Allender Vilar de Alencar

Ferramenta Para Análise de Confiabilidade de Circuitos Digitais

Campina Grande, Brasil

Dezembro de 2019

Allender Vilar de Alencar

Ferramenta Para Análise de Confiabilidade de Circuitos Digitais

Trabalho de Conclusão de Curso submetido à Coordenação de Graduação em Engenharia Elétrica da Universidade Federal de Campina Grande como parte dos requisitos necessários para a obtenção do grau de Engenheiro Eletricista.

Universidade Federal de Campina Grande - UFCG

Centro de Engenharia Elétrica e Informática - CEEI

Departamento de Engenharia Elétrica - DEE

Coordenação de Graduação em Engenharia Elétrica - CGEE

Orientador: Gutemberg Gonçalves dos Santos Júnior

Campina Grande, Brasil

Dezembro de 2019

Agradecimentos

Agradeço primeiramente aos cientistas do passado, porque sem os seus estudos insanos de anatomia, possivelmente, não estaria aqui para realizar esta tese e o meu frágil vínculo de vida estaria estilhaçado no universo.

À minha mãe, Ana, e aos meus irmãos, Alluska, Aryadne e Aristótenes, pelo apoio e ajuda em grande parte da minha vida. A todos os membros da minha família, seja de sangue, ou não, e em especial, à minha tia e meus avós por parte de mãe que deram bastante suporte ao longo da vida.

A José Vinicius, Erivan e todos aqueles que me ajudaram durante a graduação à permanecer firme frente aos problemas, resolvendo alguma questão ou ensinando-me algum assunto.

Aos meus amigos que desferem as críticas mais incisivas, ajudando-me assim, a ter não somente uma nova visão sobre as coisas, mas também, formas de como me aprimorar e me entender.

Ao professores do curso de Engenharia Elétrica que cada qual com sua convicção luta firme e com muito amor ao curso para aprimorá-lo e ajudar os nossos queridos estudantes. Em especial, ao meu oriantador, Gutemberg, por ter dado o suporte necessário para realização deste trabalho.

A todos os que tornam a dor existencial mais suave e leve, como um doce, que enquanto dura, tem textura indescritível de paz e amor, e no fim, um tom amargo-cintilante.

"Não ser amado é uma simples desventura.
A verdadeira desgraça é não saber amar."
Albert Camus

Resumo

Atualmente, com o aumento da complexidade dos circuitos digitais e diminuição do tamanho das tecnologias, os circuitos digitais estão mais susceptíveis a faltas. Para obter mais praticidade para estimar, precisamente, a confiabilidade dos mesmos, devido a circuitos críticos (Os que compõem sistemas que a sua falha põe em risco vidas humanas), Assim, Está descrito neste documento uma ferramenta e uma forma de análise, dado isso, deve ser possível obter a *netlist* do circuito a ser analisado. A ferramenta foi descrita parte em *SystemVerilog* e parte em *Python*.

Palavras-chave: Confiabilidade, falta, *netlist*, Moore, circuitos.

Abstract

Currently, with the increase of the complexity of digital circuits and the decreasing size of transistor, the digital circuits are more susceptible to faults. To obtain more practicality in estimatation, precisely, of the reliability of them, due to critical circuits (circuits that compose systems that it‘s fault put lives in risk), thereby, It’s is described in this present document a tool and an analysis form, given that, with a given circuit netlist is possible to evaluate the realiability of the circuit. The tool was developed part with SystemVerilog and part with Python.

Keywords: Reliability, fault, netlist, Moore, circuits.

Lista de tabelas

Lista de ilustrações

Lista de abreviaturas e siglas

HPC	*High-performance computing*
FPGA	*Field Programmable Gate Array*
IP	*Intellectual Property*
UART	*Universal Asynchronous Receiver/Transmitter*

Sumário

1 Introdução

O primeiro computador foi construído em *Iowa State College* no ano de 1942. Esse computador era composto por poucas peças, mas, com o avanço das tecnologias foi possível torná-los mais complexos, menores e eficientes devido a pesquisas realizadas na área (JúNIOR, 2012).

Segundo (West; Harris, 2011), Os inventores do transistor ganharam o prêmio nobel da física no ano de 1956, sendo eles, Bardeen, Brattain e seus supervisor Willian Shockley. Essa que foi a invenção que revolucionou a forma de computação de dados.

Com a invenção do transistor, temos uma industria que busca conseguir instrumentar tamanhos cada vez menores de transistores no silício, pois acabam se tornando cada vez menos custosas as unidades. Os dois tipos principais de transistores são: junção bipolar (*bipolar junction transistors*) e metal-oxido-semicondutor (*metal-oxide-semiconductor field effect transistors*).(Harris; Harris, 2013)

Na atualidade, temos bilhões de transistores em um único chip, isso se deve à diminuição extremamente rápida das dimensões do transistor e da criação de técnicas ao longo dos anos, possibilitando assim formas cada vez mais poderosas de computação e de controle.(Patterson; Hennessy, 2017)

Devido a essa diminuição ascentuada nas dimensões dos transistores faz surgir sérios problemas que têm que ser levados em consideração no fluxo de criação de chips, como consumo de potência, dissipação de calor, fuga de corrente e variação de parâmetros. Como instrinsecamente esses parâmetros também comprometem a confiabilidade de cada porta lógica, a confiabilidade dos circuitos tem que ser, cada vez mais, levada em consideração nesse fluxo, para não comprometer esses ganhos que existem devido ao avanço da tecnologia de fabricação. (FRANCO et al., 2008)

"A confiabilidade de um circuito lógico é uma medida da sua susceptibilidade às faltas permanente, intermitente e transiente."(FRANCO et al., 2008). Nos circuitos digitais as faltas são a inversão do bit na porta lógica. Essas faltas podem ser originadas por erros

de *software*, *thermal bit-flip*, variação de parâmetros, Neutrons de alta energia, partículas alfa, entre outros. Devido à diminuição do tamanho dos transistores, seu funcionamento se dá com baixos níveis de tensão e corrente, dessa forma, torna-se cada vez mais susceptível à faltas, o que torna a confiabilidade um aspecto muito importante do circuito.

1.1 Objetivos

O presente trabalho contém a proposta de criar uma ferramenta para analisar a confiabilidade de uma específica microarquitetura, se a mesma é confiável ou não para uma dada aplicação. Após isso, é possível comparar diferentes microarquiteturas e decidir qual delas é mais confiável.

Sendo possível partir de modelos matemáticos que descrevem a probabilidade de falha de cada porta e o teorema de Bayes, obtermos a probabilidade geral de falha do circuito, sendo dada a probabilidade de ocorrência de cada entrada pelo usuário. Essa probabilidade é computada partindo de estatísticas de uso do circuito analisado.

Com isso, produzir uma ferramente em ter posse de uma ferramenta simples que seja funcional e útil em análise e comparação de microarquiteturas diferentes que foram projetadas para o mesmo funcionamento.

1.2 Organização do trabalho

O Capítulo 2 contém a fundamentação teórica com as devidas demonstrações das equações que foram utilizadas para realizar o cálculo da confiabilidade utilizando a ferramenta, tendo uma abordagem mais direcionada ao funcionamento da ferramenta em si.

O Capítulo 3 descreve com detalhes a microarquitetura da ferramenta, abordando de forma modular o seu funcionamento e os procedimentos de sua utilização.

O Capítulo 4 apresenta resultados obtidos com o uso da ferramenta e alguns passos para instanciar a ferramenta, para os estudos de caso do circuito c17 que faz parte do conjunto iscas 85, que são conjuto de circuitos para realizar o cálculo de confiabilidade, e do circuito de um somador completo de 8 bits.

O Capítulo 5 contém as conclusões do trabalho em um formato mais simplificado, análise de resultados e trabalhos futuros, visando um aperfeiçoamento complementar do mesmo.

2 Confiabilidade de Circuitos digitais

Neste texto estão modeladas as equações principais do método utilizando, a probabilidade binomial de confiabilidade do artigo (de Vasconcelos et al., 2008), esse artigo será utilizado como base para esse capítulo. Sendo descrito da ideia de concepção da confiabilidade até a forma da equação fundamental da ferramenta. A equação que será demonstrada é a da figura 1 abaixo.

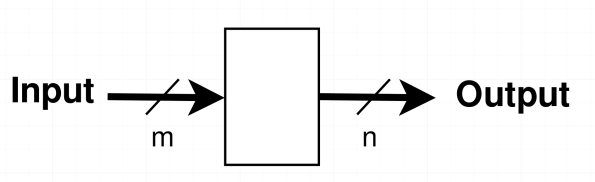
Figura 1 – Modelo de probabilidade binomial de confiabilidade.

$$R = \sum_{k=0}^{2^w-1} \prod_{correct} q_i \prod_{incorrect} (1-q_i) \sum_{j=0}^{2^m-1} p(x_j) \left(\overline{y(\mathbf{x}_j, \mathbf{e}(0)) \oplus y(\mathbf{x}_j, \mathbf{e}(k))} \right)$$

Fonte: Próprio Autor.

Como não é possível calcular de forma determinística se a saída de um cicuito está correta, a confiabilidade de um circuito é simplesmente a probabilidade da saída estar correta, instrinsecamente a saída dependerá do vetor de entradas e se as portas lógicas inverteram ou não seus bits de saída.

Figura 2 – Circuito combinacional para análise.



Fonte: Próprio Autor.

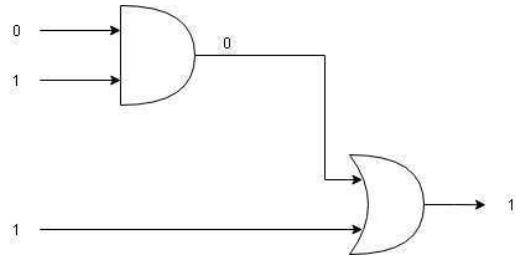
A probabilidade da entrada ser um vetor específico é a de ocorrência desse vetor de entrada, podendo ser modelada por uma distruibuição discreta de probabilidade ou ser estimado de forma estatística, essa segunda tem um custo de *hardware* muito alto. Em geral, essa distruibuição é feita de forma que a probabilidade de cada x_j seja igual, ou seja, $P(X = x_j) = \frac{1}{2^m}$. Ou simplesmente, $P(x_j) = \frac{1}{2^m}$. Com isso, ainda temos que calcular a probabilidade da saída estar correta, ou seja, $R = P(y = correta)$, onde R é a confiabilidade. Abrindo a confiabilidade usando os eventos, temos a ocorrência em ordem dos seguintes eventos: a entrada ser um vetor x_j, e dado que esse evento ocorreu, a saída estar correta, para todos os possíveis vetores de bits (j variar de 0 até $2^m - 1$), segue abaixo a equação que descreve essa confiabilidade.

$$R = \sum_{j=0}^{2^m-1} P(x_j, y = correta) = \sum_{j=0}^{2^m-1} P(x_j)P(y = correta|x_j) \tag{2.1}$$

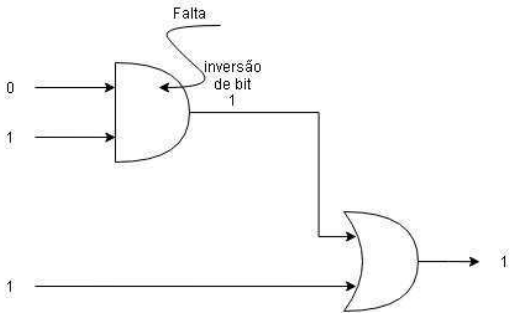
A probabilidade condicional da saída estar correta dado uma entrada depende da falta que está acontecendo no momento, devido a possibilidade de ocorrer um mascaramento da falha dentro do circuito, ou seja, a inversão do bit em uma ou mais portas lógicas não é propagada para a saída e a saída será correta nesse caso, aumentando assim a confiabilidade do circuito. A figura abaixo exemplifica esse funcionamento, ou seja, para o vetor de entrada (011), mesmo que haja uma falta na porta *And* a saída permanece inalterada.

Figura 3 – Circuito com erro mascarado

(a) Sem ocorrência de falta.



(b) Com ocorrência de falta na porta And.



Fonte: Próprio Autor.

