



Universidade Federal do Ceará
Centro de Tecnologia
Departamento de Engenharia de Teleinformática
Curso de Engenharia de Computação

PEDRO LUCAS FALCÃO LIMA

Projeto de um IP Soft Core para Detecção de
Ataques DDoS

Fortaleza, Ceará
2017

PEDRO LUCAS FALCÃO LIMA

**Projeto de um IP Soft Core para Detecção de
Ataques DDoS**

Monografia apresentada ao Curso de Engenharia de Computação da Universidade Federal do Ceará, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Computação.

Orientador: Prof. Msc. Ricardo Jardel Nunes da Silveira

**Fortaleza, Ceará
2017**

PEDRO LUCAS FALCÃO LIMA

**Projeto de um IP Soft Core para Detecção de
Ataques DDoS**

Monografia apresentada ao Curso de Engenharia de Computação da Universidade Federal do Ceará, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Computação.

Aprovada em: ____/____/____

BANCA EXAMINADORA

Prof. Msc. Ricardo Jardel Nunes da Silveira
(Orientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. Jarbas Aryel da Silveira
Universidade Federal do Ceará (UFC)

Prof. Dr. Otávio Alcântara de Lima Júnior
Instituto Federal do Ceará - IFCE

Dedico este trabalho à minha família e namorada, pessoas que
fizeram de tudo para que eu chegasse onde cheguei.

Agradecimentos

Agradeço primeiramente a Deus, que iluminou meu caminho durante essa jornada, me dando saúde e força para superar as dificuldades.

À minha namorada, Luéline Elias, pelo amor, paciência, dedicação e companheirismo em todos os momentos.

À minha família, por sua capacidade de acreditar e investir em mim. Mãe, sua dedicação foi o que deu, em alguns momentos, a esperança para seguir.

Ao meu orientador, Prof. Ricardo Jardel Nunes da Silveira, pelo acompanhamento e estreitamento da relação professor-aluno e exemplo de profissional bem como pelo apoio, incentivo, sugestões e comentários durante a supervisão dos meus estudos.

Ao meu coorientador, Prof. Marcelo Araújo Lima, pelo apoio, incentivo, sugestões e tempo dedicado para me ajudar durante meus estudos.

Aos meus amigos da Universidade Federal do Ceará, 8086FC e 8086Team pela amizade e pelos momentos de descontração e estudo.

E a todos que direta ou indiretamente fizeram parte da minha formação, o meu muito obrigado.

*"A persistência é o caminho do êxito."
(Charles Chaplin)*

Resumo

Sistemas de detecção de intrusão são cada vez mais necessários para garantir a segurança de serviços na internet. Uma vez que as ameaças de rede vem desenvolvendo-se cada vez mais. Ataques do tipo DDoS são muito comuns atualmente, uma vez que já existem recursos suficientes para realizar esse tipo de ataque em tempo real. Apesar de existirem soluções em softwares para detectar ataques DDoS, muitas vezes são ineficientes. Nesse trabalho foi construído um módulo em Hardware que realiza a detecção de ataque DDoS em tempo real, com um tempo de menos de 1μ segundos e precisão significativa em relação a soluções de softwares.

Abstract

Key-words: DDoS attack detection. Network Security. Real-time. Framework.

Lista de ilustrações

Figura 2.1 – Representação de aritmética de ponto fixo	21
Figura 2.2 – Comparativo de potência e performance entre as famílias da série 7 da Xilinx	23
Figura 3.1 – Operações que o Módulo em <i>hardware</i> efetua	25
Figura 3.2 – Módulo Nahid	25
Figura 3.3 – Design do Datapath desenvolvido por (HOQUE; KASHYAP; BHAT- TACHARYYA, 2017)	26
Figura 3.4 – Componente extend	27
Figura 3.5 – Componente Reduce	27
Figura 3.6 – Tipos do componente Mux no módulo	28
Figura 3.7 – Componente Mul	29
Figura 3.8 – Componente Adder	29
Figura 3.9 – Componente Divider	30
Figura 3.10–Componente Sqrt	31
Figura 3.11–Componente Register	32
Figura 3.12–Esquemático do controller,ciclos de entrada e saídas para as entradas do datapath num dado ciclo de <i>clock</i>	33
Figura 3.13–Ciclos do Módulo Nahid 3	33
Figura 4.1 – Simulação 1 da tabela 3	35

Lista de tabelas

Tabela 1	– Relatório de Utilização do artigo de comparação	34
Tabela 2	– Relatório de Utilização do trabalho proposto	34
Tabela 3	– Comparativo de resultados do Nahid implementado em software e hardware	35
Tabela 4	– Tempo de detecção	35

