-3_BSD e BSDAPar_BSD, sem considerar as etapas de conversão dos BSDA_BSDs.

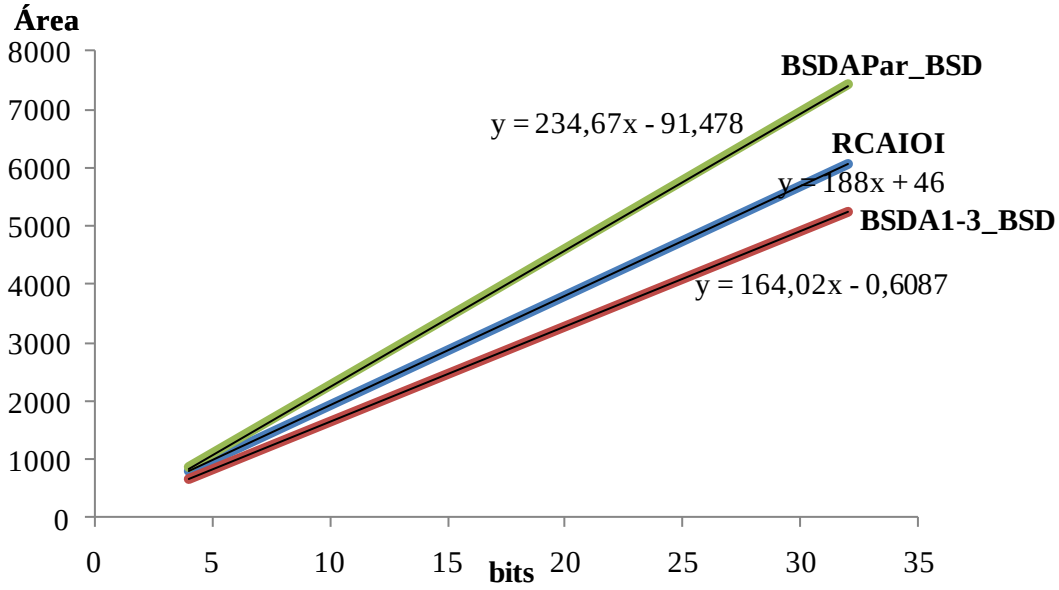


Figura C.23: Número de transistores dos somadores RCAIO, BSDA1-3_BSD e BSDAPar_BSD desconsiderando as etapas de conversão dos BSDA_BSD

Pode ser visto na Figura C.23, que sem considerar as etapas de conversão dos BSDA1-3_BSD e BSDAPar_BSD, o BSDA1-3_BSD utiliza menos transistores que o RCAIOI, ao passo que o BSDAPar_BSD utiliza mais transistores do que os demais. Isso se deve ao fato da técnica de verificação de paridade utilizar portas CMOS XORs para implementar os circuitos que verificam a paridade, e estas utilizam mais transistores que as portas utilizadas pelos demais somadores.

A Tabela C.26 apresenta os resultados referentes ao número de transistores do RCAIOI e dos somadores BSDA1-3_BSD e BSDAPar_BSD, porém desconsiderando as etapas de conversão destes. Também mostra os resultados da comparação entre as técnicas de proteção, tomando como referência o RCAIOI.

Tabela C.26: Comparação entre o número de transistores do RCAIOI e somadores BSDA1-3_BSD e BSDAPar_BSD desconsiderando as etapas de conversão dos BSDA_BSDs

bits	RCAIOI (1)	BSDA 1-3_BSD (2)	BSDAPar_ BSD (3)	Acréscimo [(1)/(2)-1*100]	Acréscimo [(1)/(3)-1*100]
4	798	656	850	22%	7%
8	1550	1312	1786	18%	15%
16	3054	2624	3658	16%	20%
32	6062	5248	7420	16%	22%

Pode ser visto na Tabela C.26 que, desconsiderando as etapas de conversão dos BSDA_BSDs, o RCAIOI utilizou, em média, 18% mais transistores que o BSDA1-3_BSD e, em média, 16% mais transistores que o BSDAPar_BSD. Além disso, é possível observar que a diferença entre o número de transistores utilizados pelo BSDAPar_BSD e pelo RCAIOI aumenta com o aumento do comprimento dos somadores devido a irregularidade da técnica de verificação de paridade.

O gráfico apresentado na Figura C.24 apresenta os resultados de consumo de potência somadores RCAIOI, BSDA1-3_BSD e BSDAPar_BSD.

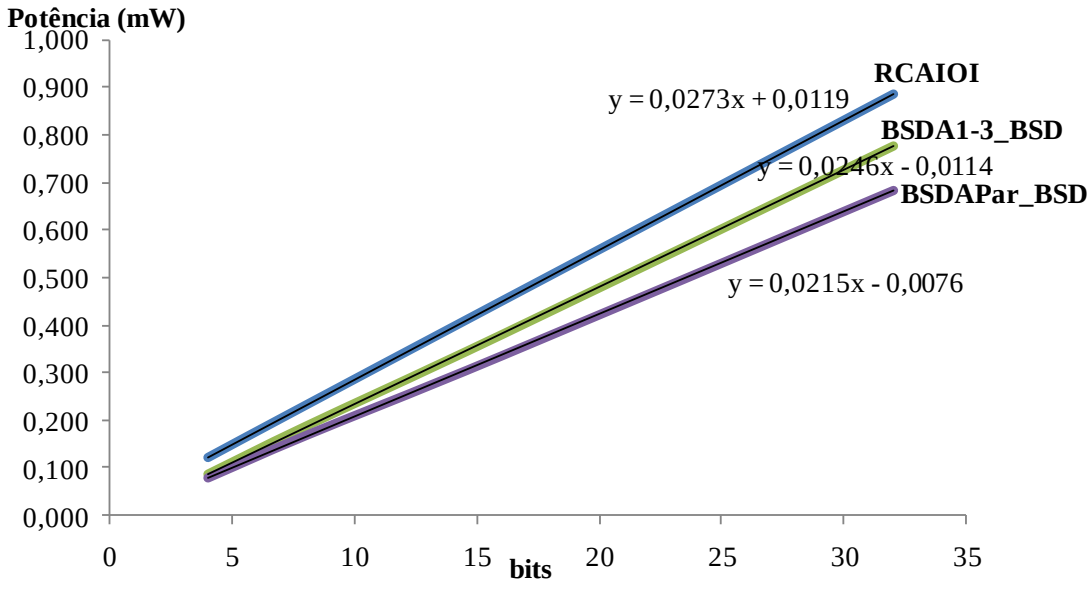


Figura C.24: Consumo de potência dos somadores RCAIOI, BSDA1-3_BSD e BSDAPar_BSD

A Figura C.24 mostra o RCAIOI apresenta uma maior consumo de potência que os BSDA_BSDs apesar destes consumirem uma quantidade considerável de potência devido às etapas de conversão. Isso é devido à lógica extra utilizada para implementar a técnica RESI.

A Tabela C.27 apresenta os resultados referentes a potência consumida pelo RCAIOI e pelos somadores BSDA1-3_BSD e BSDAPar_BSD. Também mostra os resultados da comparação entre as técnicas de proteção, tomado o RCAIOI como referência.

Tabela C.27: Comparação entre o consumo de potência do RCAIOI e somadores BSDA1-3_BSD e BSDAPar_BSD

bits	RCAIOI (1)	BSDA 1-3_BSD (2)	BSDAPar_ BSD (3)	Acréscimo [(2)/(1)-1*100]	Acréscimo [(3)/(1)-1*100]
4	0,120	0,087	0,078	39%	54%
8	0,230	0,188	0,167	23%	38%
16	0,449	0,380	0,335	18%	34%
32	0,884	0,778	0,682	14%	30%

Como pode ser observado na Tabela C.27, o RCAIOI consome, em média, 23% e 39% mais potência que o BSDA1-3_BSD e o BSDAPar_BSD, respectivamente. Também pode ser visto que a diferença entre os consumos de potência do RCAIOI e os somadores BSDA1-3_BSD e BSDAPar_BSD diminui com o aumento do número de bits das arquiteturas.

Desta forma a maior diferença entre os consumos de potência foi apresentada pelas arquiteturas de 4 bits, onde o RCAIOI consome, 39% e 54% mais potência que o BSDA1-3_BSD e que o BSDAPar_BSD, respectivamente. Já a menor diferença é apresentada pelos somadores de 32 bits, sendo que o RCAIOI consome, 14% e 30% mais potência que o BSDA1-3_BSD e que o BSDAPar_BSD, respectivamente.

A Figura C.25 mostra o gráfico que apresenta os resultados de consumo de potência dos somadores RCAIOI, BSDA1-3_BSD e BSDAPar_BSD, sem no entanto considerar as etapas de conversão destes.

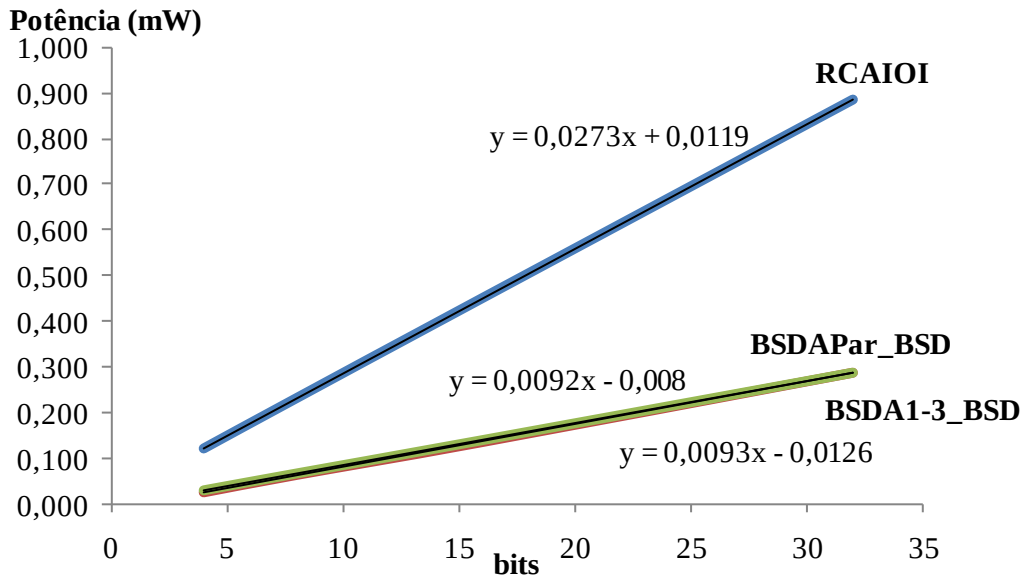


Figura C.25: Consumo de potência dos somadores RCAIOI, BSDA1-3_BSD e BSDAPar_BSD desconsiderando as etapas de conversão dos BSDA_BSDs.

A Tabela C.28 apresenta os resultados referentes a potência consumida pelo somador RCAIOI e pelos somadores BSDA1-3_BSD e BSDAPar_BSD, sem considerar as etapas de conversão dos mesmos. Também mostra os resultados da comparação entre as técnicas de proteção, tomado o RCAIOI como referência.