Dessa forma, temos que modelar além da probabilidade do vetor de entrada, o vetor de faltas. Esse segundo corresponde ao conjunto de bits que serão ligados ao circuito com faltas, esse funcionamento está descrito na seção 3.3 da ferramenta, cada bit desse vetor caracteriza a falta na porta lógica, ou seja, caso o sinal seja '1' o bit de saída será invertido, caso o sinal seja '0' a saída não será invertida.

O vetor de faltas será caracterizado por $e_{w:k}(l)$, pois esses bits serão utilizados para simular as faltas, onde w é a quantidade de portas lógicas, k é a quantidade de faltas simultâneas (quantidade de portas lógicas que alteram os seus bits ao mesmo tempo) e l é a posição dos bits de inversão no vetor (como existem para uma certa quantidade de portas w e faltas k, $\binom{w}{k}$ combinações possíveis, l varia de 1 até $\binom{w}{k}$). exemplo: $e_{5:2}(3) = 10010$, essa posição se deve ao algorítmo que será explicado nas próximas seções.

2.1 Probabilidade das entradas

2.1.1 Distribuição de probabilidade

A modelagem mais básica e funcional que se tem e que foi utilizada para construção da ferramenta é utilizar uma distribuição "uniforme"discreta, ou seja, caso haja uma quantidade de entradas 2^m a probabilidade de um vetor qualquer será igual a $\frac{1}{2^m}$, como descrito anteriormente. Segue a tabela abaixo ilustrando alguns desses pontos.

Tabela 1 – Probabilidade comportamental da entrada.

x_j	$P(x_j)$
0	$\frac{1}{2^m}$
1	$\frac{1}{2^m}$
2	$\frac{1}{2^m}$
.	.
.	.
.	.
$2^m - 3$	$\frac{1}{2^m}$
$2^m - 2$	$\frac{1}{2^m}$
$2^m - 1$	$\frac{1}{2^m}$

Fonte: Próprio Autor.

Caso o circuito

2.1.2 Contagem de uso

Para levantar de forma mais precisa a probabilidade de uma entrada ser utilizada, pode ser feita partindo de medições, ou seja, durante um intervalo de tempo de uso do

circuito Δt_{ci} e o tempo de uso de uma certa entrada Δt_{x_j}, por fim, a probabilidade de uma entrada qualquer seria dada por:

$$P(X = x_j) = \frac{\Delta t_{x_j}}{\Delta t_{ci}} \tag{2.2}$$

Futuramente a ferramenta terá outras possibilidades de funcionamento, assim, será possível utilizá-la com essa aferição de probabilidade caso seja requerido pelo usuário.

2.2 Probabilidade das faltas

Como dito anteriormente, cada porta lógica tem sua probabilidade q_i de não inverter a saída, devido a ocorrência de um evento aleatório de falta, onde i representa a porta lógica, variando de 1 até w(quantidade de portas lógicas no circuito). Sendo modelados de forma independente a probabilidade de cada porta lógica, temos por exemplo, que a probabilidade de todas estarem corretas seria $q_1.q_2.q_3...q_n$, no caso de queremos a probabilidade de uma porta estar errada a sua probabilidade se torna $1 - q_i$ e se quisessemos ter a probabilidade de somente a primeira estar errada seria $(1 - q_1).q_2.q_3...q_n$, sendo generalista dessa forma com as outras probabilidades, essas probabilidades têm forma similar à probabilidade binomial. (Arfken; Weber, 2005)

2.2.1 Portas com probabilidades iguais

Podemos então fazer uma aproximação para as probabilidades q_i, sendo então $q_1 = q_2 = q_3 = ... = q_n = q$. Dessa forma para alguns vetores de falta teremos como mostra a tabela abaixo a probabilidade.

Na tabela temos k sendo a quantidade de faltas simultâneas, ou seja, se $k = 2$, temos que são duas portas que estão com os bits trocados ao mesmo tempo. o valor l varia dentro de um k, ou seja, para um valor de k existem $\binom{w}{k}$ vetores de faltas possíveis. Temos também o vetor de faltas $e_{w:k}(l)$, ou seja, dado a quantidade de faltas simultâneas k e l determinando qual a posição dos bits de falta no vetor, ou seja, se l = 1, os bits '1' serão todos na esquerda, após serem deslocados como será explicado no bloco de geração de faltas, obtém-se quando l = $\binom{w}{k}$, todos os bits '1' na direita, tendo assim, varrido o vetor por completo.

Notemos que quanto maior for a quantidade de faltas simultâneas diminuirá expressivamente a probabilidade de ocorrência da mesma e diminui também a probabilidade da saída ser correta, dessa forma, na ferramenta também haverá um parâmetro de quantidade

Tabela 2 – Probabilidade igual para cada porta.

k	l	$e_{w:k}(l)$	$P(e_{w:k}(l))$
0	$\binom{w}{0}$	000...00	q^w
1	1	100...00	$(1-q).q^w$
	2	010...00	$(1-q).q^w$
	3	001...00	$(1-q).q^w$
	.	.	.
	.	.	.
	.	.	.
	$\binom{w}{1}$	000...01	$(1-q).q^w$
2	1	110...00	$(1-q)^2.q^{w-1}$
...	...	...	...
w	$\binom{w}{w}$	111...11	$(1-q)^w$

Fonte: Próprio Autor.

máxima de faltas simultâneas, ou seja, a quantidade de faltas simultâneas aumentará de 1 até esse parâmetro, a depender do valor que o projetista escolha.

2.2.2 Portas com probabilidades diferentes

Caso não seja utilizada a aproximação de que todas as portas tm a mesma probabilidade de falta, então temos a tabela abaixo.

Tabela 3 – Probabilidade diferente para cada porta.

k	l	$e_{w:k}(l)$	$P(e_{w:k}(l))$
0	$\binom{w}{0}$	000...00	$q_1.q_2.q_3...q_{n-1}.q_n$
1	1	100...00	$(1-q_1).q_2.q_3...q_{n-1}.q_n$
	2	010...00	$q_1.(1-q_2).q_3...q_{n-1}.q_n$
	3	001...00	$q_1.q_2.(1-q_3)...q_{n-1}.q_n$
	.	.	.
	.	.	.
	.	.	.
	$\binom{w}{1}$	000...01	$q_1.q_2.q_3...q_{n-1}.(1-q_n)$
2	1	110...00	$(1-q_1).(1-q_2).q_3...q_{n-1}.q_n$
...	...	...	...
w	$\binom{w}{w}$	111...11	$(1-q_1).(1-q_2).(1-q_3)...(1-q_{n-1}).(1-q_n)$

Fonte: Próprio Autor.

2.3 Modelo probabilístico binomial da confiabilidade

Com as probabilidades dos eventos, temos como calcular a confiabilidade do circuito digital. Como será mostrado nas seções seguintes, a confiabilidade será calculada da seguinte forma: o gerador de falta cria o vetor e coloca na saída, quando isso ocorre o circuito irá variar todas as entradas começando no 0 e indo até $2^m - 1$, quando colocar uma entrada serão comparadas as saídas dos dois circuitos (o circuito modelo de ouro que não possui falhas e o circuito com os injetores de faltas), quando varrer todo o vetor de entradas será gerado o próximo vetor de falta e repete o processo até acabarem os vetores de faltas, ou seja, quando a quantidade de faltas simultâneas for igual ao parâmetro de quantidade máxima de faltas simultâneas. Por fim, é uma das formas de calcular o seguinte somatório duplo.

$$R = \sum_{j=0}^{2^m-1} P(x_j, y = correta) \tag{2.3}$$

$$= \sum_{j=0}^{2^m-1} P(x_j) P(y = correta | x_j) \tag{2.4}$$

Como a $P(y = correta|x_j)$ depende da entrada e já vimos nas tabelas 7 e 3 como calcular a probabilidade para cada falta, ou seja, para sabermos $P(y = correta|x_j)$ basta somar quando a saída do modelo sem faltas for igual a saída dos modelos com faltas, como segue abaixo. Caso dado um vetor de falta f se $y(x_j, e(1)) == y(x_j, e(f))$ então é atribuído valor 1, caso contrário será atribuído valor zero.

Podemos reorganizar o somatório do vetor de faltas f em parâmetros já mencionados que são k e l. Após isso, temos que levar em consideração a probabilidade do vetor de entradas sendo igual para cada vetor e rearrumar a equação, como mostra na equação abaixo.

$$R = \sum_{j=0}^{2^m-1} P(x_j) \sum_{f=1}^{2^w} [y(x_j, e(1)) == y(x_j, e(f))] * P(e_{w:k}(l)) \tag{2.5}$$

$$= \sum_{j=0}^{2^m-1} P(x_j) \sum_{k=0}^{w} \sum_{l=1}^{\binom{w}{k}} [y(x_j, e_{w:0}) == y(x_j, e_{w:k}(l))] * P(e_{w:k}(l)) \tag{2.6}$$

$$= \sum_{j=0}^{2^m-1} \frac{1}{2^m} \sum_{k=0}^{w} \sum_{l=1}^{\binom{w}{k}} [y(x_j, e_{w:0}) == y(x_j, e_{w:k}(l))] * P(e_{w:k}(l)) \tag{2.7}$$

$$= \frac{1}{2^m} \sum_{j=0}^{2^m-1} \sum_{k=0}^{w} \sum_{l=1}^{\binom{w}{k}} [y(x_j, e_{w:0}) == y(x_j, e_{w:k}(l))] * P(e_{w:k}(l)) \tag{2.8}$$

Da tabela 4, ao somarmos a última coluna e dividirmos por 2^m obtemos a confiabilidade do circuito, ou seja, o algoritmo descreve da seguinte forma: primeiro circular o vetor de entrada, e só então, irá gerar o próximo vetor de falta. Dessa forma, iremos gerar todos $y(x_j, e_{w:0}) == y(x_j, e_{w:k}(l))$ para cada k, assim, basta para cada k contar a quantidade de igualdades verdadeiras e múltiplicar por $q^{w-k}(1-q)^k$. ao obter para cada k essa múltiplicação bastar somar todas e dividir por 2^m, obtendo assim, a confiabilidade pela ferramenta, . Como q é a confiabilidade da porta lógica, essa pode ser uma variável de uma função e ser traçado um gráfico da confiabilidade de circuito em função de q.

Esse gráfico seria da seguinte forma polinomial:

$$\begin{aligned} R(q) = \tfrac{1}{2^m}[q^w.count[0] + q^{w-1}.(1-q).count[1] + q^{w-2}.(1-q)^2 + ... \\ + q.(1-q)^{w-1}.count[w-1] + (1-q)^w.count[w]] \end{aligned} \tag{2.9}$$

count[i] é o contador de quantas vezes as saídas dos módulos comparados são iguais para a mesma quantidade de faltas simultâneas. Note que count[0] não precisa ser contado e é igual a quantidade de vetores de entradas (2^m). Podemos reescrever como o somatório abaixo.

$$count[i] = \sum_{l=1}^{\binom{w}{k}} [y(x_j, e_{w:0}) == y(x_j, e_{w:k}(l))] \tag{2.10}$$

$$\begin{aligned} R(q) = \tfrac{1}{2^m}[q^w.2^m + q^{w-1}.(1-q).count[1] + q^{w-2}.(1-q)^2 + ... \\ + q.(1-q)^{w-1}.count[w-1] + (1-q)^w.count[w]]) \end{aligned} \tag{2.11}$$

$$\begin{aligned} = q^w + [q^{w-1}.(1-q).count[1] + q^{w-2}.(1-q)^2 + ... \\ + q.(1-q)^{w-1}.count[w-1] + (1-q)^w.count[w]] \end{aligned} \tag{2.12}$$

$$= q^w + \tfrac{1}{2^m} \sum_{i=1}^{w} q^{w-i}.(1-q)^i.count[i] \tag{2.13}$$

2.4 Confiabilidade Aproximada

Como o valor de $q^{w-i}.(1-q)^i$ diminui com o aumento de i devido a ser q > 0,5 e a quantidade de vezes que a saída será igual possivelmente também diminuirá, pois

Tabela 4 – Sequência do comparador.