Lista de abreviaturas e siglas

DoS	Denial of Service
DDoS	Distributed Denial of Service
FPGA	Field Programmable Gate Array
IDS	Intrusion Detection System
CPU	Central Processing Unit
ASIC	Application Specific Integrated Circuits
IP	Intellectual property

Lista de símbolos

X Vetor de entrada para correlação NaHiD

Sumário

1	INTRODUÇÃO	14
1.1	Objetivos	15
1.1.1	<i>Objetivos Gerais</i>	15
1.2	Organização da monografia	15
2	REFERENCIAL TEÓRICO	17
2.1	Ataques	17
2.2	DoS	17
2.3	DDoS	17
2.4	Deteccção	18
2.5	Correlação	18
2.6	Soluções em <i>Hardware</i>	19
2.7	Técnicas Utilizadas	19
2.7.1	<i>Soma:</i>	19
2.7.2	<i>Módulo:</i>	20
2.7.3	<i>Divisão:</i>	20
2.7.3.1	<i>Aritmética de Ponto Fixo</i>	20
2.7.4	<i>Média aritmética:</i>	21
2.7.5	<i>Desvio padrão:</i>	21
2.8	FPGA	21
2.9	FPGA'S Xinlix de série 7	22
3	MÓDULO NAHID	24
3.1	Módulo Nahid	24
3.1.1	<i>Datapath</i>	25
3.1.1.1	<i>Extend</i>	27
3.1.1.2	<i>Reduce</i>	27
3.1.1.3	<i>Mux</i>	27
3.1.1.4	<i>Mul</i>	28
3.1.1.5	<i>Adder</i>	29
3.1.1.6	<i>Divider</i>	30
3.1.1.7	<i>Sqrt</i>	30
3.1.1.8	<i>Register</i>	31
3.1.2	<i>Controller</i>	32
4	RESULTADOS EXPERIMENTAIS	34

5	CONCLUSÕES E TRABALHOS FUTUROS	36
	REFERÊNCIAS	37
6	CÓDIGOS DO MÓDULO	39
6.1	Código Nahid	39
6.2	Código Datapath	41
6.3	Código Extend	46
6.4	Código Reduce	47
6.5	Código Mux	47
6.6	Código Mul	47
6.7	Código Adder	47
6.8	Código Register	48
6.9	Código Controller	48

1 Introdução

A crescente difusão da *internet* e sistemas *web* na atualidade, cada vez mais serviços são disponibilizados por meio da rede mundial de computadores. Serviços como: armazenamento, transações financeiras e plataformas de dados cadastrais são cada vez mais comuns. Por isso, é necessário que tais serviços cumpram os requisitos de disponibilidade e segurança. Assim, sistemas de detecção de intrusão (IDS), são comumente usados para garantir a segurança por meio de análise e detecção de tráfegos maliciosos, bem como tomar medidas corretivas em caso de tráfegos maliciosos.

Mediante a esse crescimento de usuários e serviços na *internet*, ameaças na rede vem desenvolvendo-se cada vez mais. Por isso, nota-se uma maior complexidade nessas ameaças. De acordo com Mandia e PROSISE (2001) são considerados ataques a segurança qualquer vento que interrompam os procedimentos normais causando algum nível de crise, tais como invasões de computador, ataques de negação de serviço, furto de informações por pessoal interno. Ataques podem ser do tipo de negação de serviços.

Os ataques *DoS* (sigla para Denial of Service), que podem ser interpretados como "Ataques de Negação de Serviços", consistem em tentativas de fazer com que computadores - servidores *Web*, por exemplo tenham dificuldade ou mesmo sejam impedidos de executar suas tarefas. Para isso, em vez de "invadir" o computador ou mesmo infectá-lo com *malwares*, o autor do ataque faz com que a máquina receba tantas requisições que esta chega ao ponto de não conseguir dar conta delas. Em outras palavras, o computador fica tão sobrecarregado que nega serviço (HOQUE; KASHYAP; BHATTACHARYYA, 2017). Ataques do tipo *DoS* distribuídos são chamados de ataques *DDoS*.

DDoS, sigla para *Distributed Denial of Service*, é um tipo de ataque *DoS* de grandes dimensões, ou seja, que utiliza até milhares de computadores para atacar uma determinada máquina, distribuindo a ação entre elas. Trata-se de uma forma que aparece constantemente no noticiário, já que é o tipo de ataque mais comum na *internet* (ALECRIM, 2008).

Para detectar ataques *DDoS* em tempo real, o mecanismo de detecção deve ser capaz de detectar ataques de forma eficiente de um pequeno conjunto de características relevantes. Portanto, é necessária uma medida efetiva para classificar um tráfego em tempo real.