Tabela C.28: Comparação entre consumo de potência do RCAIOI e somadores BSDA1-3_BSD e BSDPar_BSD desconsiderando as etapas de conversão dos somadores BSDA1-3_BSD e BSDAPar_BSD

bits	RCAIOI (1)	BSDA 1-3_BSD (2)	BSDAPar_ BSD (3)	Acréscimo [(1)/(2)-1*100]	Acréscimo [(1)/(3)-1*100]
4	0,120	0,025	0,029	385%	319%
8	0,230	0,063	0,065	268%	252%
16	0,449	0,134	0,138	235%	226%
32	0,884	0,285	0,285	210%	210%

Pode ser visto na Tabela C.28, que sem considerar as etapas de conversão dos somadores BSDA1-3_BSD e BSDAPar_BSD, o RCAIOI consome, em média, 274% e 252% mais potência que o BSDA1-3_BSD e que o BSDAPar_BSD, respectivamente.

Também pode ser visto que, conforme o número de bits dos somadores aumenta a diferença entre o consumo de potência do RCAIOI e o consumo dos BSDA_BSDs diminui. Assim a maior diferença entre os consumos de potência foi apresentada pelas arquiteturas de 4 bits, onde o RCAIOI consome 385% e 319% mais potência que os somadores BSDA1-3_BSD e BSDAPar_BSD, respectivamente. Já a menor diferença entre os consumos de potência foi apresentada pelas arquiteturas de 32 bits, sendo que o RCAIOI consome 210% mais potência, tanto no caso do BSDA1-3_BSD como no caso do BSDAPar_BSD.

C.5 Comparação entre somadores não-protegidos: RCA x BSDA_BSD x BSDA_C2

O gráfico apresentado na Figura C.26 mostra os resultados de atraso crítico para as arquiteturas não protegidas implementadas neste trabalho.

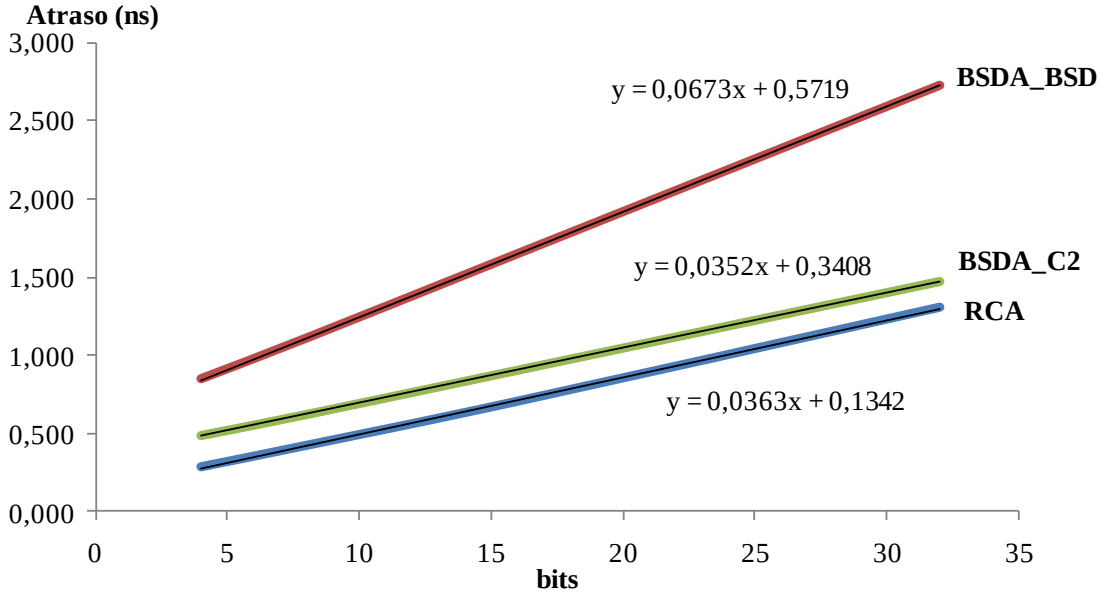


Figura C.26: Atraso crítico dos somadores RCA, BSDA_BSD e BSDA_C2

Como pode ser visto na Figura C.26, o somador RCA apresenta um atraso menor que o somador BSDA_C2, apesar deste também ser um somador *carry-free* e não utilizar conversor de entrada para os operandos. No entanto o BSDA_C2 ainda necessita converter o resultado da soma gerada em BSD para complemento de dois.

Por outro lado, o BSDA_C2 apresenta um atraso consideravelmente menor que o BSDA_BSD, pois este necessita de dois estágios de conversão, isto é, conversores de entrada e conversor de saída, ao passo que o BSDA_C2 possui apenas o conversor de saída.

Como pode ser visto na Tabela C.29, a medida que o número de bits aumenta a diferença entre o atraso crítico do RCA e do BSDA_C2 diminui. Isso ocorre porque com o aumento do número de bits dos somadores, o atraso do bloco que executa a soma BSD do BSDA_C2 tornar-se insignificante diante do atraso do conversor de saída do mesmo. Assim o atraso crítico do BSDA_C2 de 4 bits apresenta a maior diferença em relação ao RCA de 4 bits, sendo 69% maior e os somadores de 32 bits apresentam a menor diferença, ficando o atraso crítico do BSDA_C2 apenas 13% maior que o RCA.

Tabela C.29: Comparação do atraso crítico (em ns) dos somadores RCA, BSDA_BSD e BSDA_C2 não-protegidos.

bits	BSDA_C2 (1)	RCA (2)	BSDA_BSD (3)	[(1)/(2)-1*100]	[(3)/(1)-1*100]%
4	0,485	0,287	0,845	69%	74%
8	0,620	0,423	1,107	46%	79%
16	0,903	0,704	1,648	28%	83%
32	1,469	1,302	2,728	13%	86%

No entanto quando os somadores BSDA_BSD e BSDA_C2 são comparado, conforme o número de bits aumenta, a diferença entre o atraso crítico de ambos também aumenta. Isso ocorre devido a etapa de conversão de entrada do BSDA_BSD e além disto o bloco que executa a soma BSD deste somador apresenta um atraso mais significativo que o bloco da soma BSD do BSDA_C2. Desta forma, o BSDA_BSD de 4

bits apresenta a menor diferença, sendo 74% maior que o BSDA_C2 de 4 bits. Já o BSDA_BSD de 32 bits apresenta a maior diferença, sendo 86% maior que o BSDA_C2 de 32 bits.

Também pode ser visto na Tabela C.29 que o BSDA_C2 apresenta um atraso crítico, em média, 39% maior que o atraso crítico do RCA. Já o BSDA_BSD apresenta um atraso crítico, em média, 80% maior que o BSDA_C2.

O gráfico da Figura C.27 apresenta o número de transistores utilizados na implementação dos somadores RCA, BSDA_BSD e BSDA_C2 não-protegidos.

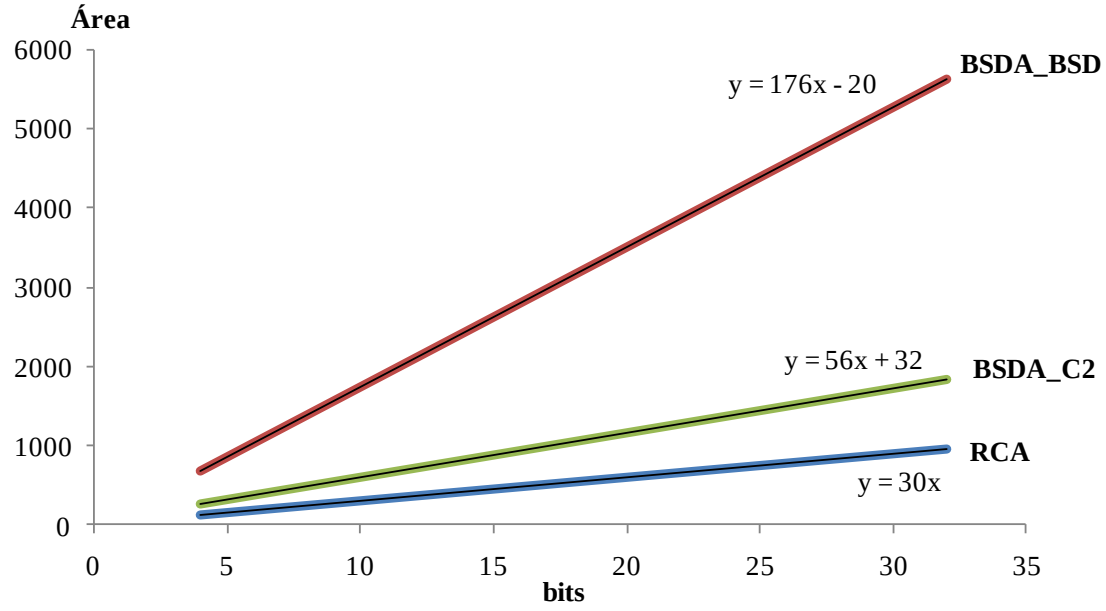


Figura C.27: Número de transistores utilizados pelos somadores RCA, BSDA_BSD e BSDA_C2 não protegidos

A Figura C.27 mostra que o BSDA_C2 utilizou mais transistores que o RCA, pois além de utilizar o somador de saída, que utiliza praticamente a mesma quantidade de transistores que o RCA, ainda possui o bloco que calcula a soma propriamente dita.

Em contrapartida, o BSDA_C2 utiliza uma quantidade significativamente menor de transistores que o BSDA_BSD, pois além de não necessitar de conversor de entrada o bloco que calcula a soma propriamente dita utiliza menos transistores que o BSDA_BSD.

Na Tabela C.30 pode ser observado que conforme o número de bits dos somadores aumenta, a diferença entre a utilização de transistores dos somadores BSDA_C2 e RCA diminui, desta forma o BSDA_C2 de 4 bits utiliza 113% mais transistores do que o RCA de 4 bits e o BSDA_C2 de 32 bits utiliza 90% mais transistores que o RCA de 32 bits.

Quando o somador analisado é o BSDA_BSD, com o aumento do número de bits do somador, o comportamento é contrário, ou seja, aumenta a diferença entre o número de transistores utilizados pelo BSDA_C2 e BSDA_BSD. Assim o somador BSDA_BSD de 4 bits utilizou 167% mais transistores que o BSDA_C2 de 4 bits, ao passo que a maior diferença é apresentada pelos somadores de 32 bits, sendo que o BSDA_BSD utiliza 208% mais transistores que o BSDA_C2.

Tabela C.30: Comparação do número de transistores dos somadores RCA, BSDA_BSD e BSDA_C2 não-protegidos.

bits	BSDA_C2 (1)	RCA (2)	BSDA_BSD (3)	[(1)/(2)-1*100]	[(3)/(1)-1*100]
4	256	120	684	113%	167%
8	480	240	1388	100%	189%
16	928	480	2796	93%	201%
32	1824	960	5612	90%	208%

Também pode ser observado na Tabela C.30 que o BSDA_C2 utiliza, em média, 99% mais transistores que o RCA, ao passo que o BSDA_BSD utiliza, em média, 191% mais transistores que o BSDA_C2.