k	l	$e_{w:k}(l)$	x_j	$P(e_{w:k}(l))$	$y(x_j, e_{w:0}) == [y(x_j, e_{w:k}(l))].P(e_{w:k}(l))$
0	$\binom{w}{0}$	000...00	0	q^w	q^w
			1	q^w	q^w
			...		...
			2^m-1	q^w	q^w
1	1	100...00	0	$(1-q).q^{w-1}$	$(1-q).q^{w-1}.[y(0, e_{w:0}) == y(0, e_{w:1}(1))]$
			1	$(1-q).q^{w-1}$	$(1-q).q^{w-1}.[y(1, e_{w:0}) == y(1, e_{w:1}(1))]$
			...		...
			2^m-1	$(1-q).q^{w-1}$	$(1-q).q^{w-1}.[y(2^m-1, e_{w:0}) == y(2^m-1, e_{w:1}(1))]$
	2	010...00	0	$(1-q).q^{w-1}$	$(1-q).q^{w-1}.[y(0, e_{w:0}) == y(0, e_{w:1}(2))]$
			1	$(1-q).q^{w-1}$	$(1-q).q^{w-1}.[y(0, e_{w:0}) == y(0, e_{w:1}(2))]$
	...	...	...	$(1-q).q^{w-1}$	...
	$\binom{w}{1}$	000..01	2^m-1	$(1-q).q^{w-1}$	$(1-q).q^{w-1}.[y(2^m-1, e_{w:0}) == y(2^m-1, e_{w:1}(\binom{w}{1}))]$
...	...	...	...	...	...
w	$\binom{w}{w}$	111...11	0	$(1-q)^w$	$(1-q)^w.[y(0, e_{w:0}) == y(0, e_{w:w}(\binom{w}{w}))]$
			1	$(1-q)^w$	$(1-q)^w.[y(1, e_{w:0}) == y(1, e_{w:w}(\binom{w}{w}))]$
			...		...
			2^m-1	$(1-q)^w$	$(1-q)^w.[y(2^m-1, e_{w:0}) == y(2^m-1, e_{w:w}(\binom{w}{w}))]$

Fonte: Próprio Autor.

com o aumento da quantidade de faltas simultâneas é menos provável que a saída seja correta. Então podemos truncar o somatório definindo o parâmetro quantidade máxima de faltas simultâneas (F) como sendo menor que w, acelerando o processo e tendo ainda uma confiabilidade próxima do real. Sendo representado na seguinte equação:

$$R(q) \approx R_{ap}(q) = q^w + \frac{1}{2^m} \sum_{i=1}^{F} q^{w-i}.(1-q)^i.count[i] \tag{2.14}$$

Notemos que q^w é a confiabilidade R_L livre de faltas e a outra parte da soma é R_{FM} que é a parte de mascaramento das faltas, essas notações podem ser encontradas em alguns documentos como o do (FRANCO et al., 2008). A equação mais importante e que resume todo o funcionamento da ferramenta é a equação 2.14.

2.4.1 Erro de aproximação

Como a série será truncada, para valores diferentes de q e microarquiteturas diferentes, teremos aproximações diferentes, então temos que estimar o percentual de confiabilidade contido na confiabilidade aproximada, ou seja até o nosso parâmetro F de quantidade simultâneas máxima de faltas, podendo variar de um até w(w - quantidade de portas no circuito). Definamos da seguinte forma o percentual:

$$\epsilon(q) = \frac{R_{ap}(q)}{R(q)} \tag{2.15}$$

como queremos saber o quanto está contido até um F, podemos atribuir que a microarquitetura nunca obtém uma saída errada, ou seja, o contador count[i] pode ser reecrito.

$$count[i] = \alpha(i)\binom{w}{i}2^m \tag{2.16}$$

Sendo $\alpha(i)$ o percentual de contagem para i faltas simultâneas, quantas saídas estão corretas, lembremos que quando $i = 0$, temos sempre $\alpha(0) = 1$, pois nenhuma porta lógica tem seu sinal trocado. Dessa forma dada a condição de que temos uma microarquitetura super tolerante a faltas, teremos nesse que $\alpha(i) = 1$ para todo $i \in [0, w]$. Dada essas condições teremos que R se tornará.

$$R(q) = q^w + \tfrac{1}{2^m}\sum_{i=1}^{w} q^{w-i}.(1-q)^i.count[i] \tag{2.17}$$

$$= q^w.\frac{2^m}{2^m} + \tfrac{1}{2^m}\sum_{i=1}^{w} q^{w-i}.(1-q)^i.count[i] \tag{2.18}$$

$$= \tfrac{1}{2^m}\sum_{i=0}^{w} q^{w-i}.(1-q)^i.count[i] \tag{2.19}$$

$$= \tfrac{1}{2^m}\sum_{i=0}^{w} q^{w-i}.(1-q)^i.\binom{w}{i}.2^m \tag{2.20}$$

$$= \sum_{i=0}^{w}\binom{w}{i}.q^{w-i}.(1-q)^i \tag{2.21}$$

$$\tag{2.22}$$

Notemos que essa forma é de um binômio de Newton, dessa forma podemos reescrever a

confiabilidade como abaixo.

$$R(q) = \sum_{i=0}^{w} \binom{w}{i} . q^{w-i} . (1-q)^i \tag{2.23}$$
$$= (q + 1 - q)^w \tag{2.24}$$
$$= (1)^w \tag{2.25}$$
$$= 1 \tag{2.26}$$
$$\tag{2.27}$$

Assim, o percentual fica resumido a seguinte função:

$$\epsilon(q) = \frac{R_{ap}(q)}{1} \tag{2.28}$$
$$= q^w + \tfrac{1}{2^m} \sum_{i=1}^{F} q^{w-i} . (1-q)^i . count[i] \tag{2.29}$$
$$= \sum_{i=0}^{F} \binom{w}{i} . q^{w-i} . (1-q)^i \tag{2.30}$$

3 Desenvolvimento da ferramenta

A *FPGA* tem um papel importante no paradigma de computação desta ferramenta, pois sua arquitetura tem a possibilidade de a cada ciclo de *clock* executar uma ação específica para resolver o problema (possibilidade de crescimento de paralelismo de dados via alteração da arquitetura) e as formas de computação por computadores não têm essa possibilidade, pois têm um único conjunto de instruções que têm de ser seguido de forma rigorosa, ou seja, uma microarquitetura expecífica simplificada é capaz de solucionar diretamente o problema sem a preocupação de quanto de memória será utilizada como nas soluções via computador.

Para desenvolver esta ferramenta foi necessário dividí-la em partes para solucioná-las de forma mais simples e prática. Na figura 4 temos a visualização do diagrama com os respectivos módulos.

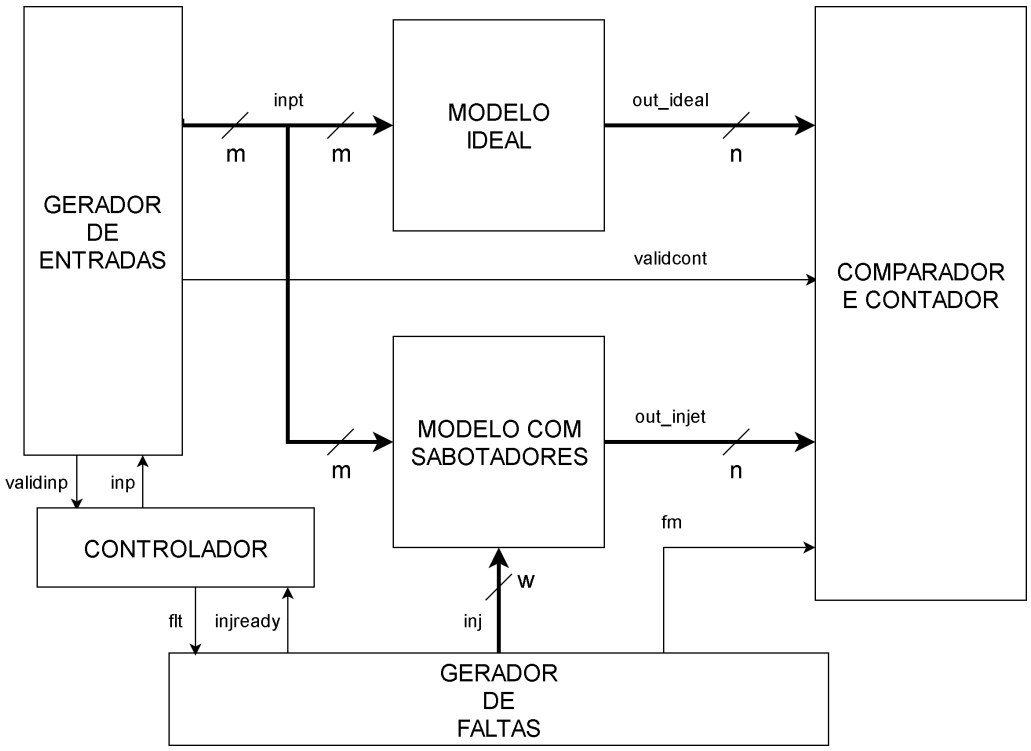
3.1 Modelo ideal

Este é o modelo original, independente da forma como foi gerado, esse arquivo será, em geral, a própria *netlist* do circuito, podendo ser em outra linguagem também, entranto como o modelo com sabotadores precisará alterar a *netlist*, então fica mais simples utilizá-la como o modelo de referências, ou seja, que não apresentará falhas.

3.2 Modelo com sabotadores

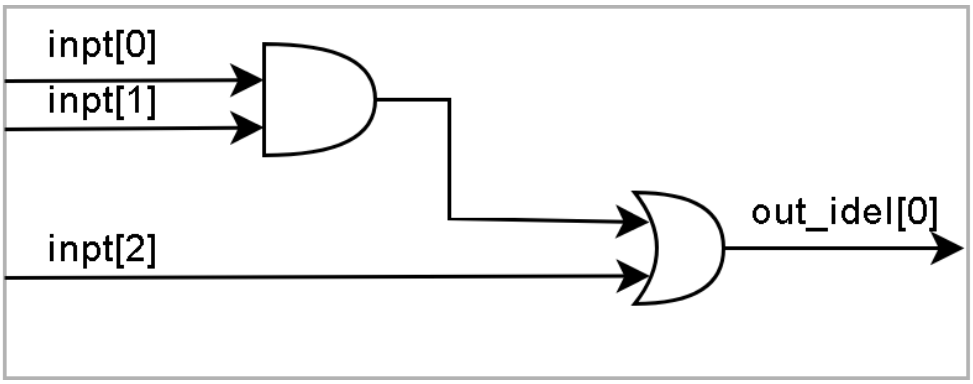
Como o problema de faltas transitórias em um circuito digital é simplementes a inversão do bit de sinal, então, basta simplesmente inverter ou não a saída de uma combinação lógica de forma controlada. Esse controle será realizado por um sinal externo, como já foi explicado quando o mesmo for "1"inverterá a saída e quando for "0"manterá

Figura 4 – Diagrama geral da ferramenta.



Fonte: Próprio Autor.

Figura 5 – *Netlist* básica ideal.

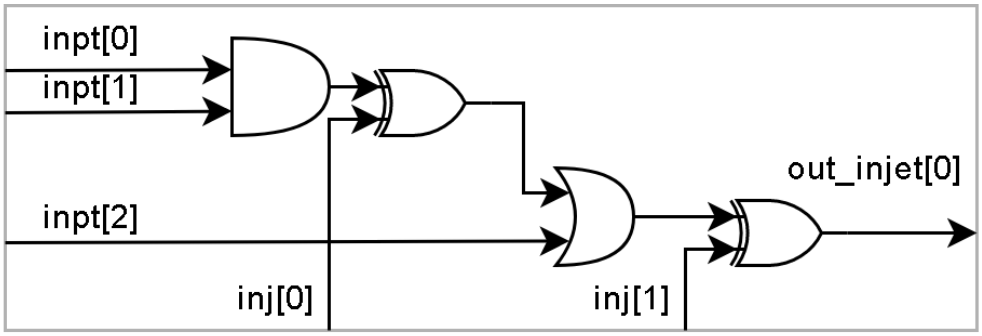


Fonte: Próprio Autor.

como estava.