Essa detecção passa, por uma série de análises de dados, por isso é necessário a utilização de medidas estatísticas, consequentemente cálculos computacionalmente complexos. O alto rendimento é essencial para a escalabilidade da detecção, o que é necessário no caso de ataques *DDoS*.

Diante disso, soluções baseadas em software são ineficientes para aplicações de tempo real, uma vez que eles exigem grande quantidade de ciclos de *CPU* de propósitos gerais. Logo, é necessário que soluções em *hardware* estejam presentes nas detecções de ataques *DDoS*. Podendo assim, ser gerados sistemas híbridos (*hardware* e *software*) que possuem alto desempenho e precisão.

Os tipos de Hardware que possuem características para acomodar grandes lógicas e possuem alto desempenho são as *FPGAs* e *ASIC*, porém, as *FPGAs* oferecem adaptabilidade dinâmica, que é importante para aplicações que requerem mudanças frequentes em suas configurações, como a detecção de ataques *DDoS* que evoluem com frequência.

Por isso foi proposto um módulo de detecção, afim de garantir desempenho e precisão de ataques, utilizando *FPGA*, que junto a sistemas de *softwares* consigam detectar ataques *DDoS*.

1.1 Objetivos

1.1.1 Objetivos Gerais

- A implementação de um módulo em hardware capaz de detectar de ataques em tempo real.
- A implementação de um módulo em hardware que possua um ganho de desempenho e possua precisão satisfatória em relação a módulos em softwares.
- Utilização das *FPGAs* mais recentes
- Ganhos de desempenho em relação a trabalhos similares.
- Estudo de métodos que utilizam aproximação aos resultados de cálculos em softwares de maneira otimizada.
- A Utilização de IP cores no desenvolvimento, para uma maior agilidade e confiabilidade na construção do código.

1.2 Organização da monografia

Os estudos deste trabalho estão organizados da seguinte forma: No próximo capítulo será apresentado um estudo bibliográfico sobre ameaças de rede e ataques *DDoS*, detecção e solução em *hardware*. No Capítulo 3, a modelagem do módulo Nahid é descrita e no Capítulo 4, apresentamos o desempenho pelo módulo estudado por meio da taxa de correlação calculada na detecção e tempo de computação nessa detecção. Por fim, o último

capítulo deste trabalho apresenta as conclusões realizadas a partir dos resultados obtidos e algumas perspectivas para a continuação deste trabalho.

2 Referencial Teórico

2.1 Ataques

Ataques a uma rede de computadores são ações maliciosas em que *softwares* são utilizados para de alguma forma prejudicar, interromper uma ação ou invadir uma rede ou máquina, afim de se beneficiar com tal ação. Todavia um ataque, deve ser caracterizado a partir de que se comprova que ele não é apenas ameaça e sim uma atividade maliciosa. Por isso, vale ressaltar que o ataque é a ação propriamente dita, enquanto, uma ameaça a um sistema é algo que possa afetar ou atingir o seu funcionamento, operação, disponibilidade e integridade. Podemos dizer que um ataque ocorre quando uma ameaça intencional é realizada (PINHEIRO, 2017). Outra definição de ataque está relacionada com um tráfego não desejado, que é qualquer tipo de tráfego de rede não requisitado e/ou inesperado, cujo único propósito é consumir recursos computacionais da rede, desperdiçar tempo e dinheiro dos usuários e empresas e que pode gerar algum tipo de vantagem ou benefício (lucro) para seus criadores (FEITOSA; SOUTO; SADOK, 2008).

2.2 DoS

Um dos tipos mais comuns de ataque é o DoS (Denial Of Service), que é um tipo de ataque no qual se lança um grande número de requisições sobre uma vítima, de forma a sobrecarregar a vítima, evitando assim que a mesma faça algum tipo de trabalho "útil" (HANDLEY; RESCORLA, 2006). Uma característica importante desses tipos de ataques é que o objetivo principal desse tipo de ataque não é a invasão em busca de dados ou informações, mas a negação do serviço que a vítima está utilizando ou oferecendo. Uma das estratégias mais utilizadas para a realização desse tipo de ataque é o envio de múltiplas requisições à vítima em questão, de forma a gerar uma sobrecarga tal que ela não suporte tantas requisições e comece a negar serviços, o que fora da situação de ataque ela seria capaz de realizar normalmente (MANDIA; PROSISE, 2001).