O gráfico da Figura C.28 apresenta a potência consumida pelos somadores RCA, BSDA_BSD e BSDA_C2 não-protegidos.

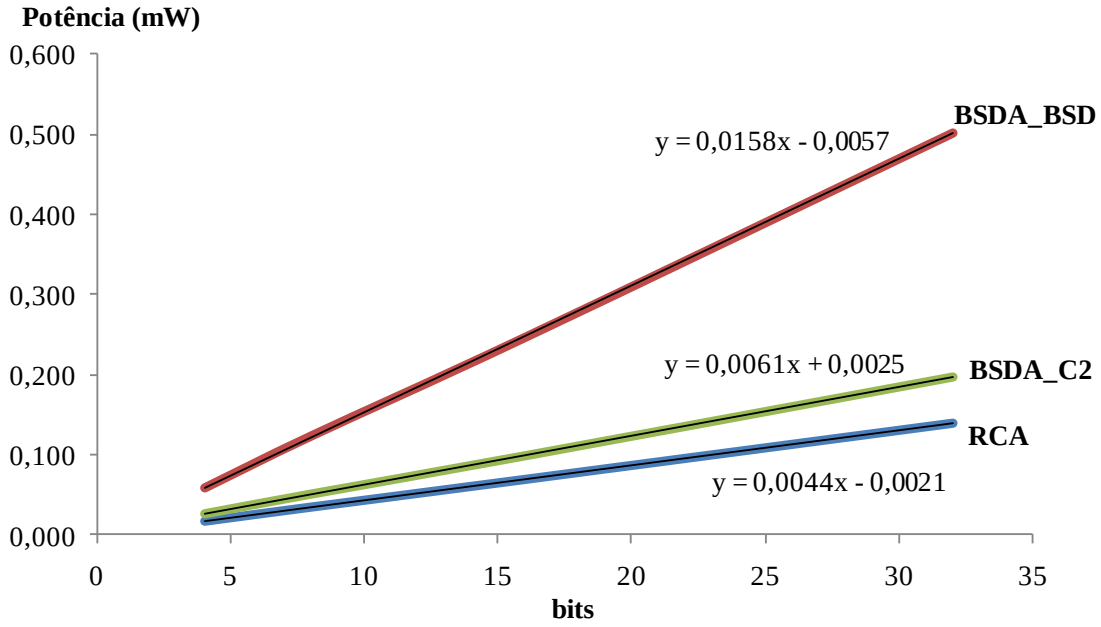


Figura C.28: Potência consumida pelos somadores RCA, BSDA_BSD e BSDA_C2

Como pode ser observado na Figura C.28 o BSDA_BSD consome mais potência que os demais somadores devido às etapas de conversão. Já o somador BSDA_C2 consome mais potência que o RCA visto que além de possuir um RCA para implementar a etapa de conversão de saída também possui a soma BSD propriamente dita.

Como pode ser visto na Tabela C.31, a medida que o número de bits aumenta a diferença entre o consumo de potência do RCA e do BSDA_C2 diminui. Assim o consumo do BSDAC2 de 4 bits apresenta a maior diferença em relação ao RCA de 4 bits, sendo 72% maior e os somadores de 32 bits apresentam a menor diferença, ficando o atraso crítico do BSDA_C2 apenas 42% maior que o RCA.

No caso do BSDA_BSD, com o aumento do número de bits dos somadores, aumenta a diferença entre a potência consumida pelos BSDA_C2 e BSDA_BSD. Assim o somador BSDA_BSD de 4 bits consome 115% mais potência que o BSDA_C2 de 4

bits, ao passo que a maior diferença é apresentada pelos somadores de 32 bits, sendo que o BSDA_BSD consome 155% mais potência que o BSDA_C2.

Tabela C.31: Comparação da potência consumida pelos somadores RCA, BSDA_BSD e BSDA_C2 não-protegidos.

bits	BSDA_C2 (1)	RCA (2)	BSDA_BSD (3)	[(1)/(2)-1*100]	[(3)/(1)-1*100]
4	0,027	0,016	0,057	72%%	115%
8	0,051	0,033	0,123	54%%	139%
16	0,099	0,068	0,246	45%%	149%
32	0,197	0,139	0,501	42%%	155%

Também pode ser observado na Tabela C.31 que o BSDA_C2 consome, em média, 53% mais potência que o RCA, ao passo que o BSDA_BSD consome, em média, 139% mais potência que o BSDA_C2.

A análise entre os somadores BSDA_BSD e RCA foi omitida por já ter sido apresentada na seção 4.1.

C.6 Comparação entre BSDAs com operandos em complemento de dois (BSDA_C2, BSDA1-3_C2 e BSDAPar_C2)

O gráfico da Figura C.29 apresenta o atraso crítico dos somadores BSDA_C2.

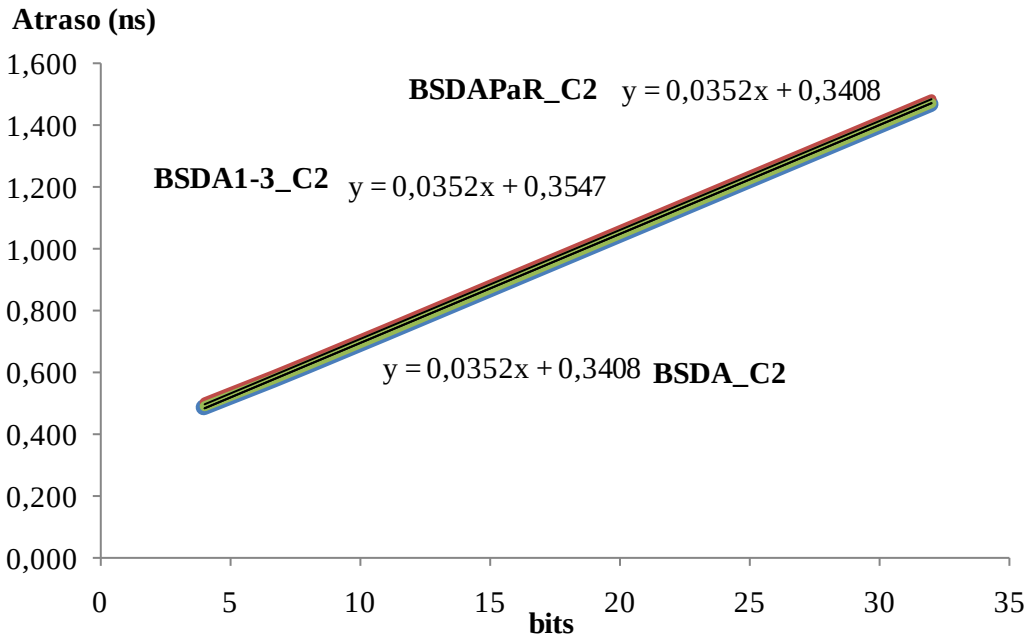


Figura C.29: Atraso crítico dos somadores BSDA_C2

Como pode ser visto na Figura C.29, os somadores BSDA_C2 e BSDAPar_C2 apresentam o mesmo atraso visto que a técnica de proteção aplicada ao BSDAPar não insere atraso ao somador por ser executada em paralelo ao cálculo da soma. Já a técnica utilizada no BSDA1-3_C2 insere um aumento no atraso do somador devido à utilização de mais hardware para implementação da técnica.

A Tabela C.32 mostra que o aumento no atraso do BSDA1-3_C2 em relação ao BSDA_C2 é mínimo ficando, em média, 2% maior. Ainda pode ser observado que com

o aumento do número de bits dos somadores, esta diferença entre os atrasos fica cada vez menor, pois o atraso inserido pela técnica de proteção com codificação 1 de 3 passa a tornar-se insignificante diante do atraso do conversor de saída dos somadores BSDA_C2.

Tabela C.32: Comparação do atraso crítico (em ns) dos somadores BSDA_C2

bits	BSDA_C2 (1)	BSDA1-3_C2 (2)	BSDAPar_C2 (3)	Acréscimo [(2)/(1)-1*100]	Acréscimo [(3)/(1)-1*100]
4	0,485	0,499	0,485	3%	0%
8	0,620	0,634	0,620	2%	0%
16	0,903	0,916	0,903	2%	0%
32	1,469	1,483	1,469	1%	0%

A Figura C.30 mostra o gráfico que apresenta o número de transistores utilizados pelos somadores BSDA_C2.

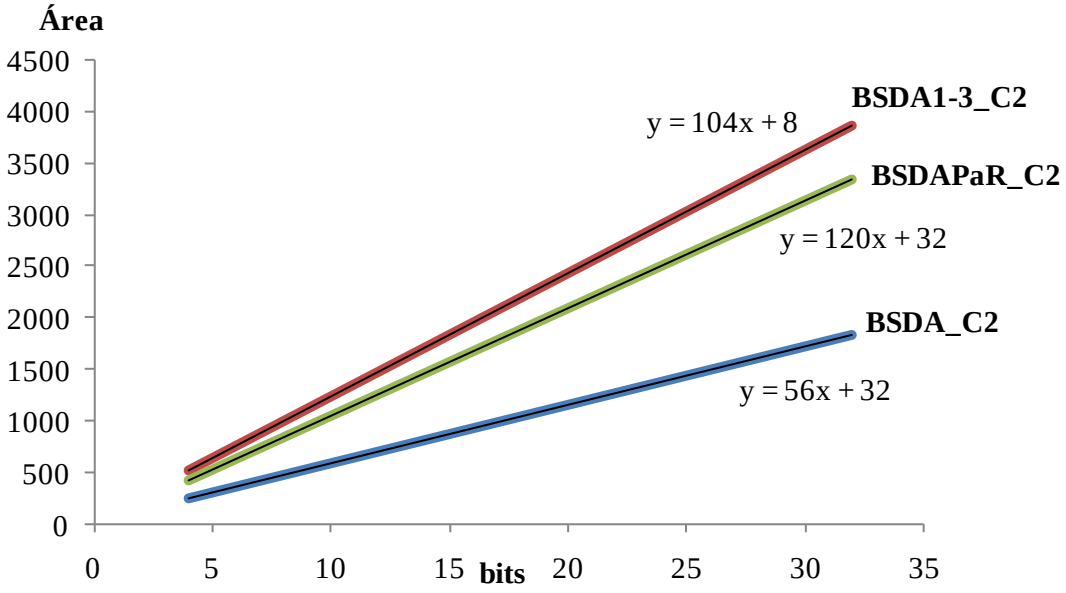


Figura C.30: Número de transistores utilizados pelos somadores BSDA_C2

Como pode ser observado na Figura C.30, as versões protegidas do somadores BSDA_C2 utilizam uma quantidade consideravelmente maior de transistores que o BSDA_C2 não-protegido.