A implementação desse modelo é realizada alterando a netlist por um script escrito em *Python*, realizando as eventuais alterações necessárias para inserir os sabotadores (portas xor após cada porta lógica) e colocar uma das entradas dessa porta lógica como sinal de entrada do bloco, sendo o diagrama de blocos segundo a figura 6.

Figura 6 – *Netlist* básica com injetores.



Fonte: Próprio Autor.

3.3 Gerador de faltas

Este módulo gerador 6.1.2 tem a função de gerar o vetor de bits. Cada bit desse vetor será conectado à porta lógica que fará ou não a inversão da porta lógica do circuito original, essa inversão foi descrita na seção 3.2. o vetor tem uma quantidade fixa e parametrizada de bits, que serão iguais a quantidade de circuitos combinacionais da *netlist*. A quantidade simultânea de bits com valor 1, ou seja nível lógico alto, serão iguais a quantidade de faltas simultâneas. A quantidade de faltas simultâneas inicia em valor 0 e vai até o número máximo de faltas a ser computado definido por parâmetro.

Ao ser definido a quantidade máxima de faltas e a quantidade de portas lógicas, esse circuito cria um vetor com a quantidade de elementos iguais a quantidade máxima de faltas e a quantidade de bits de cada elemento é a quantidade de portas lógicas, [INJECTOR-1:0] vector [FAULTS]. Dessa forma o vector[0] inicia com um bit 1 mais significativo e esse vai sendo deslocado até o final, essa é a primeira falta simultânea, como mostra a tabela abaixo.

Quando esse processo terminar, teremos que iniciar o processo para duas faltas simultâneas, dessa forma o vector[1] é iniciado com seu bit mais significativo em 1 e o vector[0] recebe o valor de vector[1] deslocado de 1 para a direita (vector[1] » 1), assim, a saída se torna uma "ou"desses dois vetores. Da mesma forma que ocorreu para uma falta simultânea, para quantidades maiores de simultâneas teremos sempre a tentativa de deslocar o vector[0] para a direta, quando não for mais possível e a condição de ter terminado ainda não for atingida (o término ocorre quando todos os vetores não podem mais serem deslocados para a direita, ou seja, o vector[k-1], sendo k-1 o maior valor de vetor ativo no momento, for igual a 1«Fm), teremos que tentar deslocar o vetor mais acima, nesse caso, o vector[1] é o vetor acima do zero, então será deslocado o vector[1] de

Tabela 5 – Vetor de faltas para 1 falta simultânea.

k	l	$vector[0]$
1	1	100...00
	2	010...00
	3	001...00
	...	...
	$\binom{w}{1}$	000...01

Fonte: Próprio Autor.

1 para a direita e o vector [0] irá receber o novo valor de vector [1] deslocado de 1 para a direita como mostra na tabela abaixo.

Tabela 6 – Vetor de faltas para 2 faltas simultâneas.

k	l	$vector[1]$	$vector[0]$	$out = vector[1] \| vector[0]$
2	1	1000...00	0100...00	1100...00
	2	1000...00	0010...00	1010...00
	3	1000...00	0001...00	1001...00
	...	...		...
	w-1	1000...00	0000...01	1000..01
	w	0100...00	0010...00	0110..00
	w+1	0100...00	0001...00	0101..00
	2*w-4	0100...00	0000...10	0100..10
	2*w-3	0100...00	0000...10	0100..01
	2*w-2	0010...00	0001...00	0011..00
	...	...	...	...
	$\binom{w}{2}$	0010...10	0000...01	0000...11

Fonte: Próprio Autor.

De forma recursiva, temos que quando o vector[0] não puder mais ser deslocado para a direita, será analisado se o vector[1] pode ser, caso também não possa, será analisado se o vector[2] pode, e assim, recursivamente até encontrar algum vetor que possa ser deslocado de 1 para para a direita, quando for encontrado todos os vectores abaixo dele, de índice menor, serão alocador sucessivamente com seus bits 1 a direita do mesmo, exemplo:

Dessa forma o algorítmo implementado em hardware é recursivo e escalável para toda quantidade máxima de faltas simultâneas e quantidade de portas lógicas a serem controladas.

Tabela 7 – Vetor de faltas para 3 faltas simultâneas.

k	l	$vector[2]$	$vector[1]$	$vector[0]$	$out = vector[1] \| vector[0]$
3	$\binom{w-1}{2}$	1000...00	0000...10	0000...01	1000...11
	$\binom{w-1}{2} + 1$	0100...00	0010...00	0001...00	0111...00

Fonte: Próprio Autor.

3.4 Gerador de entradas

O modulo gerador de entradas é o bloco encarregado de variar, como foi explicado na forma de calcular a confiabilidade, ou seja, o vetor inicializa em 0 e varre de um em um até chegar em $2^m - 1$, volta a 0 e recomeça o processo. O módulo é bem simples sendo controlado pelo bloco controlador, esse será explicado depois.

3.5 Controlador

Este bloco é incubido de controlar o funcionamento do circuito como um todo, ele faz a ligação entre o módulo gerador de faltas e o módulo de entrada, ou seja, ele é quem determina se o circuito irá alterar o vetor de faltas ou a entrada. Nesse trabalho foi utilizado a varredura de entradas dentro de cada vetor, ou seja, variam todas as entradas e depois é que muda o vetor de faltas, após isso será gerado o próximo vetor de faltas e serão varidas todas as entradas novamente, se repetindo até terminar todo o processo.

3.6 Comparador e Contador

Este bloco receberá na sua entrada os vetores de saída do modelo ideal e do modelo com sabotadores, a quantidade de faltas simultâneas e um sinal advindo do gerador de entradas, para assim, poder incrementar seu contador.

Esses contadores terão a quantidade de bits necessárias para contar o máximo que precisa em cada falta simultânea, ou seja, $m + floor(\log_2 \binom{w}{i})$, onde i será o valor que dará a maior quantidade de bits. E terá a quantidade máxima de faltas como quantidade de elementos.

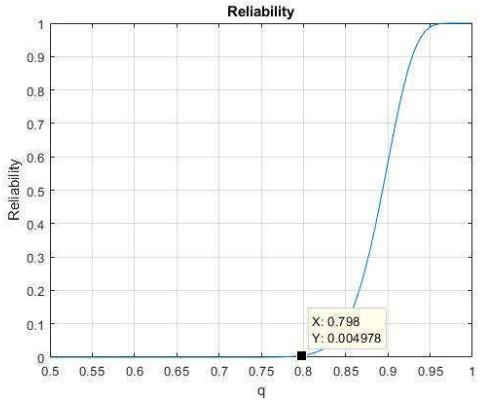
O funcionamento é bem simples, quando o vetor de entradas e o vetor de entradas simultâneas alteram as saídas dos modelos ideal e com faltas, será incrementado de um o vetor $count[F_m]$ (F_m é o número de faltas simultâneas decrementado de um) se as saídas forem diferentes, caso contrário mantém o mesmo valor que estava.

3.7 Quantidade máxima de faltas e sua confiabilidade

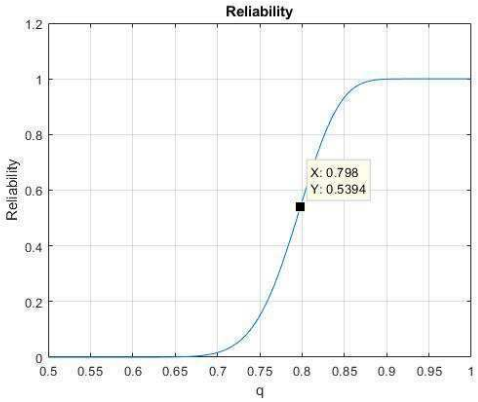
Caso o usuário queira diminuir o tempo gasto na contagem, basta calcular para a confiabilidade q da suas portas lógicas e a quantidade w de portas lógicas no circuito, uma quantidade máxima de faltas que aproxime bem os resultados como foi mostrado na seção 2.4.1. Abaixo segue o percentual ϵ para um circuito genério de 100 portas utilizando apenas 10, 20, 30, 40 faltas simultâneas e analisando para portas com confiabilidade em torno de 0,8. Esses resultados foram obtidos utilizando o script do apêndice 6.2.2.

Figura 7 – Efeitos do truncamento da série.

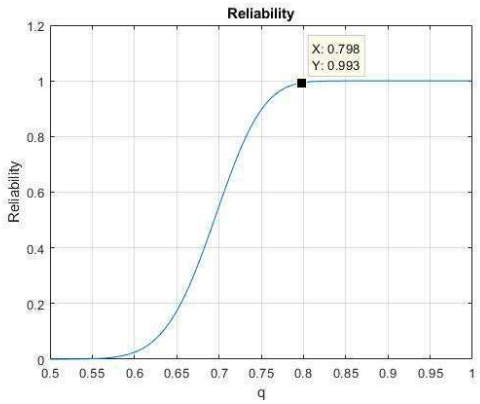
(a) 10 faltas simultâneas.



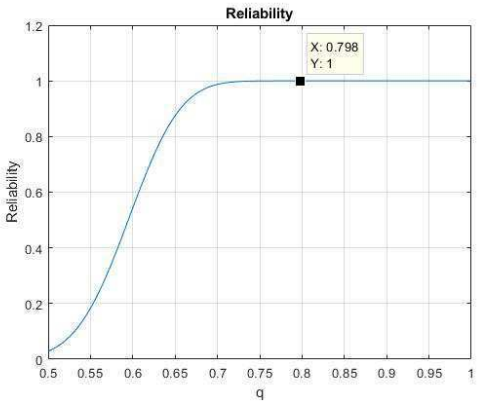
(b) 20 faltas simultâneas.



(c) 30 faltas simultâneas.



(d) 40 faltas simultâneas.



Fonte: Próprio Autor.

Notemos que nesse caso como mostra a figura 7c se tivessemos um circuito com essas características (número de portas lógicas) e o mesmo não invertesse a saída independente da falta, teríamos aproximadamente 99,3% do total da confiabilidade nessas 30 primeiras faltas simultâneas que é uma boa aproximação, dado que os contadores não serão máximos ($\alpha(i) \neq 1$, para todo i > 0), dessa forma utilizando apenas até 30 faltas simultâneas seria

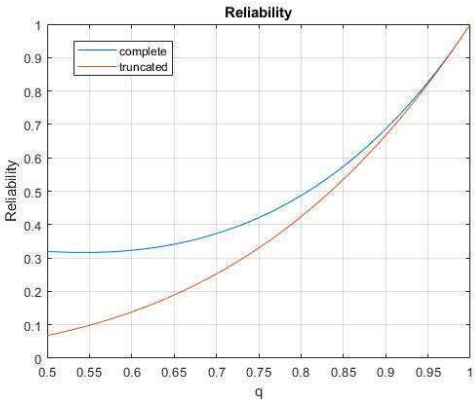
suficiente para convergir para o resultado, caso o usuário queira utilizar mais valores para aumentar a precisão, fica a cargo do mesmo decidir.

3.8 Confiabilidade

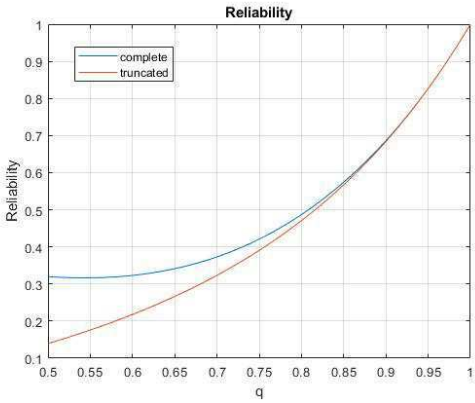
Para calcular a confiabilidade basta atribuir os valores dos contadores após a quantidade de entradas possíveis no vetor count do script 6.2.3, exemplo: count = [64 75 147 154 192 23]; Com 5 portas e esses contadores, nas figuras 8 temos a confiabilidade (azul) e comparando a confiabilidade com as confiabilidade truncadas.

Figura 8 – Confiabilidade completa e aproximada.