2.3 DDoS

DDoS (Distributed Denial of Service) é um tipo específico de ataque DoS de grandes dimensões, ou seja, que utiliza até milhares de computadores para atacar uma determinada máquina, distribuindo a ação entre elas (ALECRIM, 2008). Trata-se de uma forma muito utilizada, já que é o tipo de ataque mais comum na internet. Esta é uma outra definição "Um ataque DDoS usa muitos computadores para lançar um ataque DoS coordenado contra

um ou mais alvos. Usando a tecnologia cliente / servidor, o atacante é capaz de multiplicar a eficácia do DoS significativamente, aproveitando os recursos de vários computadores cúmplices involuntários, que servem como plataformas de ataque "(STEIN, 2002).

2.4 Detecção

A detecção de um ataque DDoS é um trabalho relativamente complexo, uma vez que esse tipo de ataque ocorre em tempo real, e muitas vezes é difícil de ser identificado a máquina que é o atacante principal, por isso utilizamos o conceito de "tráfego não desejado". Como visto anteriormente, antes da ação do ataque acontecer, existem as ameaças aquela rede ou máquina, por isso tráfegos que não são desejados (que podem ser vistos como ameaça anteriormente a ação do ataque propriamente dito) devem ser identificados e o sistema de segurança podem tomar as devidas medidas. Existem muitas técnicas de detecção de ataques DDoS que fazem uso de cálculos estatísticos (AHMED *et al.*, 2014), porém essas soluções não proporcionam alta precisão de detecção DDoS, pois fazem uso de apenas um pequeno conjunto de recursos de tráfego, ou seja, com poucos dados de tráfego a detecção se torna inviável. A Correlação é uma medida estatística que é muito utilizada nas detecções de ataques (YU *et al.*, 2012).

2.5 Correlação

Correlação é uma medida que mede o relacionamento entre duas variáveis. Uma das maiores vantagens da correlação é que ela não necessita de uma grande quantidade de variáveis, para chegar a uma conclusão de relacionamento entre dois conjuntos. Portanto para uma detecção de ataques consistentes é necessária uma medida de correlação efetiva para classificar ataques DDoS em tempo real, mesmo quando usa um pequeno número de recursos de tráfego. Uma correlação de forma resumida, é um conjunto de cálculos que, a partir das variáveis de entrada, retorna o relacionamento (similaridade, linearidade e direção) entre as variáveis, em um valor entre -1 e 1. Para calcular a correlação, é necessário se ter os valores de entrada, e um módulo (de *software* ou *hardware*) que realize os cálculos abaixo descritos, retornando o resultado da correlação. Segue abaixo a fórmula da correlação, conhecida como NaHiD, implementada em hardware neste trabalho para detecção de ataques DDoS.

$$NaHiD(X,Y) = 1 - \frac{1}{n} \sum_{i=1}^n \frac{(|X(i) - Y(i)|)}{||\mu X - sX| - X(i)| + ||\mu Y - sY| - Y(i)|} \quad (2.1)$$

onde

- μX : Média aritmética do objeto de tráfego X.

- μX : Média aritmética do objeto de tráfego Y.
- sX : Desvio padrão do objeto de tráfego X.
- sY : Desvio Padrão do objeto de tráfego Y.

2.6 Soluções em *Hardware*

Quando é necessário realizar um considerável número de cálculos em um dado sistema computacional, pode-se utilizar utilizar uma abordagem baseada em *hardware*, em *software* ou ainda uma abordagem mista. As utilizações de soluções baseadas em *software* possuem duas grandes vantagens: Podem ser reutilizadas para soluções de propósitos gerais, além serem de fácil implementação. Já as soluções baseadas em *hardware*, tendem a oferecer um alto desempenho e baixo consumo de energia (HOQUE; KASHYAP; BHATTACHARYYA, 2017). Na abordagem de Ataques DDoS, utilizamos um sistema misto, baseado em *hardware* e *software*, para detectar, em tempo de acontecimento, os ataques. Precisa-se portanto de velocidade e precisão numérica suficiente para diferenciar ataques de situações normais. *Softwares* contendo cálculos matemáticos complexos, em geral exigem grande quantidade de ciclos de CPU, o que pode implicar em uma detecção atrasada. Entretanto, uma arquiteturas baseada em solução mista, de hardware e software, pode oferecer o desempenho e precisão necessários, uma vez que tenhamos a flexibilidade de portabilidade do *software*, e pelo lado do *hardware*, ciclos de execução implementados de acordo com o problema específico (MIRANDA *et al.*, 2002) .