Como pode ser visto na Tabela C.33, conforme aumenta o número de bits dos somadores, aumenta a diferença entre o número de transistores utilizados pelos somadores BSDA1-3_C2 e BSDAPar_C2 em relação ao BSDA_C2. Assim os somadores BSDA1-3_C2 e BSDAPar_C2 de 4 bits utilizam, respectivamente, 100% e 66% mais transistores que o BSDA_C2 de 4 bits, ao passo que a maior diferença é apresentada pelos somadores de 32 bits, sendo que o BSDA1-3_C2 utiliza 112% mais transistores que o BSDA_C2, ao passo que o BSDAPar_C2 utiliza 83%. Isso se deve ao fato dos somadores BSDA_C2 apresentarem um aumento constante à medida que o número de bits dos somadores aumenta, pois são regulares.

Tabela C.33: Comparação do número de transistores utilizados pelos somadores BSDA_C2

bits	BSDA_C2 (1)	BSDA1-3_C2 (2)	BSDAPar_C2 (3)	Acréscimo [(2)/(1)-1*100]	Acréscimo [(3)/(1)-1*100]
4	256	512	424	100%	66%
8	480	992	840	107%	75%
16	928	1952	1672	110%	80%
32	1824	3872	3336	112%	83%

Também pode ser observado na Tabela C.33 que o BSDA1-3_C2 utiliza, em média, 107% mais transistores que o BSDA_C2, ao passo que o BSDAPar_C2 utiliza, em média, 76% mais transistores que o BSDA_C2.

A Figura C.31 mostra o gráfico que apresenta a potência consumida pelos somadores BSDA_C2.

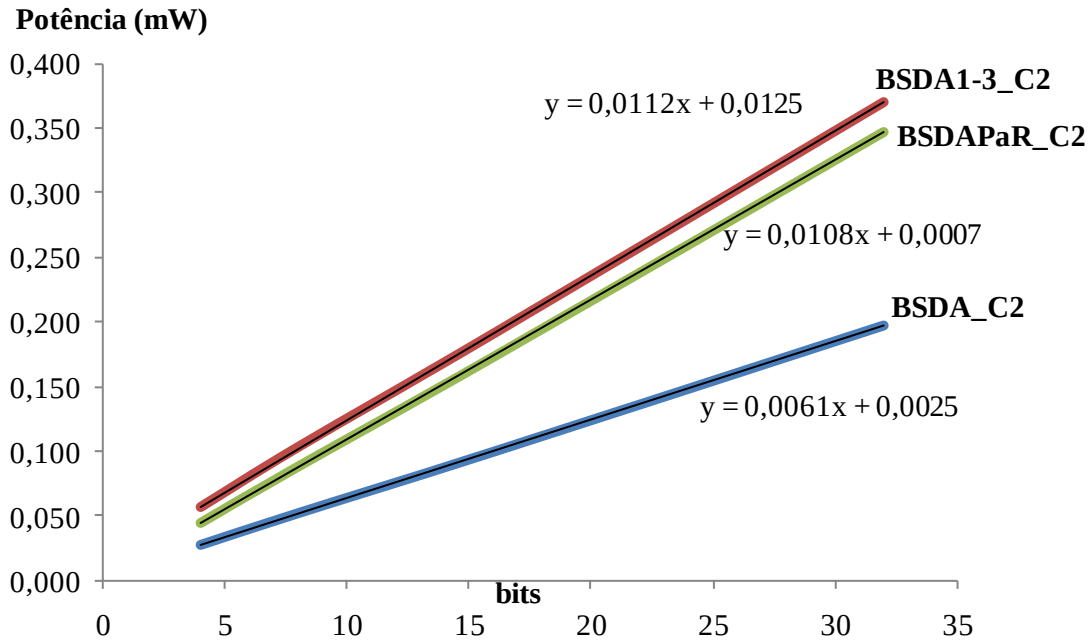


Figura C.31: Potência consumida pelos somadores BSDA_C2

A Figura C.31 mostra que os somadores BSDA_C2 protegidos apresentam um consumo de potência consideravelmente maior que o BSDA_C2.

A Tabela C.34 mostra que conforme aumenta o número de bits dos somadores, diminui a diferença entre a potência consumida pelos BSDA_C2 e BSDA1-3_C2. Assim o somador BSDA1-3_C2 de 4 bits consome 112% mais potência que o BSDA_C2 de 4 bits, ao passo que a menor diferença foi apresentada pelos somadores de 32 bits, sendo que o BSDA1-3_C2 consome 89% mais potência que o BSDA_C2.

Tabela C.34: Comparação da potência consumida pelos somadores BSDA_C2

bits	BSDA_C2 (1)	BSDA1-3_C2 (2)	BSDAPar_C2 (3)	Acréscimo [(2)/(1)-1*100]	Acréscimo [(3)/(1)-1*100]
4	0,027	0,057	0,044	112%	65%
8	0,051	0,103	0,088	101%	71%
16	0,099	0,191	0,173	93%	75%
32	0,197	0,371	0,347	89%	77%

No entanto, quando o BSDAPar_C2 é analisado, a Tabela C.34 mostra que conforme o número de bits dos somadores aumenta, a diferença entre a potência consumida pelos somadores BSDAPar_C2 e BSDA_C2 também aumenta. Assim o somador BSDAPar_C2 de 4 bits consome 65% mais potência que o BSDA_C2 de 4 bits, ao passo que a maior diferença foi apresentada pelos somadores de 32 bits, sendo que o BSDAPar_C2 consome 77% mais potência que o BSDA_C2. Também pode ser observado que o BSDA1-3_C2 consome, em média, 99% mais potência que o BSDA_C2, ao passo que o BSDAPar_C2 consome, em média, 72% mais potência que o BSDA_C2.

C.7 Comparação entre RCA protegidos e BSDA_C2 protegidos

A Figura C.32 mostra o gráfico que apresenta o atraso crítico somadores RCATMR, BSDA1-3_C2 e BSDAPar_C2.

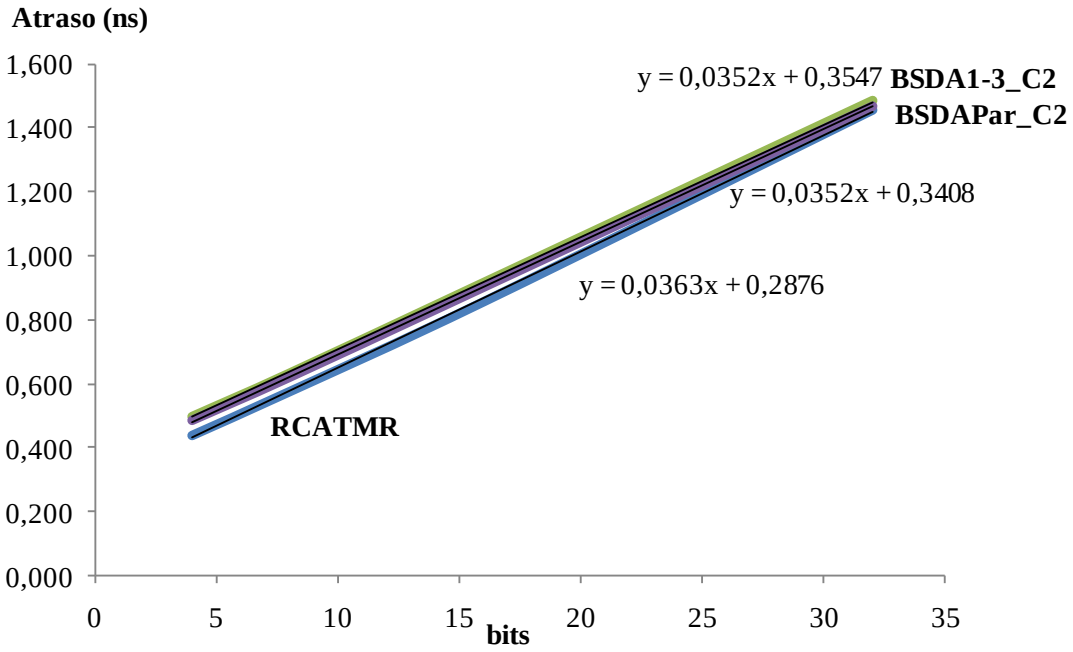


Figura C.32: Atraso crítico dos somadores RCATMR, BSDA1-3_C2 e BSDAPar_C2

Como pode ser visto na Figura C.32, os somadores BSDA_C2 protegidos apresentam praticamente o mesmo atraso crítico que o RCATMR. Isso é devido ao fato dos somadores BSDA_C2 utilizarem um RCA para a etapa de conversão de saída, pois mesmo o RCATMR utilizando três cópias do RCA, estas executam as somas em paralelo, possuindo assim o atraso crítico igual ao de um RCA mais o votador. Igualmente aos somadores BSDA_C2 que possuem o atraso crítico de uma RCA mais o atraso do bloco que executa a soma BSD propriamente dita.

A Tabela C.35 mostra que o aumento no atraso do BSDA1-3_C2 em relação ao RCATMR é, em média, 8% maior. Ainda pode ser observado que com o aumento do número de bits dos somadores, esta diferença entre os atrasos fica cada vez menor, pois os atrasos inseridos pela técnica de proteção com codificação 1 de 3 ao BSDA_C2 e pelo votador ao RCA tornam-se insignificante diante do atraso RCA utilizado no conversor do BSDA1-3_C2 e no RCATMR. Por este mesmo motivo, a diferença entre os atrasos críticos dos o BSDAPar_C2 e RCATMR diminui conforme o número de bits

dos somadores aumenta. Também pode ser visto que o BSDAPar_C2 apresenta,em média, um atraso 6% maior que o RCATMR.

Tabela C.35: Comparação do atraso crítico (em ns) dos somadores RCATMR, BSDA1-3_C2 e BSDAPar_C2

bits	RCATMR (1)	BSDA1-3_C2 (2)	BSDAPar_C2 (3)	Acréscimo [(2)/(1)-1*100]	Acréscimo [(3)/(1)-1*100]
4	0,441	0,499	0,485	13%	10%
8	0,577	0,634	0,620	10%	7%
16	0,858	0,916	0,903	7%	5%
32	1,455	1,483	1,469	2%	1%

A Figura C.33 mostra o gráfico que apresenta o número de transistores utilizados pelos somadores RCATMR, BSDA1-3_C2 e BSDAPar_C2.