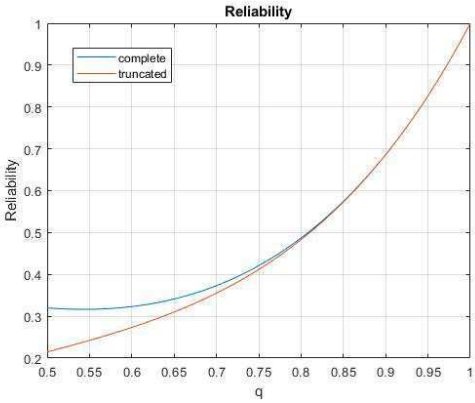
(a) 1 faltas simultâneas.



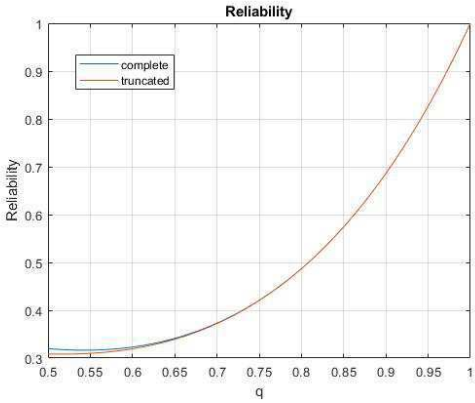
(b) 2 faltas simultâneas.



(c) 3 faltas simultâneas.



(d) 4 faltas simultâneas.



Fonte: Próprio Autor.

Note que a convergência depende tanto do 'q' do circuito, quanto da quantidade máxima de faltas simultâneas.

4 Resultados

Esta seção contém os resultados colhidos ao utilizar a ferramenta para avaliar a confiabilidade dos dois circuitos já citados, o c17 do iscas 85 e o somador completo de 8 bits.

4.1 c17.v

A obtenção da *netlist* pode ser feita via o site[1] oficial.

4.1.0.1 Inserção de sabotadores

O primeiro passo é alterar a *netlist* e inserir os sabotadores utilizando o script 6.3.1, ao utilizar, altere o nome do arquivo, no *script*, onde tem o primeiro *filename* deve-se colocar o nome do arquivo original, neste caso, "c17.v", no segundo *filename* o nome do arquivo a ser gerado, neste caso, foi utilizado "c17inj.v". Ao realizar essas alterações e executá-lo, obtemos o arquivo com o nome "c17inj.v", ao abrí-lo altera-se o nome do módulo para c17inj e ficam prontos para serem utilizados.

A netlist original e a alterada pelo script estão mostradas na figura 9.

4.1.0.2 Instanciação Geradores, controlador e contador

Para realizar essa instanciação no módulo do topo, bastou utilizar o código do apêndice 6.1.1 o qual consta a base já interligada com os nomes dos sinais utilizados e os que serão gerados na parte de instanciar os modelos, ideal e sabotado. Deve-se ligar o sinal de '*clock*' a algum dos geradores de '*CLOCK*' da *fpga*

[1] link para acesso das netlists *iscas* 85: http://www.pld.ttu.ee/ maksim/benchmarks/iscas85/verilog/

Figura 9 – *Netlists* do circuito a ser analisado

(a) *Netlist* original(ideal).

```
module c17 (N1,N2,N3,N6,
   N7,N22,N23);

input N1,N2,N3,N6,N7;

output N22,N23;

wire N10,N11,N16,N19;

nand NAND2_1 (N10, N1, N3);
nand NAND2_2 (N11, N3, N6);
nand NAND2_3 (N16, N2, N11);
nand NAND2_4 (N19, N11, N7);
nand NAND2_5 (N22, N10, N16);
nand NAND2_6 (N23, N16, N19);

endmodule
```

(b) *Netlist* com sabotadores.

```
module c17inj (N1,N2,N3,N6,
      N7,N22,N23
,E0,E1,E2,E3,E4,E5);
input E0,E1,E2,E3,E4,E5;
wire I0,I1,I2,I3,I4,I5;
input N1,N2,N3,N6,N7;
output N22,N23;
wire N10,N11,N16,N19;
nand NAND2_1 (I0, N1, N3);
xor xor_0 (N10, I0, E0);
nand NAND2_2 (I1, N3, N6);
xor xor_1 (N11, I1, E1);
nand NAND2_3 (I2, N2, N11);
xor xor_2 (N16, I2, E2);
nand NAND2_4 (I3, N11, N7);
xor xor_3 (N19, I3, E3);
nand NAND2_5 (I4, N10, N16);
xor xor_4 (N22, I4, E4);
nand NAND2_6 (I5, N16, N19);
xor xor_5 (N23, I5, E5);
endmodule
```

Fonte: Próprio Autor.

4.1.0.3 Instanciação *netlists*

Usando os scripts 6.3.2 e 6.3.3 para entradas e saídas não empacotadas, ou seja, teremos uma ligação para cada fio de entrada e de saída, na condição da netlist ter entradas empacotadas, basta apagar a parte desempacotada gerada pelo *script* e inserir manualmente. obtemos a intanciação abaixo.

Figura 10 – Instanciação dos módulos

(a) *Netlist* original(ideal).

```
c17 DUTideal(inpt[0], inpt[1], inpt[2], inpt[3], inpt[4],
out_ideal[0], out_ideal[1]
);
```

(b) *Netlist* com sabotadores.

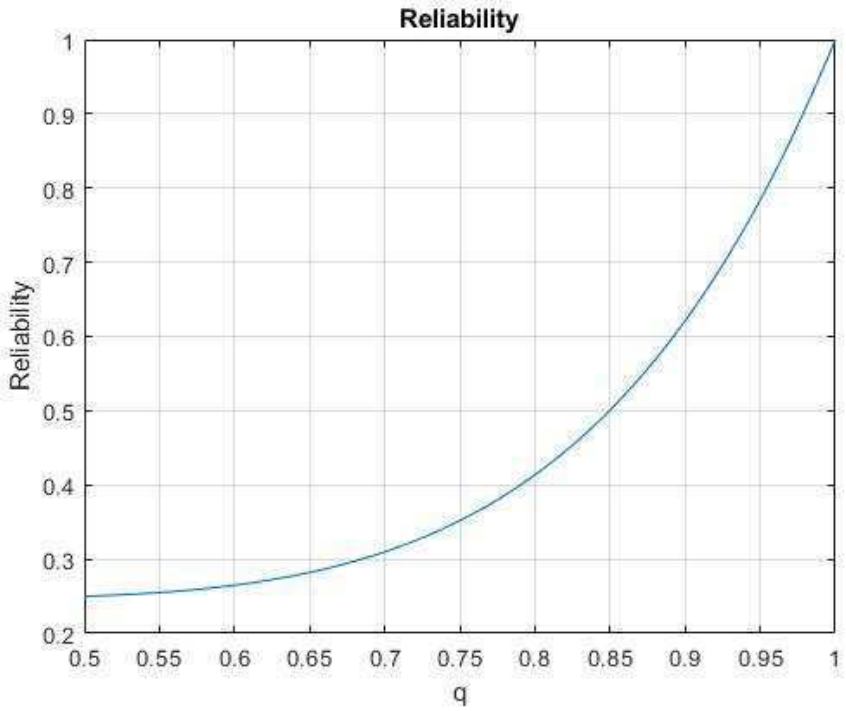
```
c17inj DUTinj(inpt[0], inpt[1], inpt[2], inpt[3], inpt[4],
out_injet[0], out_injet[1],
inj[0], inj[1], inj[2], inj[3], inj[4], inj[5]
);
```

Fonte: Próprio Autor.

4.1.0.4 Calculando confiabilidade

Com os valores dos contadores após o encerramento do processo(Se o circuito for pequeno, pode-se utilizar os displays da *FPGA* para visualizar os valores dos contadores, no caso mais geral deve-se instanciar um módulo de *UART* e guardar em um registrador de deslocamento o valor a ser enviado de forma serial), podemos gerar o gráfico de confiabilidade, abaixo temos usando o script do *matlab* 6.2.1

Figura 11 – Confiabilidade circuito c17.



Fonte: Próprio Autor.

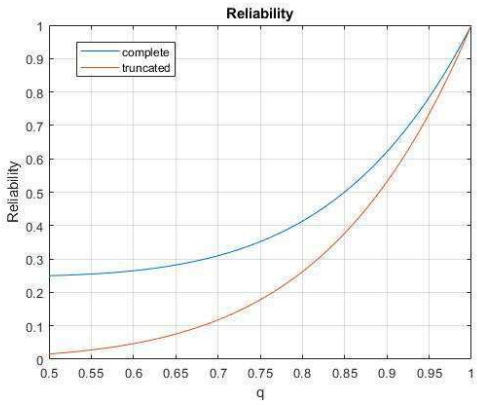
Comparando a confiabilidade com as confiabilidade truncadas, temos também utilizando o script 6.2.3.

4.2 Somador 8 bits

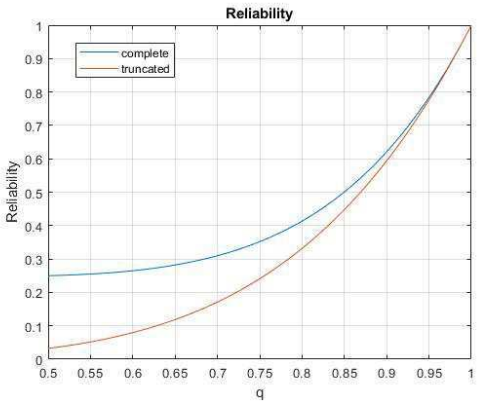
Utilizando um somador 8 bits simples, e realizando os mesmos passos anteriores (inserção de sabotadores, intanciação no modelo base com parâmetros atualizados e utilizando um módulo cp2102 para conexão *UART*), o código da implementação da *UART*, estão nos anexos 6.1.6 e 6.1.7. Temos para até três faltas simultâneas a seguinte curva de confiabilidade, segundo a figura 13.

Figura 12 – Confiabilidade completa e aproximada circuito c17.

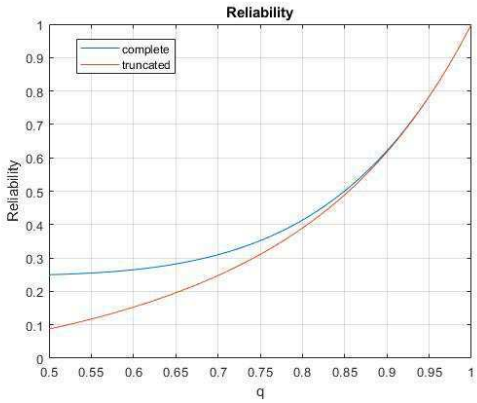
(a) 0 faltas simultâneas.



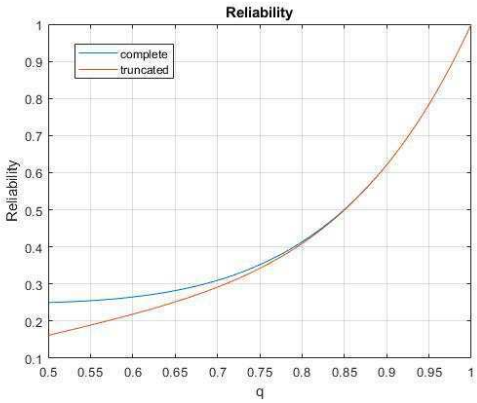
(b) 1 faltas simultâneas.



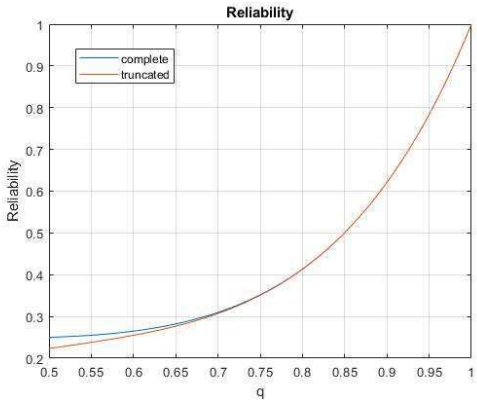
(c) 2 faltas simultâneas.