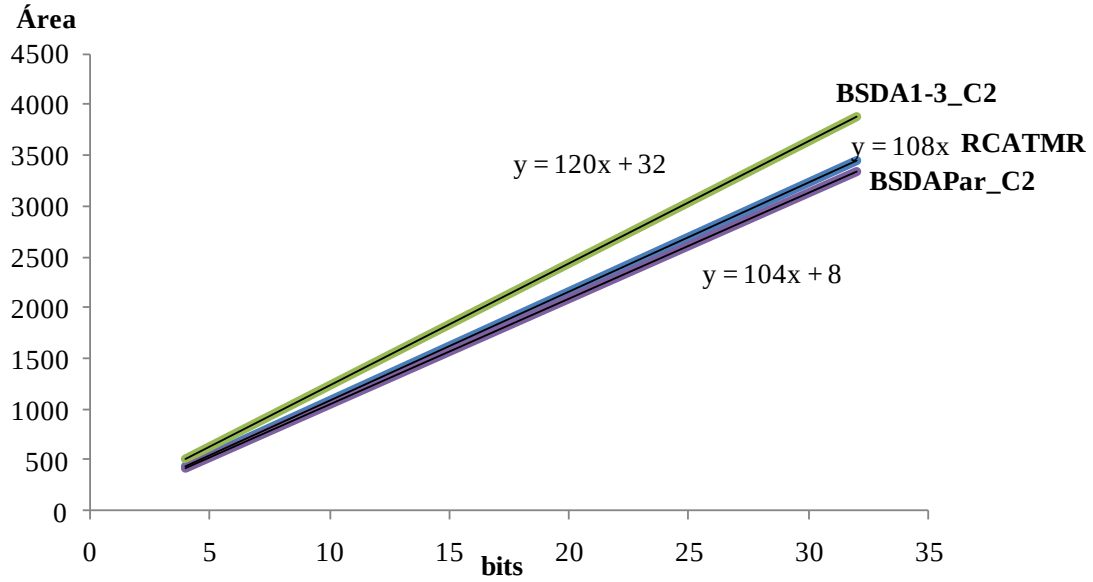


Figura C.33: Número de transistores utilizados pelos somadores RCATMR, BSDA1-3_C2 e BSDAPar_C2

Pode ser observado na Figura C.33 que o BSDAPar_C2 utiliza menos transistores que o demais somadores. Ao passo que o somador BSDA1-3_C2 utiliza mais transistores entre os somadores analisados.

Pode ser observado na Tabela C.36 que a diferença entre a quantidade de transistores utilizados pelo RCATMR e o BSDAPar_C2 é mínima, sendo que o RCATMR utiliza, em média, 3% mais transistores que o BSDAPar_C2, ao passo que o BSDA1-3_C2 utiliza,em média, 15% mais transistores que o RCATMR.

Tabela C.36: Comparação do número de transistores utilizados pelos somadores RCATMR, BSDA1-3_C2 e BSDAPar_C2

bits	RCATMR (1)	BSDA1-3_C2 (2)	BSDAPar_C2 (3)	Acréscimo [(2)/(1)-1*100]	Acréscimo [(3)/(1)-1*100]
4	432	512	424	19%	2%
8	864	992	840	15%	3%
16	1728	1952	1672	13%	3%
32	3456	3872	3336	12%	4%

A Figura C.34 mostra o gráfico que apresenta a potência consumida pelos somadores RCATMR, BSDA1-3_C2 e BSDAPar_C2.

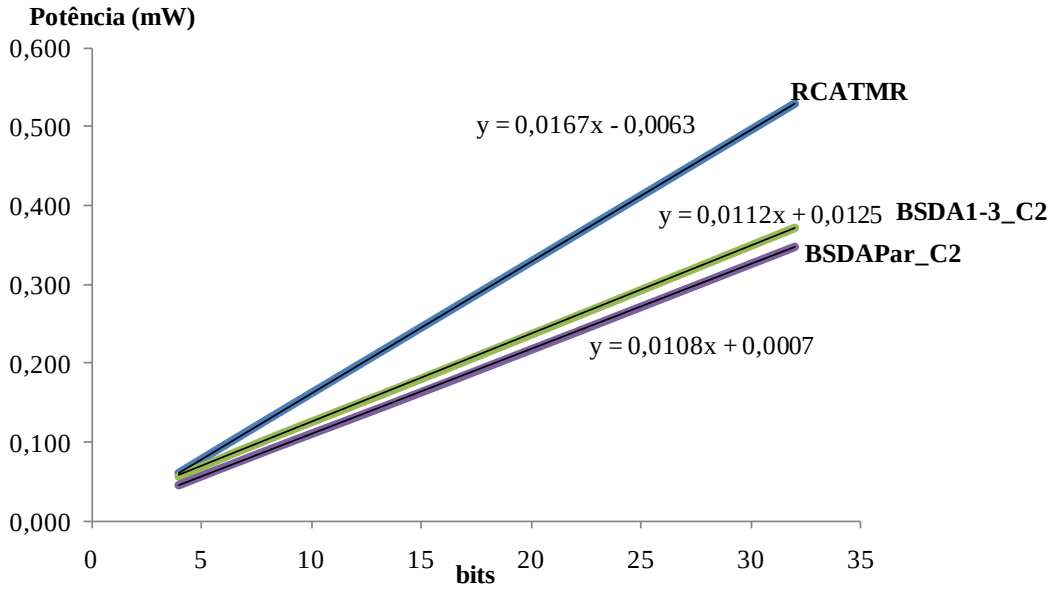


Figura C.34: Potência consumida pelos somadores RCATMR, BSDA1-3_C2 e BSDAPar_C2

Pode ser visto na Figura C.34 que ambos somadores BSDA_C2 apresentam um consumo de potência inferior ao consumo do somador RCATMR. Isso ocorre porque as técnicas de proteção aplicadas aos somadores BSDA1-3_C2 e BSDAPar_C2 não geram muita atividade de chaveamento ao contrário dos somadores RCA utilizados pelo RCATMR.

A Tabela C.37 mostra que, conforme o número de bits dos somadores aumenta, tanto o BSDA1-3_C2 quanto o BSDAPar_C2, apresentam um aumento entre as diferenças da potência consumida em relação ao RCATMR. Assim os somadores de 4 bits apresentam a menor diferença, sendo que o RCATMR apresenta um consumo apenas 7% maior que o BSDA1-3_C2 e 37% maior que o BSDAPar_C2. Ao passo que a maior diferença entre as potências consumidas foi apresentada pelos somadores de 32 bits, onde o RCATMR ficou 42% maior que o BSDA1-3_C2 e 52% maior que o BSDAPar_C2.

Tabela C.37: Comparação da potência consumida pelos somadores RCATMR, BSDA1-3_C2 e BSDAPar_C2

bits	**RCATMR (1)**	**BSDA1-3_C2 (2)**	**BSDAPar_C2 (3)**	**Acréscimo [(1)/(2)-1*100]**	**Acréscimo [(1)/(3)-1*100]**
4	0,060	0,057	0,044	7%	37%
8	0,128	0,103	0,088	24%	45%
16	0,261	0,191	0,173	37%	51%
32	0,528	0,371	0,347	42%	52%

Também é possivel observar na Tabela C.37 que o RCATMR consome, em média, 27% mais potência que o BSDA1-3_C2 e, em média, 46% mais potência que o BSDAPar_C2.

A Figura C.35 mostra o gráfico que apresenta o atraso crítico dos somadores RCAIOI, BSDA1-3_C2 e BSDAPar_C2.

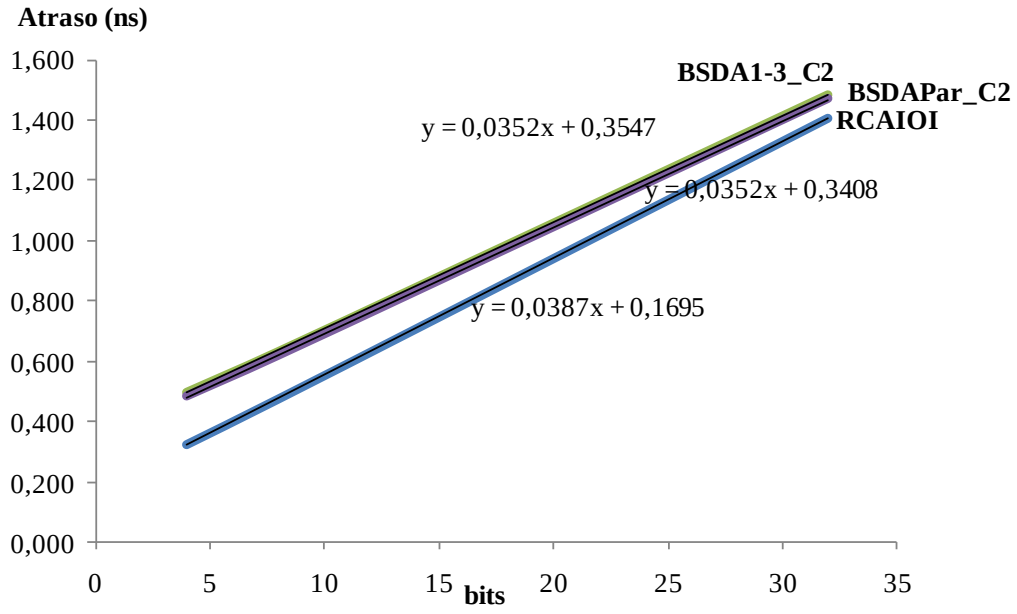


Figura C.35: Atraso crítico dos somadores RCAIOI, BSDA1-3_C2 e BSDAPar_C2

Como pode ser observado na Figura C.35, o somador RCAIOI apresenta um atraso crítico inferior aos somadores BSDA1-3_C2 e BSDAPar_C2. Isso ocorre porque o somador RCA utilizado pelo RCAIOI foi implementado usando portas NAND, ao passo que o RCA utilizado pelos conversores dos somadore BSDA1-3_C2 e BSDAPar_C2 foram implementados utizando portas CMOS XOR, o que resulta em um atraso maior. Além disto, a verificação de erros é feita em paralelo ao cálculo da soma, não inserindo portanto atraso ao RCAIOI(exceto no caso da ocorrência de uma falha, onde o RCAIOI executará o a soma mais uma vez).

Porém, a Tabela C.38 mostra que, com o aumento do número de bits dos somadores, a diferença entre o atraso crítico do RCAIOI em relação ao atraso crítico dos somadores BSDA1-3_C2 e BSDAPar_C2 diminui. Isso porque com o aumento do número de bits o atraso inserido pela cadeia de *carry* dos RCAs utilizados pelos somadores RCAIOI, BSDA1-3_C2 e BSDAPar_C2 torna-se dominte sobre os atrasos dos demais blocos dos somadores, independentemente de como tais RCAs foram implementados.

Assim o somador BSDA1-3_C2 de 4 bits apresenta um atraso 53% maior que o RCAIOI e o BSDAPar_C2 de 4 bits apresenta um atraso 49% maior que o RCAIOI. Já a menor diferença foi apresentada pelos somadores de 32 bits, sendo que o BSDA1-3_C2 apresenta um atraso 5% maior que o RCAIOI ao passo que o BSDAPar_C2 apresenta um atraso 4% maior que o RCAIOI.

Tabela C.38: Comparação do atraso crítico (em ns) dos somadores RCAIOI, BSDA1-3_C2 e BSDAPar_C2

bits	**RCAIOI (1)**	**BSDA1-3_C2 (2)**	**BSDAPar_C2 (3)**	**Acréscimo [(2)/(1)-1*100]**	**Acréscimo [(3)/(1)-1*100]**
4	0,326	0,499	0,485	53%	49%
8	0,477	0,634	0,620	33%	30%
16	0,786	0,916	0,903	17%	15%
32	1,407	1,483	1,469	5%	4%

Além disso, pode ser observado na Tabela C.38 que o BSDA1-3_C2 apresenta um atraso, em média, 27% maior que o RCAIOI, ao passo que o BSDAPar_C2 apresenta um atraso, em média, 24% maior que o RCAIOI.

A Figura C.36 mostra o gráfico que apresenta o número de transistores utilizado pelos somadores RCAIOI, BSDA1-3_C2 e BSDAPar_C2.

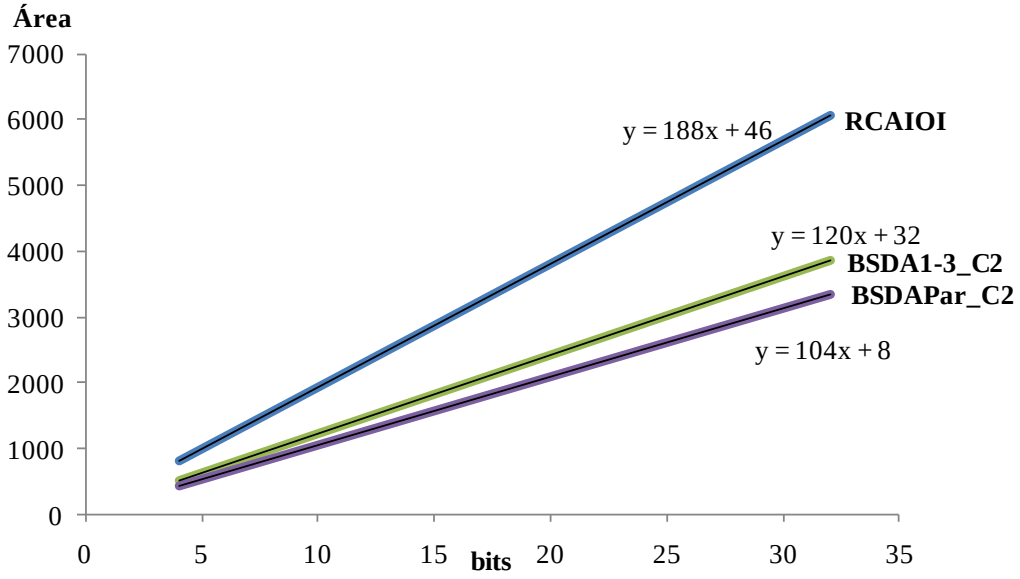


Figura C.36: Número de transistores utilizados pelos somadores RCAIOI, BSDA1-3_C2 e BSDAPar_C2

Pode ser visto na Figura C.36 que o RCAIOI utiliza mais transistores que os somadores BSDA_C2 protegidos. Isso ocorre porque para implementar a técnica de verificaçao de paridade o RCAIOI necessita de uma grande quantidade de portas CMOS XOR, que utiliza mais transistores que as demais. Além disso também são necessarios multiplexadores de entrada e saída e registrador.

A Tabela C.39 mostra que com o aumento do número de bits dos somadores, a diferença entre a quantidade de transistores utilizados pelo RCAIOI em relação aos somadores BSDA1-3_C2 e BSDAPar_C2 manteve-se praticamente constante, pois estes são regulares. Além disso, pode ser visto que o RCAIOI utiliza, em média, 56% mais transistores que o BSDA1-3_C2 e 84% mais transistores que o BSDAPar_C2.

Tabela C.39: Comparação do número de transistores utilizados pelos somadores RCAIOI, BSDA1-3_C2 e BSDAPar_C2

bits	RCAIOI (1)	BSDA1-3_C2 (2)	BSDAPar_C2 (3)	Acréscimo [(1)/(2)-1*100]	Acréscimo [(1)/(3)-1*100]
4	798	512	424	56%	88%
8	1550	992	840	56%	85%
16	3054	1952	1672	56%	83%
32	6062	3872	3336	57%	82%

A Figura C.37 mostra o gráfico que apresenta o consumo de potência dos somadores RCAIOI, BSDA1-3_C2 e BSDAPar_C2.

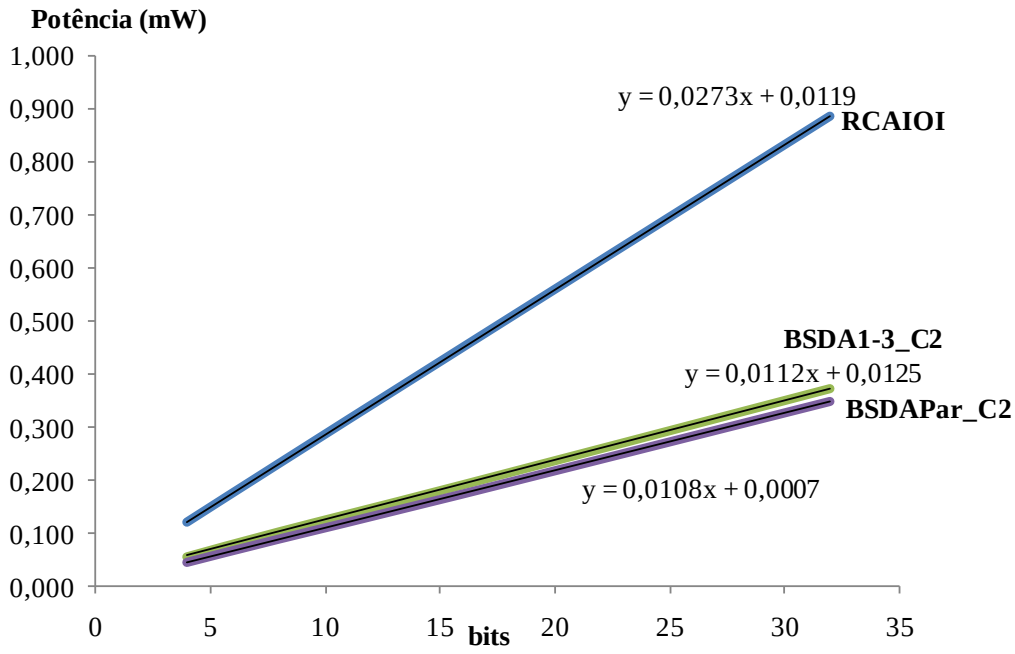


Figura C.37: Potência consumida pelos somadores RCAIOI, BSDA1-3_C2 e BSDAPar_C2

Pode ser observado na Figura C.37 que o RCAIOI consome uma quantidade de potência consideravelmente maior que os somadores BSDA_C2. Isso ocorre porque o somador RCAIOI além de utilizar os blocos necessarios para a implementação da técnica de proteção com verificação de paridade, ainda utiliza o Somador Completo com *Carry* Redundante – SCCR o que contribui para o aumento da atividade de chaveamento gerando assim um maior consumo de potência, ao contrário dos somadores BSDA_C2 que utilizam um RCA normal para a etapa de conversão de saída.

Tabela C.40: Comparação da potência consumida pelos somadores RCAIOI, BSDA1-3_C2 e BSDAPar_C2

bits	RCAIOI (1)	BSDA1-3_C2 (2)	BSDAPar_C2 (3)	Acréscimo [(1)/(2)-1*100]	Acréscimo [(1)/(3)-1*100]
4	0,120	0,057	0,044	113%	173%
8	0,230	0,103	0,088	123%	162%
16	0,449	0,191	0,173	135%	160%
32	0,884	0,371	0,347	139%	155%

A Tabela C.40 mostra que o RCAIOI apresenta um consumo, em média, 127% maior que o BSDA1-3_C2, ao passo que o RCAIOI apresenta um consumo, em média, 162% maior que o BSDAPar_C2.

C.8 Comparação entre BSDA1-3_BSD e BSDA1-3_C2

A Figura C.38 mostra o gráfico que apresenta o atraso crítico dos somadores BSDA1-3_BSD e BSDA1-3_C2.

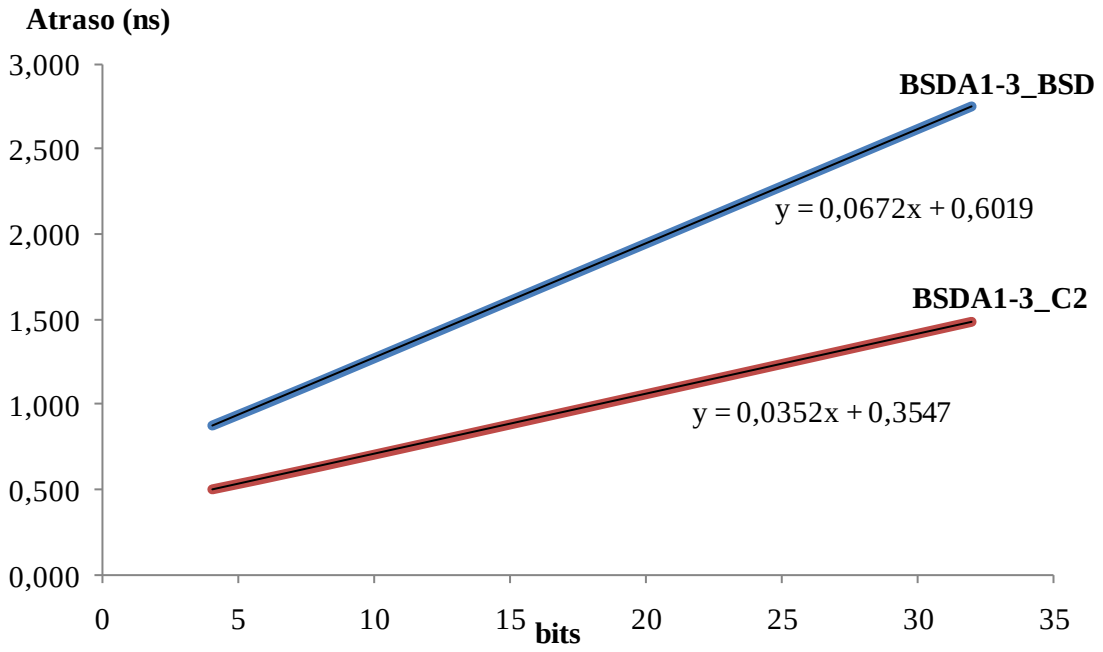


Figura C.38: Atraso crítico dos somadores BSDA1-3_BSD e BSDA1-3_C2

Pode-se observar na Figura C.38 que o BSDA1-3_BSD apresenta um atraso maior que o BSDA1-3_C2, devido à etapa de conversão de entrada que o BSDA1-3_BSD necessita e devido ao circuto que implementa a soma BSD também apresentar um atraso maior que a soma BSD do BSDA1-3_C2. Também é devido a estes mostivos que o BSDA1-3_BSD utiliza mais transistores que o BSDA1-3_C2, como pode ser observado na Figura C.39 que apresenta o número de transistores utilizados por tais somadores.