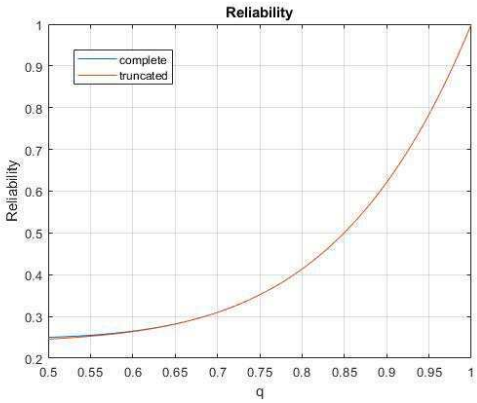
(d) 3 faltas simultâneas.



(e) 4 faltas simultâneas.

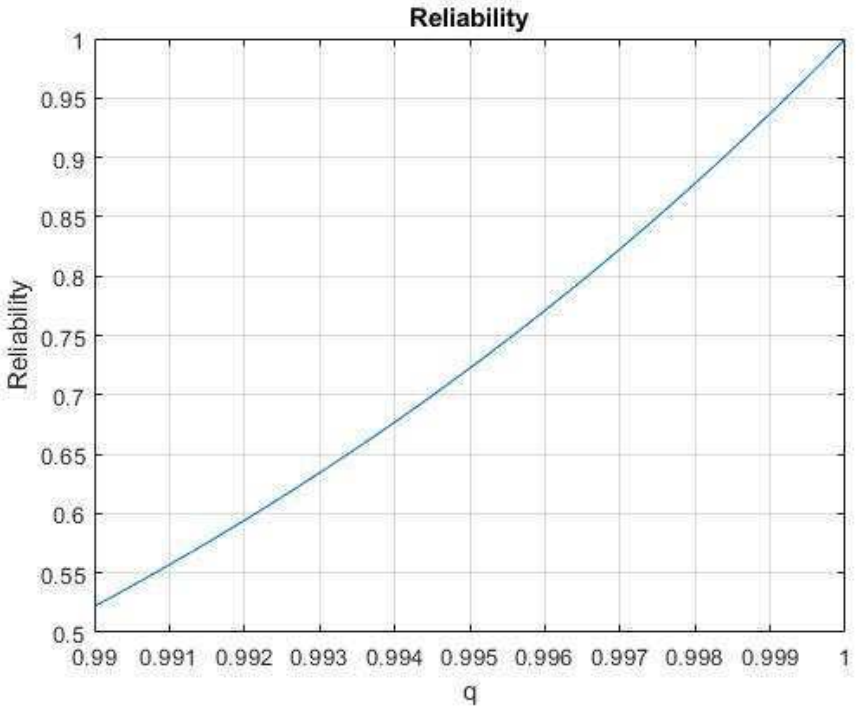


(f) 5 faltas simultâneas.



Fonte: Próprio Autor.

Figura 13 – Alcance (q) de 0,99 até 1 para somador 8 bits.



Fonte: Próprio Autor.

5 Conclusões

Ficou evidente que estimar a confiabilidade de um circuito combinacional é importante e a ferramenta tem sua devida utilidade, mesmo que ainda tenha algumas limitações de velocidade.

O trabalho objetivou a criação de uma plataforma onde fosse possível, de forma mais simplificada, obter a confiabilidade em relação a valores de probabilidade de células individuais, tornando-se a análise de diferentes circuitos uma simples troca de parâmetros e a mudança de sua *netlist*. Para circuitos grandes o esforço computacional e memória utilizada são muito grandes para fazer esse levantamento via computação comum, dessa forma, uma ferramenta em *FPGA* viabiliza e facilita o processo.

Nas aplicações foram utilizados circuitos menores, devido ao tempo que seria muito alto para realizar o cálculo para circuitos com muitas entradas, para o somador foi utilizada a condição de relaxamento de diminuição de quantidade máxima de faltas simultâneas, notando-se que é possível acelerar o processo ao diminuir a precisão, ou seja, caso o intuito seja comparar microarquiteturas diferentes, já é suficiente, dado que a probabilidade de ocorrência de múltiplas faltas simultâneas é baixa.

Como trabalhos futuros, planeja-se escalar a quantidade de núcleos de cálculos, ou seja, contador, circuito ideal e circuito com faltas, para aumentar a velocidade. Essa configuração de aceleração pode ser facilmente implementada, travando-se uma certa quantidade de pinos de entradas e diminuindo essas do gerador de entradas, ou seja, se dobrar a quantidade de núcleos (diminuir um pino de entrada, colando um em "0"e o circuito duplicado em "1"), diminui pela mentade o tempo necesário, assim sucessivamente. Planeja-se também alterar um pouco o circuito injetor de faltas, o mesmo terá uma espécie de fila em sua saída, ou seja, se o circuito de faltas não estiver sendo requisitado que gere outra falta, o mesmo ainda irá trabalhar gerando a próxima, para que quando o seja solicitado, já tenha no próximo ciclo de *clock* o vetor de faltas (Note que como roda todas as entradas enquanto está travado o injetor, adicionar somente um outro registrador de

saída já é suficiente).

Referências

Arfken, G. B.; Weber, H. J. *Mathematical Methods for physicists*. 6. ed. [S.l.]: Elsevier, Inc., 2005. Citado na página 6.

de Vasconcelos, M. C. R. et al. Reliability analysis of combinational circuits based on a probabilistic binomial model. In: *2008 Joint 6th International IEEE Northeast Workshop on Circuits and Systems and TAISA Conference*. [S.l.: s.n.], 2008. p. 310–313. ISSN null. Citado na página 3.

FRANCO, D. T. et al. Signal probability for reliability evaluation of logic circuits. *Elsevier Microelectronics Reliability*, v. 48, p. 1586–1591, Aug 2008. ISSN 0026-2714. Citado 2 vezes nas páginas 1 e 10.

Harris, D. M.; Harris, S. L. *Projeto Digital e Arquitetura de Computadores*. [S.l.]: Elsevier, Inc., 2013. Citado na página 1.

JúNIOR, G. G. dos S. *Robust design of deep-submicron digital circuits*. Tese (Doutorado) — Télécom ParisTech, 2012. Citado na página 1.

Patterson, D. A.; Hennessy, J. L. *Computer Organization and Design The Hardware/Software Interface: RISC-V Edition*. 1. ed. [S.l.]: Elsevier, Inc., 2017. Citado na página 1.

West, N. H. E.; Harris, D. M. *CMOS VLSI Design A Circuits and Systems Perspective*. 4. ed. [S.l.]: Addison-Wesley, 2011. Citado na página 1.

6 Apêndice

6.1 *SystemVerilog*

6.1.1 Instanciação geral de módulos(geradores, controlador e contador)

```
1  //=========================================================
2  //  PARAMETERs OF THE CIRCUIT
3  //=========================================================
4  parameter M        = 41;   // NUMBER OF INPUTS
5  parameter N        = 32;   // NUMBER OF OUTPUT
6  parameter INJECTOR = 202;  // NUMBER OF INJECTOR INSIDE THE CIRCUIT
7  parameter FAULTS   = 3;    // MAXIMUM NUMBER OF FAULTS
8  parameter COMB     = 1353400;    // MAXIMUM COMBINATION
9  //counter 1353400 is the maximum value of combination
10 //of ports and simultaneous faults Cw,(w/2) or (w-1)/2 or (w+1)/2,
11 //if use less than w/2 simultaneous
12 //faults, calculate the maximum combination of used simultaneous
13 //In this case 1353400 = 202!/(199!3!)
14
15 //=========================================================
16 //  REG/WIRE declarations
17 //=========================================================
18 wire rst;                //Reset for the modules
19 wire clock;              //Clock for the modules
20 wire injready;           //Injector vector is ready
21 wire flt;                //Stop signal from the injector generator
22 wire inp;                //Stop singal from the input generator
23 wire validinp;           //Signal of input module is ready
24 wire [$clog2(FAULTS)-1:0] fm;
```

```
25 //Signal = faults(simultaneous fault) - 1;
26 //telling the counter[fm]
27 wire validcont; //Signal that tells the counter
28 //that he can counts or not.
29 wire [M-1:0] inpt;  //Input Vector
30 wire [INJECTOR-1:0] inj;    //Injector Vector
31 wire [M + $clog2(COMB) - 1:0] count [FAULTS];
32 wire [M + $clog2(COMB) - 1:0] countr;
33 wire [N-1:0] out_ideal; //output of ideal    module
34 wire [N-1:0] out_injet; //output of injected module
35
36
37 //=========================================================
38 //  Modules
39 //=========================================================
40 // Fault Generator
41 injector #(INJECTOR, FAULTS) DUTinjec(
42 .clk(clock),
43 .reset(rst),
44 .stop(flt),
45 .out(inj[INJECTOR-1:0]),
46 .ready(injready),
47 .fm(fm)
48 );
49 // Input Generator
50 inpt_gen #(M)   inpuut(
51 .clk(clock),
52 .reset(rst),
53 .stop(inp),
54 .inpt_i(inpt),
55 .validinp(validinp),
56 .validcont(validcont)
57 );
58 // Control
59 cntrl intfc(
60 .clk(clock),
61 .reset(rst),
62 .ready(injready),
63 .validinp(validinp),
64 .flt(flt),
```

```
.inp(inp)
);
//Compute Relibability(counter)
compt_rel #(M, N, FAULTS) compt_rl(
.clk(clock),
.reset(rst),
.outp_i(out_ideal),
.outpf_i(out_injet),
.validflt(injready),
.fm(fm),
.count_i(count),
.validcont(validcont)
);
```

6.1.2 Módulo gerador de faltas (injector.sv)

```
//------------------------------------------------------
// Design Name : injector
// File Name   : injector.sv
// Function    : Generate every faults from one to the number
//of especified faults
// Coder       : Allender Vilar de Alencar
// version     : 0.6
//------------------------------------------------------

module injector #(parameter INJECTOR = 6, parameter FAULTS = 6)
  (input logic clk, reset, stop,
   output logic [INJECTOR-1:0] out,
   output logic ready,
        output logic [$clog2(FAULTS)-1:0] fm);

typedef enum logic [2:0] { A, B, C, D, E, F, IDLE} State;


logic Ready;

logic [INJECTOR-1:0] vector [FAULTS];
logic [$clog2(FAULTS)-1:0] k;
logic reject;