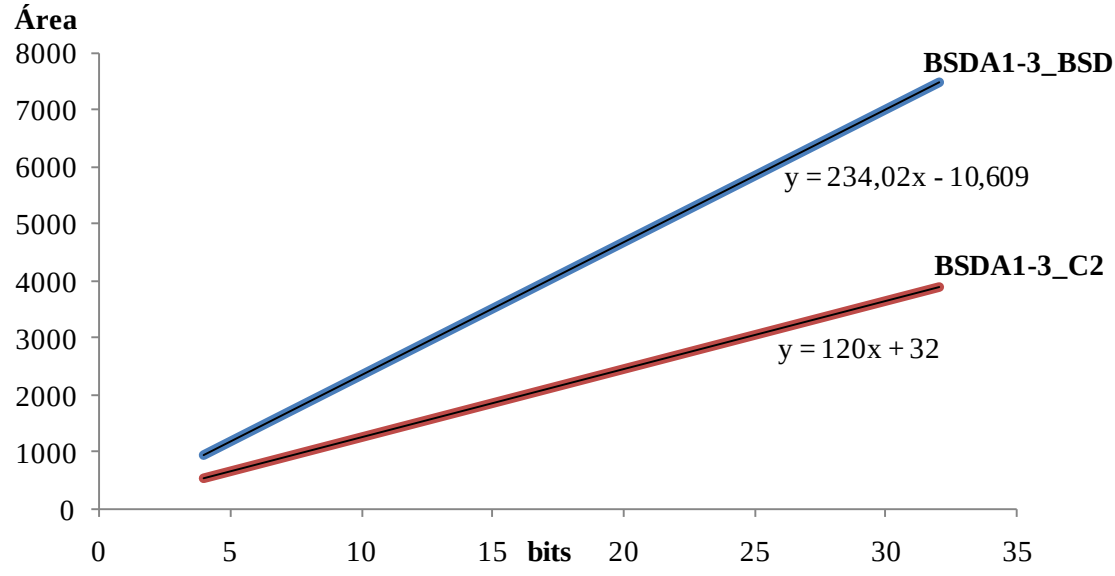


Figura C.39: Número de transistores utilizados pelos somadores BSDA1-3_BSD e BSDA1-3_C2

A Figura C.40 mostra o gráfico que apresenta o consumo de potência dos somadores BSDA1-3_BSD e BSDA1-3_C2 onde é possivel observar que o BSDA1-3_BSD apresenta um consumo de potência maior que o BSDA1-3_C2.

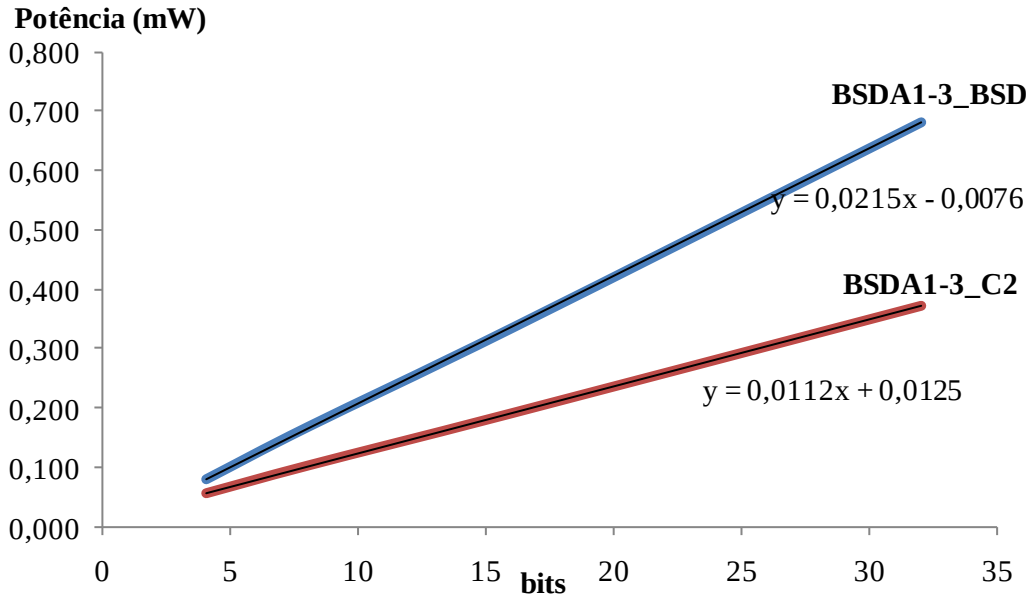


Figura C.40: Potência Consumida pelos somadores BSDA1-3_BSD e BSDA1-3_C2

A Tabela C.41 mostra os resultados de atraso crítico, número de transistores e consumo de potência para os somadores BSDA1-3_BSD e BSDA1-3_C2. Também são mostrados os valores de acréscimo do atraso crítico, de transistores utilizados e de potência consumida, tomando-se o BSDA1-3_C2 como refêrencia.

Tabela C.41: Comparação do atraso crítico, número de transistores e potência consumida entre os somadores BSDA1-3_BSD e BSDA1-3_C2.

	Atraso (ns)			***Potência (mW)***			***Nº de transistores***		
bits	**BSDA 1-3_BSD (1)**	**BSDA 1-3_C2 (2)**	**Acrésc. [(2)/(1)-1*100]**	**BSDA 1-3_BSD (1)**	**BSDA 1-3_C2 (2)**	**Acrésc. [(2)/(1)-1*100]**	**BSDA 1-3_BSD (1)**	**BSDA 1-3_C2 (2)**	**Acrésc. [(2)/(1)-1*100]**
4	0,873	0,499	75%	0,078	0,057	38%	925	512	81%
8	1,137	0,634	79%	0,167	0,103	62%	1862	992	88%
16	1,677	0,916	83%	0,335	0,191	75%	3734	1952	91%
32	2,754	1,483	86%	0,682	0,371	84%	7478	3872	93%

Como pode ser visto na Tabela C.41, conforme o número de bits aumenta a diferença entre o atraso crítico do BSDA1-3_BSD e BSDA1-3_C2 apresenta um pequeno aumento, sendo que o BSDA1-3_BSD de 4 bits apresenta um atraso crítico 75% maior que o BSDA1-3_C2 e o BSDA1-3_BSD de 32 bits é 86% maior que o BSDA1-3_C2 de 32 bits. Também pode ser visto que o BSDA1-3_BSD apresenta um atraso, em média, 81% maior que o BSDA1-3_C2.

Em relação ao consumo de potência, o BSDA1-3_BSD consome, em média, 65% mais potência que o BSDA1-3_C2 e conforme o número de bits dos somadores aumenta, a diferença entre o consumo de potência dos somadores também aumenta. Assim o consumo de potência do BSDA1-3_BSD de 4 bits apresenta a menor diferença em relação ao BSDA1-3_C2 de 4 bits, ficando 38% menor. Já o BSDA1-3_BSD de 32 bits apresenta a maior diferença em relação ao BSDA1-3_C2 de 32 bits, consumindo 84% mais potência. Além disso, pode ser visto que o BSDA1-3_BSD consome, em média, 65% mais potência que o BSDA1-3_C2.

Em relação ao número de transistores utilizados, o BSDA1-3_BSD utiliza, em média, 88% mais transistores que o BSDA1-3_C2, sendo que com o aumento do número de bits dos somadores, a diferença entre os transistores utilizados pelo BSDA1-3_BSD em relação ao BSDA1-3_C2 apresenta um pequeno aumento. Assim o BSDA1-3_BSD de 4 bits utiliza 81% mais transistores que o BSDA1-3_C2 de 4 bits, ao passo que o BSDA1-3_BSD de 32 bits utiliza 93% mais transistores que o BSDA1-3_C2 de 32 bits.

C.9 Comparação entre BSDAPar_BSD e BSDAPar_C2

A Figura C.41 mostra o gráfico que apresenta o atraso crítico dos somadores BSDAPar_BSD e BSDAPar_C2.

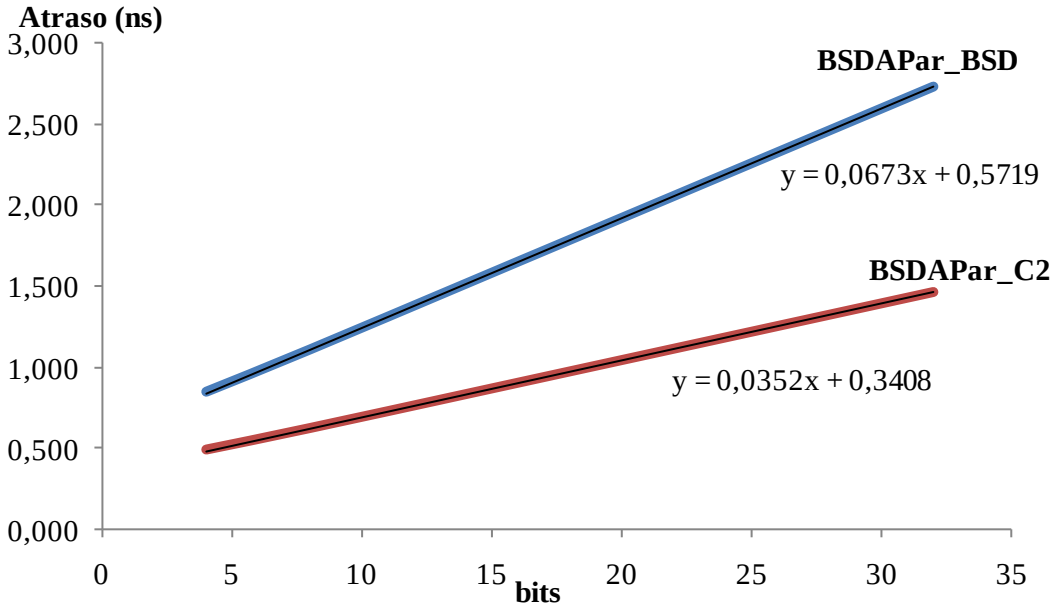


Figura C.41: Atraso crítico dos somadores BSDAPar_BSD e BSDAPar_C2

Pode-se observar na Figura C.41 que o BSDAPar_BSD apresenta um atraso maior que o BSDAPar_C2, devido à etapa de conversão de entrada que o BSDAPar_BSD necessita e devido ao circuto que implementa a soma BSD deste também apresentar um atraso maior que a soma BSD do BSDAPar_C2. Também é devido a estes mostivos que o BSDAPar_BSD utiliza mais transistores que o BSDAPar_C2, como pode ser observado na

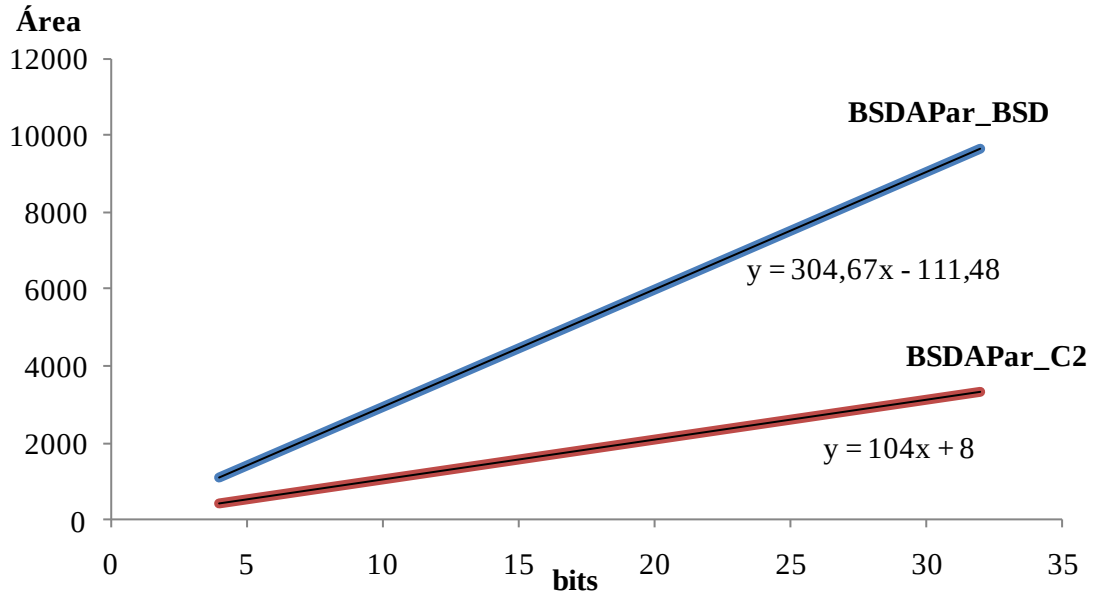


Figura C.42 que apresenta o número de transistores utilizados por tais somadores.

A

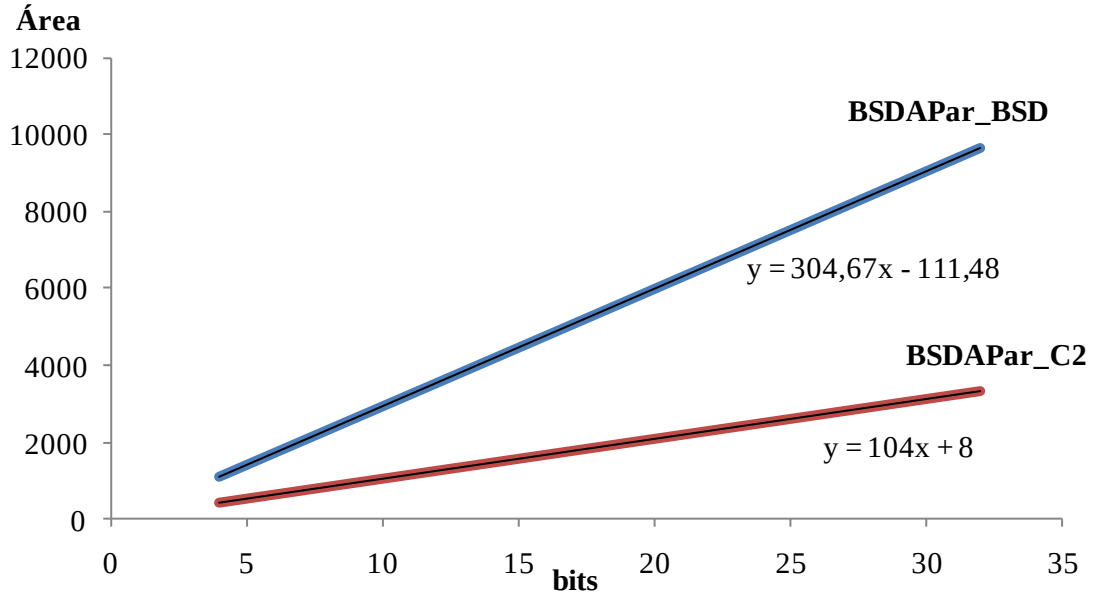


Figura C.42 mostra o gráfico que apresenta o número de transistores utilizados pelos somadores BSDAPar_BSD e BSDAPar_C2.

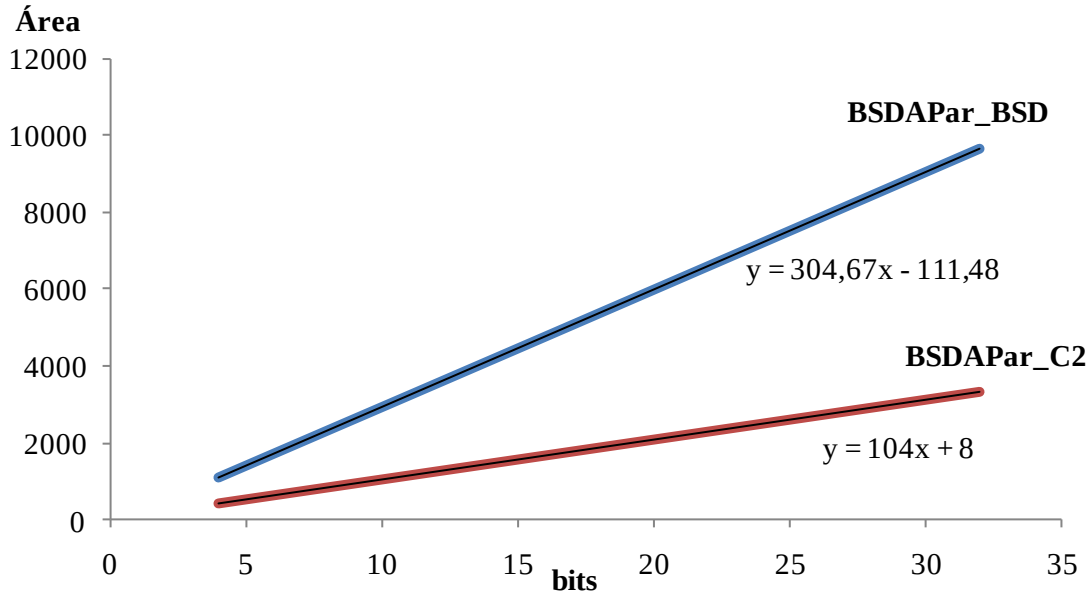


Figura C.42: Número de transistores utilizados pelos somadores BSDAPar_BSD e BSDAPar_C2

A Figura C.43 mostra o gráfico que apresenta o consumo de potência dos somadores BSDAPar_BSD e BSDAPar_C2 onde é possivel observar que o BSDAPar_BSD também apresenta um consumo de potência maior que BSDAPar_C2.

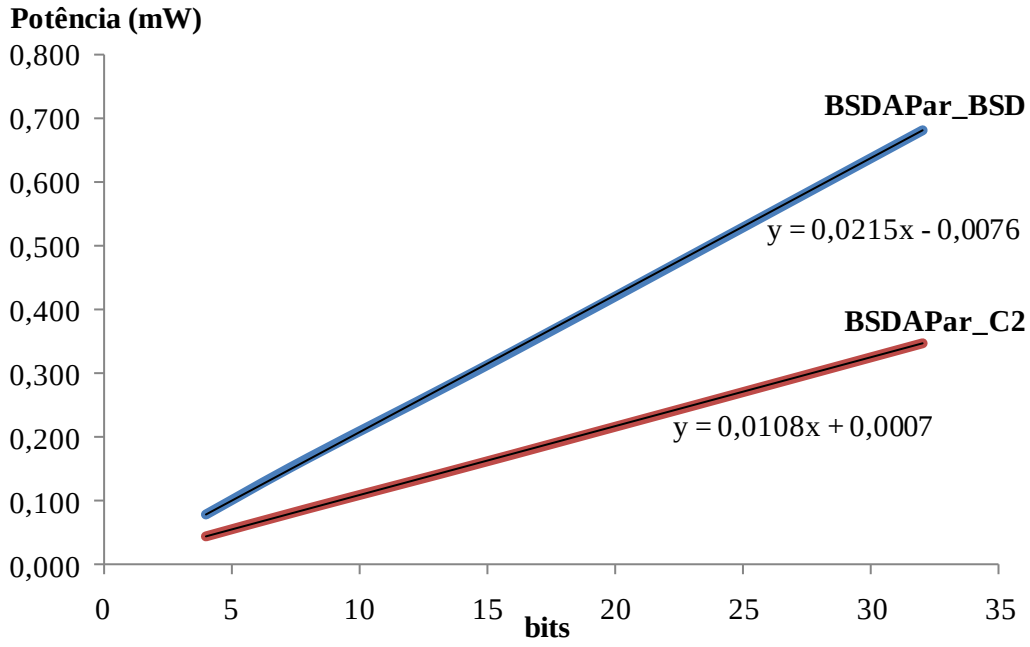


Figura C.43: Potência consumida pelos somadores BSDAPar_BSD e BSDAPar_C2

Tabela C.42: Comparação do atraso crítico, número de transistores e potência consumida entre os somadores BSDAPar_BSD e BSDAPar_C2.

	Atraso (ns)			***Potência (mW)***			***Nº de Transistores***		
bits	**BSDA Par_BSD (1)**	**BSDA Par_C2 (2)**	**Acrésc. [(2)/(1)-1*100]**	**BSDA Par_BSD (1)**	**BSDA Par_C2 (2)**	**Acrésc. [(2)/(1)-1*100]**	**BSDA Par_BSD (1)**	**BSDA Par_C2 (2)**	**Acrésc. [(2)/(1)-1*100]**
4	0,845	0,485	74%	0,078	0,044	77%	1110	424	162%
8	1,107	0,620	79%	0,167	0,088	90%	2326	840	177%
16	1,648	0,903	83%	0,335	0,173	93%	4758	1672	185%
32	2,728	1,469	86%	0,682	0,347	96%	9640	3336	189%

Como pode ser visto na Tabela C.42, conforme o número de bits aumenta, a diferença entre o atraso crítico do BSDAPar_BSD e BSDAPar_C2 apresenta um pequeno aumento, sendo que o BSDAPar_BSD de 4 bits apresenta um atraso crítico 74% maior que o BSDAPar_C2 e o BSDAPar_BSD de 32 bits é 86% maior que o BSDAPar_C2 de 32 bits. Também pode ser visto que o BSDAPar_BSD apresenta um atraso, em média, 80% maior que o BSDAPar_C2.

Em relação ao consumo de potência, e conforme o número de bits dos somadores aumenta, a diferença entre o consumo de potência dos somadores também aumenta. Assim o consumo de potência do BSDAPar_BSD de 4 bits apresenta a menor diferença em relação ao BSDAPar_C2 de 4 bits, ficando 77% menor. Já o BSDAPar_BSD de 32 bits apresenta a maior diferença em relação ao BSDAPar_C2 de 32 bits, consumindo 96% mais potência. Além disso, pode ser visto na Tabela C.42 que o BSDAPar_BSD consome, em média, 89% mais potência que o BSDAPar_C2.

Em relação ao número de transistores utilizados, o BSDAPar_BSD utiliza, em média, 178% mais transistores que o BSDAPar_C2, sendo que com o aumento do número de bits dos somadores, a diferença entre os transistores utilizados pelo BSDAPar_BSD em relação ao BSDAPar_C2 apresenta um pequeno aumento. Assim o BSDAPar_BSD de 4 bits utiliza 162% mais transistores que o BSDAPar_C2 e o BSDAPar_BSD de 32 bits utiliza 189% mais transistores que o BSDAPar_C2 de 32 bits.