```

```
25 State currentState, nextState;
26
27 always_ff @(posedge clk)
28 begin
29   if (reset)
30     begin
31       currentState <= A;
32       nextState = A;
33     end
34   else currentState <= nextState;
35   case(currentState)
36     // Start the vector and fm(fault minus one = FAULTS - 1)
37     A: begin
38       fm = '0;
39       k = '0;
40       reject = '1;
41       vector[0] = 1 << (INJECTOR-1);
42       nextState = F;
43       out <= '0;
44       Ready <= '0;
45                   ready <= '0;
46     end
47     B: if(vector [0] > 1)
48       begin
49         vector[0] <= vector[0] >> 1;
50         nextState <= F;
51         currentState <= F;
52         Ready <= '0;
53                         ready <= '0;
54         out = '0;
55         k = fm;
56       end
57     else if(vector[fm] == 1 << fm)
58       begin
59         nextState = F; //olhar
60         k = fm;
61       end
62     else
63       begin
64         nextState = C;
```

```
65          reject = 1;
66          k = 0;
67        end
68      C:  begin
69            ready <= '0;
70            if(vector[k+1]>>1 == vector[k]) k = k + 1;
71      else
72        begin
73          nextState <= E;
74          currentState <= E;
75          Ready <= 0;
76          //vector[k+1] <= vector[k+1]>>1;
77          reject <= 0;
78        end
79            end
80      D:
81            if(Ready)
82        begin
83          Ready <= 0;
84          vector[fm] = 0;
85        end
86      else if(k<(fm + 1))
87        begin
88          vector[k] = 1 << (INJECTOR - fm + k - 1);
89          k <= k + 1;
90        end
91      else if(k == (fm + 1))
92        begin
93          nextState <= F;
94          currentState <= F;
95          k = fm;
96          out <= '0;
97          //reject <= '0;
98        end
99      E: begin
100           ready <= '0;
101           if(reject == 0)
102       begin
103         vector[k+1] = vector[k+1] >> 1;
104         reject = 1;
```

```
105         //if(k > 0) k--;
106       end
107     else if(k > 0)
108       begin
109         vector[k] = vector[k+1] >> 1;
110         if(k > 0)k--;
111       end
112     else if(k == 0)
113       begin
114         vector[k] = vector[k+1] >> 1;
115         nextState <= F;
116         k = fm;
117         out <= '0;
118         currentState <= F;
119       end
120           end
121     F:
122           if(fm + 1 == 1 && !Ready)
123       begin
124         out <= vector[0];
125         Ready <= 1;
126                         ready <= '1;
127         if(!stop)
128                                 if(vector[fm] == 1 << fm
129                                 && fm + 1 == FAULTS)
130                                           begin
131                                                   nextState <= IDLE;
132                                                   Ready <= 0;
133                                                   ready <= '0;
134                                           end
135           else if(vector[fm] == 1 << fm)
136             begin
137               nextState = D;
138               ready <= '0;
139             end
140         else begin
141                             nextState = B;
142                                  Ready <= 0;
143                                  ready <= '0;
144                         end
```

```
145       end
146     else if(fm + 1 == 2 && !Ready)
147       begin
148         out <= vector[0] | vector[1];
149         Ready <= 1;
150                           ready <= '1;
151         if(!stop)
152                             if(vector[fm] == 1 << fm
153                             && fm + 1 == FAULTS)
154                                     begin
155                                             nextState <= IDLE;
156                                             Ready <= 0;
157                                             ready <= '0;
158                                     end
159           else if(vector[fm] == 1 << fm)
160             begin
161               nextState = D;
162               ready <= '0;
163             end
164                             else begin
165                                             nextState = B;
166                                             Ready <= 0;
167                                             ready <= '0;
168                           end
169       end
170     else if(fm + 1 == 3 && !Ready)
171       begin
172         out <= vector[0] | vector[1] | vector[2];
173         Ready <= 1;
174                           ready <= '1;
175         if(!stop)
176                             if(vector[fm] == 1 << fm
177                             && fm + 1 == FAULTS)
178                                     begin
179                                             nextState <= IDLE;
180                                             Ready <= 0;
181                                             ready <= '0;
182                                     end
183           else if(vector[fm] == 1 << fm)
184             begin
```

```
185               nextState = D;
186               ready <= '0;
187             end
188                           else begin
189                                           nextState = B;
190                                           Ready <= 0;
191                                           ready <= '0;
192                           end
193       end
194     else if(!Ready) // pass vel de melhorias
195       begin
196         out <= out | vector[k];
197         if(k > 0) k--;
198         else if(k == 0) begin
199                                   Ready <= 1;
200                                   ready <= '1;
201                                   if(!stop)
202                                     if(vector[fm] == 1 << fm
203                                     && fm + 1 == FAULTS)
204                                             begin
205                                                     nextState <= IDLE;
206                                                     Ready <= 0;
207                                                     ready <= '0;
208                                             end
209                                     else if((vector[fm] == 1 << fm)
210                                     && Ready)
211                                             begin
212                                                     nextState <= D;
213                                                     currentState <= D;
214                                                     fm <= fm + 1;
215                                                     ready <= '0;
216                                             end
217                                     else begin
218                                        nextState = B;
219                                             Ready <= 0;
220                                             ready <= '0;
221                                     end
222                           end
223       end
224     else if(vector[fm] == 1 << fm
```

```
225     && fm + 1 < FAULTS && Ready)
226       begin
227         if(!stop) begin
228           nextState <= D;
229           currentState <= D;
230           fm++;
231           k <= 0;
232                               Ready <= 0;
233                               ready <= '0;
234         end
235       end
236
237     else
238       if(!stop) nextState = B;
239         IDLE:
240                   ready <= '0;
241
242
243   endcase;
244
245 end
246
247 endmodule
```

6.1.3 Módulo gerador de entradas (inpt$_gen.sv$)

```
1  //----------------------------------------------------
2  // Design Name : inpt_gen
3  // File Name   : inpt_gen.sv
4  // Function    : Generate every inputs to the especified use
5  // Coder       : Allender Vilar de Alencar
6  // version     : 1.3
7  //----------------------------------------------------
8  module inpt_gen #(
9  parameter M = 5
10 )
11 (clk, reset, stop, inpt_i, validinp, validcont);
12 //inputs
13 input logic clk, reset, stop;
14 //outputs
```

```
15 output logic [M-1:0] inpt_i;
16 logic [M-1:0] inpt_o;
17 output logic validinp, validcont;
18
19
20 typedef enum logic [1:0]{IDLE, CMPT, STRT} State;
21
22 //FSM
23 State cState, nState;
24 //Next State logic
25 always_ff @(posedge clk)
26         if(reset)
27                 cState = STRT;
28         else
29                 cState = nState;
30
31 always_ff @(posedge clk)
32         if(reset)
33                 inpt_i = 0;
34         else
35            inpt_i = inpt_o;
36 //States for the start, idle and calculate
37 always_comb
38         case(cState)
39                 IDLE: begin
40                         if(!stop) begin
41                                 validinp    = 0;
42                                 validcont = 0;
43                                 inpt_o    = inpt_i + 1;
44                                 nState    = CMPT;
45                         end
46                         else begin
47                                 validinp    = 0;
48                                 validcont = 0;
49                                 inpt_o    = inpt_i;
50                                 nState    = IDLE;
51                         end
52                 end
53                 CMPT: begin
54                         validcont    = 1;
```

```
55                          nState        = IDLE;
56                          inpt_o        = inpt_i;
57                          if(inpt_o == '1) begin
58                                  validinp    = 1;
59                          end
60                          else
61                                  validinp    = 0;
62                  end
63                  STRT: begin
64                          validinp    = 0;
65                          validcont = 0;
66                          inpt_o        = inpt_i;
67                          if(!stop)
68                                  nState        = CMPT;
69                          else
70                                  nState        = STRT;
71                  end
72          endcase
73
74  endmodule
```

6.1.4 Módulo controlador (cntrl.sv)

```
1   //-------------------------------------------------------
2   // Design Name : cntrl
3   // File Name   : cntrl.sv
4   // Function    : control the input and fault generator.
5   // Coder       : Allender Vilar de Alencar
6   // version     : 1.2
7   //-------------------------------------------------------
8   module cntrl (clk, reset, ready, validinp, flt, inp);
9
10  typedef enum logic [1:0] {INPT, FALT, WAIT, STOP} State;
11
12  input logic ready, validinp, clk, reset;
13  output logic flt, inp;
14
15  State cState, nState;
16
17          //fsm
```

```
18          always_ff @(posedge clk)
19                  if(reset) cState <= STOP;
20                  else      cState <= nState;
21          //States that control the input generator and fault generator
22          always_comb
23          case (cState)
24                  INPT:
25                  begin
26                          inp <= 0;
27                          flt <= 1;
28                          if(validinp) nState <= FALT;
29                          else       nState <= INPT;
30                  end
31                  FALT:
32                  begin
33                     if(ready) begin
34                             inp <= 1;
35                             flt <= 0;
36                                  nState <= FALT;
37                          end
38                          else begin
39                             inp <= 1;
40                             flt <= 1;
41                             nState <= WAIT;
42                     end
43                  end
44                  STOP:
45                  begin
46                          inp <= 1;
47                          flt <= 1;
48                          if(ready)  nState <= WAIT;
49                          else       nState <= STOP;
50                  end
51                  WAIT: begin
52                     inp <= 1;
53                          flt <= 1;
54                          if(ready)  nState <= INPT;
55                          else       nState <= WAIT;
56             end
57          endcase
```

```
58
59 endmodule
```

6.1.5 Módulo Comparador e contador (compt_rel.sv)

```
1  //-----------------------------------------------------
2  // Design Name : compt_rel
3  // File Name   : compt_rel.sv
4  // Function    : Module that counts for compute reliability
5  // Coder       : Allender Vilar de Alencar
6  // version     : 1.2
7  //-----------------------------------------------------
8  module compt_rel #(
9  parameter M = 5,
10 parameter N = 2,
11 parameter FAULTS = 6
12 )
13 (clk, reset, outp_i, outpf_i, validflt, fm, count_i, validcont);
14 //inputs
15 input logic clk, reset, validflt, validcont;
16 input logic [N-1:0] outp_i ;
17 input logic [N-1:0] outpf_i ;
18 input logic [$clog2(FAULTS)-1:0] fm;
19
20 //outputs
21
22 //In this case 20 = 6!/(3!3!)
23 output logic [M + $clog2(20) - 1:0] count_i [FAULTS];
24 logic [M + $clog2(20) - 1:0] count_o [FAULTS];//counter 20 is the maximum
25 //of ports and simultaneous faults Cw,(w/2) or (w-1)/2 or (w+1)/2, if use
26 //fault, calculate the maximum combination of used simultaneous
27
28 //Count ff
29 always_ff @(posedge clk) begin
30         if(reset) begin
31                 for(int i = 0; i < FAULTS ; i++)
32                         count_i [i] <= '0;
33         end
34         else begin
35                 count_i[fm] <= count_o[fm];
```

```
36          end
37
38  end
39  //Count increment logic
40  always_comb
41          if(validcont & (outp_i == outpf_i))
42                  count_o[fm] = count_i[fm] + 1;
43          else
44                  count_o[fm] = count_i[fm];
45
46
47  endmodule
```

6.1.6 Instanciação UART no módulo base

```
1   logic [1:0] z;
2   logic clkok;
3   logic [$clog2((M + $clog2(COMB))/8)-1:0] countrest;
4   logic alt;
5
6   always_ff @(posedge CLOCK_50)
7   begin
8           if(~SW[2])
9           begin
10                  transmit      <= '0;
11                  data_to_send  <= '0;
12                  countr        <= count[z];
13                  countrest     <= '0;
14                  alt           <= '0;
15          end
16          else
17          begin
18                  if(!is_transmitting & !alt)
19                  begin
20                          if(countrest == 0)
21                                  countr        <= count[z];
22                     else
23                                  countr       <= countr >> 8;
24                          if(SW[9])
25                                  data_to_send <= 8'h66;
```

```
                    else
                                data_to_send <= countr [7:0];
                        transmit      <= 1;
                        countrest++;
                        alt <= 1;
                end
                else begin
                        transmit      <= 0;
                   alt <= 0;
            end
        end
end


logic [7:0] data_to_send;
logic transmit;
logic is_transmitting;

uart UART(
    .clk(CLOCK_50), // The master clock for this module
    .rst(~KEY[3]), // Synchronous reset.
    .rx(GPIO_0[1]), // Incoming serial line
    .tx(GPIO_0[2]), // Outgoing serial line
    .transmit(transmit), // Signal to transmit
    .tx_byte(data_to_send), // Byte to transmit
    .is_transmitting(is_transmitting) // Low when transmit line is idle.
);
//Switchs to choose the vector
always_comb
        z = {SW[1],SW[0]};
```

6.1.7 Módulo UART (uart.v)

```
//`timescale 1ns / 1ps
// Documented Verilog UART
// Copyright (C) 2010 Timothy Goddard (tim@goddard.net.nz)
// Distributed under the MIT licence.
//
// Permission is hereby granted, free of charge, to any person obtaining a
// of this software and associated documentation files (the "Software"), t
```

```
8  // in the Software without restriction, including without limitation the rights
9  // to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
10 // copies of the Software, and to permit persons to whom the Software is
11 // furnished to do so, subject to the following conditions:
12 //
13 // The above copyright notice and this permission notice shall be included in
14 // all copies or substantial portions of the Software.
15 //
16 // THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
17 // IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
18 // FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
19 // AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
20 // LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FRON
21 // OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
22 // THE SOFTWARE.
23 //
24 module uart(
25     input clk, // The master clock for this module
26     input rst, // Synchronous reset.
27     input rx, // Incoming serial line
28     output tx, // Outgoing serial line
29     input transmit, // Signal to transmit
30     input [7:0] tx_byte, // Byte to transmit
31     output received, // Indicated that a byte has been received.
32     output [7:0] rx_byte, // Byte received
33     output is_receiving, // Low when receive line is idle.
34     output is_transmitting, // Low when transmit line is idle.
35     output recv_error // Indicates error in receiving packet.
36     );
37
38 parameter CLOCK_DIVIDE = 217; // clock rate (50Mhz) / (baud rate (57600) * 4)
39
40 // States for the receiving state machine.
41 // These are just constants, not parameters to override.
42 parameter RX_IDLE = 0;
43 parameter RX_CHECK_START = 1;
44 parameter RX_READ_BITS = 2;
45 parameter RX_CHECK_STOP = 3;
46 parameter RX_DELAY_RESTART = 4;
47 parameter RX_ERROR = 5;
```

```
48 parameter RX_RECEIVED = 6;
49
50 // States for the transmitting state machine.
51 // Constants - do not override.
52 parameter TX_IDLE = 0;
53 parameter TX_SENDING = 1;
54 parameter TX_DELAY_RESTART = 2;
55
56 reg [10:0] rx_clk_divider = CLOCK_DIVIDE;
57 reg [10:0] tx_clk_divider = CLOCK_DIVIDE;
58
59 reg [2:0] recv_state = RX_IDLE;
60 reg [5:0] rx_countdown;
61 reg [3:0] rx_bits_remaining;
62 reg [7:0] rx_data;
63
64 reg tx_out = 1'b1;
65 reg [1:0] tx_state = TX_IDLE;
66 reg [5:0] tx_countdown;
67 reg [3:0] tx_bits_remaining;
68 reg [7:0] tx_data;
69
70 assign received = recv_state == RX_RECEIVED;
71 assign recv_error = recv_state == RX_ERROR;
72 assign is_receiving = recv_state != RX_IDLE;
73 assign rx_byte = rx_data;
74
75 assign tx = tx_out;
76 assign is_transmitting = tx_state != TX_IDLE;
77
78 always @(posedge clk) begin
79         if (rst) begin
80                 recv_state = RX_IDLE;
81                 tx_state = TX_IDLE;
82         end
83
84         // The clk_divider counter counts down from
85         // the CLOCK_DIVIDE constant. Whenever it
86         // reaches 0, 1/16 of the bit period has elapsed.
87    // Countdown timers for the receiving and transmitting
```

```
88          // state machines are decremented.
89          rx_clk_divider = rx_clk_divider - 1;
90          if (!rx_clk_divider) begin
91                  rx_clk_divider = CLOCK_DIVIDE;
92                  rx_countdown = rx_countdown - 1;
93          end
94          tx_clk_divider = tx_clk_divider - 1;
95          if (!tx_clk_divider) begin
96                  tx_clk_divider = CLOCK_DIVIDE;
97                  tx_countdown = tx_countdown - 1;
98          end
99
100         // Receive state machine
101         case (recv_state)
102                 RX_IDLE: begin
103                         // A low pulse on the receive line indicates the
104                         // start of data.
105                         if (!rx) begin
106                                 // Wait half the period - should resume in the
107                                 // middle of this first pulse.
108                                 rx_clk_divider = CLOCK_DIVIDE;
109                                 rx_countdown = 2;
110                                 recv_state = RX_CHECK_START;
111                         end
112                 end
113                 RX_CHECK_START: begin
114                         if (!rx_countdown) begin
115                                 // Check the pulse is still there
116                                 if (!rx) begin
117                                         // Pulse still there - good
118                                         // Wait the bit period to resume half-
119                                         // through the first bit.
120                                         rx_countdown = 4;
121                                         rx_bits_remaining = 8;
122                                         recv_state = RX_READ_BITS;
123                                 end else begin
124                                         // Pulse lasted less than half the per
125                                         // not a valid transmission.
126                                         recv_state = RX_ERROR;
127                                 end
```

```
128                         end
129                 end
130                 RX_READ_BITS: begin
131                         if (!rx_countdown) begin
132                                 // Should be half-way through a bit pulse
133                                 // Read this bit in, wait for the next if
134                                 // have more to get.
135                                 rx_data = {rx, rx_data[7:1]};
136                                 rx_countdown = 4;
137                                 rx_bits_remaining = rx_bits_remaining - 1;
138                                 recv_state = rx_bits_remaining ? RX_READ_B
    RX_CHECK_STOP;
139                         end
140                 end
141                 RX_CHECK_STOP: begin
142                         if (!rx_countdown) begin
143                                 // Should resume half-way through the stop
144                                 // This should be high - if not, reject th
145                                 // transmission and signal an error.
146                                 recv_state = rx ? RX_RECEIVED : RX_ERROR;
147                         end
148                 end
149                 RX_DELAY_RESTART: begin
150                         // Waits a set number of cycles before accepting
151                         // another transmission.
152                         recv_state = rx_countdown ? RX_DELAY_RESTART :
    RX_IDLE;
153                 end
154                 RX_ERROR: begin
155                         // There was an error receiving.
156                         // Raises the recv_error flag for one clock
157                         // cycle while in this state and then waits
158                         // 2 bit periods before accepting another
159                         // transmission.
160                         rx_countdown = 8;
161                         recv_state = RX_DELAY_RESTART;
162                 end
163                 RX_RECEIVED: begin
164                         // Successfully received a byte.
165                         // Raises the received flag for one clock
```

```
166                     // cycle while in this state.
167                     recv_state = RX_IDLE;
168             end
169     endcase
170
171     // Transmit state machine
172     case (tx_state)
173             TX_IDLE: begin
174                     if (transmit) begin
175                             // If the transmit flag is raised in the idle
176                             // state, start transmitting the current conte
177                             // of the tx_byte input.
178                             tx_data = tx_byte;
179                             // Send the initial, low pulse of 1 bit period
180                             // to signal the start, followed by the data
181                             tx_clk_divider = CLOCK_DIVIDE;
182                             tx_countdown = 4;
183                             tx_out = 0;
184                             tx_bits_remaining = 8;
185                             tx_state = TX_SENDING;
186                     end
187             end
188             TX_SENDING: begin
189                     if (!tx_countdown) begin
190                             if (tx_bits_remaining) begin
191                                     tx_bits_remaining = tx_bits_remaining
192                                     tx_out = tx_data[0];
193                                     tx_data = {1'b0, tx_data[7:1]};
194                                     tx_countdown = 4;
195                                     tx_state = TX_SENDING;
196                             end else begin
197                                     // Set delay to send out 2 stop bits.
198                                     tx_out = 1;
199                                     tx_countdown = 8;
200                                     tx_state = TX_DELAY_RESTART;
201                             end
202                     end
203             end
204             TX_DELAY_RESTART: begin
205                     // Wait until tx_countdown reaches the end before
```

```
206                             // we send another transmission. This covers the
207                             // "stop bit" delay.
208                             tx_state = tx_countdown ? TX_DELAY_RESTART :
    TX_IDLE;
209                     end
210             endcase
211 end
212
213 endmodule
```

6.2 *Scripts Matlab*

6.2.1 Gerador de gráfico de confiabilidade

```
%{
Script that generate the graph of the reliability
in function of reliability 'q' of ports.
version: 1.1
Coder: Allender V.
%}
x = linspace(0.5,1,100);
w = vpa(6); %Number of ports in netlist
y = 0;
count = [32 34 113 152 126 46 9];
for k = 0:6 %from 0 to max simultaneous faults
    y = y + count(k+1).*x.^(w-k).*(1-x).^k;
end
y = y./32;
plot(x,y);
title('Reliability')
xlabel('q')
ylabel('Reliability')
grid;
```

6.2.2 Gerador de gráfico de confiabilidade truncado

```
%{
Script that generate the graph of the reliability
with maximum number of faults limited
```

```
in function of reliability 'q' of ports.
version: 1.2
Coder: Allender V.
%}
x = linspace(0.5,1,100);
w = vpa(100); %Number of ports in netlist
y = 0;
for k = 0:40 %from 0 to max simultaneous faults
    y = y + nchoosek(w,k).*x.^(w-k).*(1-x).^k;
end
plot(x,y);
title('Reliability')
xlabel('q')
ylabel('Reliability')
grid;
```

6.2.3 Gerador de comparação entre confiabilidade completa e truncada

```
%{
Script that generate the graph for comparison
of the reliability and truncated reliability
in function of reliability 'q' of ports.
version: 1.3
Coder: Allender V.
%}
x = linspace(0.5,1,100);
w = vpa(5); %Number of ports in netlist
count = [64 75 147 154 192 23];
y1 = 0;
for k = 0:5 %from 0 to max simultaneous faults(PUT COMPLETE)
    y1 = y1 + count(k+1).*x.^(w-k).*(1-x).^k;
end
y1 = y1./64;
plot(x,y1);
title('Reliability')
xlabel('q')
ylabel('Reliability')
hold on
y2 = 0;
```

```
for k = 0:3 %from 0 to max simultaneous faults(TRUNCATED)
    y2 = y2 + count(k+1).*x.^(w-k).*(1-x).^k;
end
y2 = y2./64;
plot(x,y2);

hold off
legend('complete','truncated')
grid;
```

6.3 *Scripts Python*

6.3.1 Insert the injector in the netlist

```
#Coder - Allender Vilar
#version - 1.2
#Insert in fisrt file 'filename' the injectors
#and save in the second 'filename'
ports = list()
ports_quantity = 0
# file to open
filename = 'adder.v' #ideal netlist file
with open(filename) as fin:
    for line in fin:
        ports.append(line)

# couting ports
for port in ports:
    m = port.split()
    if m:#ports supported if need more, just add
        if(m[0] == 'nand' or m[0] == 'not' or m[0] == 'nor'
           or m[0] == 'xor' or m[0] == 'and' or m[0] == 'or'):
            ports_quantity = ports_quantity + 1
print(ports_quantity)

#file that will be generated
filename = 'adderinj.v' #name of the netlist
y = 0
count = 0
aux = 0
```

```
with open(filename, 'w') as fout:
    for port in ports:
        m = port.split()
        if m:
            if(count == 0):
                for i in port:
                    if(i == ')'):
                        count = 1
                        aux = aux + 1
                    porti = port.replace(");", "")
                    porti = porti.replace("c17", "c17inj")#change the name of module
                fout.write(porti)
                if (count == 1):
                    for i in range(0, ports_quantity):
                        fout.write(",E" + str(i))
                    fout.write(");\n")

                    fout.write("input E0")
                    for i in range(1, ports_quantity):
                        fout.write(",E" + str(i))
                    fout.write(";\n")

                    fout.write("wire I0" )
                    for i in range(1, ports_quantity):
                        fout.write(",I" + str(i))
                    fout.write(";\n")

            else: #ports supported if need more, just add
                if(m[0] == 'nand' or m[0] == 'not' or m[0] == 'nor' or
                   m[0] == 'xor' or m[0] == 'and' or m[0] == 'or'):
                    print(m[2][1:4])
                    porti = port.replace(m[2], "(I"+ str(y) +",")
                    fout.write(porti)
                    fout.write('xor xor_' + str(y) + " " + m[2] +  " I" + str(y) +
                               ', E' + str(y) + ");" + "\n")
                    y = y + 1
                else:
                    fout.write(port)
```

6.3.2 Gerador de instanciação de entradas e saídas desempacotadas - modulo ideal

```
#Coder - Allender Vilar
#version - 1.0
#Generate intantiation inputs, outputs
M = 5
N = 2
filename = 'dutidel.txt'

with open(filename, 'w') as dut:
    dut.write("(")
    for i in range(0, M):
        dut.write("inpt[" + str(i) + "], ")
    dut.write("\n")

    for i in range(0, N-1):
        dut.write("out_ideal[" + str(i) + "], ")
    dut.write("out_ideal[" + str(N-1) + "]")
    dut.write("\n")
    dut.write(");")
```

6.3.3 Gerador de instanciação de entradas e saídas desempacotadas - módulo sabotado

```
#Coder - Allender Vilar
#version - 1.1
#Generate intantiation inputs, outputs and injections
M = 5
N = 2
FAULTS = 6
filename = 'dutinj.txt'

with open(filename, 'w') as dut:
    dut.write("(")
    for i in range(0, M):
```

```
    dut.write("inpt[" + str(i) + "], ")
dut.write("\n")

for i in range(0, N):
    dut.write("out_injet[" + str(i) + "], ")
dut.write("\n")

for i in range(0, FAULTS-1):
    dut.write("inj[" + str(i) + "], ")
dut.write("inj[" + str(FAULTS-1) + "]")
dut.write("\n")
dut.write(");")
```