2.7 Técnicas Utilizadas

Nota-se que a correlação proposta por (HOQUE; KASHYAP; BHATTACHARYYA, 2017), necessita de uma certa precisão, pois a mesma é uma medida de detecção. Para tal é necessário que os resultados de suas operações aritméticas tenham uma certa precisão. É necessário estudar cada componente dessa formulação, e realizar a implementação em *hardware* de forma que o sistema possa tirar conclusões assertivas, a partir de cálculos precisos. Serão descritas as operações e as formas em hardware de se realizar esses cálculos:

2.7.1 Soma:

A soma pode ser implementada, com componentes do tipo somador aritmético, já bem conhecidos pela comunidade. Vale ressaltar que a soma pode ser utilizada no módulo, média e desvio padrão.

2.7.2 Módulo:

O módulo pode ser facilmente implementado em *hardware* com componentes do tipo somador. Vale ressaltar que o módulo pode ser utilizado no desvio padrão.

2.7.3 Divisão:

A divisão de dois números quaisquer possui uma certa complexidade em *hardware*, por não utilizar em suas operações apenas o conjunto dos números naturais, tendo que utilizar os conjuntos dos números fracionários. Para isso, utilizou-se um IP da plataforma VIVADO da *Xilinx*, chamado *Divider Generator*, esse IP realiza possui uma implementação de divisão de dois números, e retorna um resultado em uma representação de números fracionários com uma precisão escolhida de forma personalizada *Divider Generator v5.1 LogiCORE IP Product Guide*. Um conjunto opções de configuração desse IP, implica em uma latência específica (quantos ciclos de clock, são necessários para um término de operação) da computação da de divisão. Uma dessas opções consiste justamente no uso de aritmética de ponto fixo, em particular na quantidade de bits utilizada para representar a parte inteira, assim como a parte fracionária.

2.7.3.1 Aritmética de Ponto Fixo

Para a representação de números fracionários, tem-se duas abordagens mais conhecidas: Ponto fixo e Ponto Flutuante. A representação em ponto flutuante trás consigo grande flexibilidade e precisão, porém a um custo muito elevado em termos de *hardware*, adotando uma representação baseada em mantissa e expoente, requer circuitos de hardware com complexidade que vão muito além dos somadores aritméticos comuns de números naturais, que são os mesmos utilizados para aritmética de ponto fixo. A representação em ponto fixo divide a palavra binária em duas partes com quantidades de bits pré-determinadas (parte que representa o número inteiro e parte que representa em decimal), deixando portanto a precisão de ambas as partes sempre constante. A vantagem da representação em ponto fixo é facilidade e custo da implementação em *hardware* frente a representação em ponto flutuante, gerando um excelente desempenho e uma certa precisão nos resultados (WOODS *et al.*, 2008), desde de que se faça uma boa escolha da quantidade de bits a atribuir para cada uma das duas partes, o que requer um conhecimento prévio da faixa numérica a ser representada em cada parte (inteira e fracionária).

Na figura 2.1 podemos ver um exemplo de um número fracionário representado em ponto 4.4, ou seja, 4 bits representando a parte inteira e 4 bits representando a parte decimal.

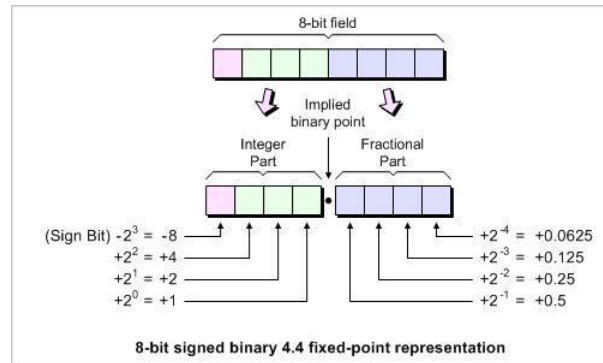


Figura 2.1 – Representação de aritmética de ponto fixo

2.7.4 Média aritmética:

A média aritmética é um tipo de soma de variáveis seguido por uma divisão, sendo esta última a parte mais complexa da média aritmética. Uma alternativa viável é utilizar divisão aritmética binária, de forma que, com pequenos ajustes em uma operação (ajustes que não influenciem no resultado dessa operação), seja gerado um resultado aproximado. No caso da média, seriam uma operação de soma de todos os elementos em questão, dividido pelo número de elementos, segundo 4 a divisão por números pares são aproximadas do resultado apenas pelo deslocamento de algum bit, desprezando a parte fracionária. Vale ressaltar que a Média Aritmética pode ser utilizada no desvio padrão.

2.7.5 Desvio padrão:

O Desvio padrão é a variável em questão subtraída da média, após isso elevada ao quadrado. Após essa série de cálculos, calcula-se a raiz quadrada, tendo então o desvio padrão. Então é necessário um multiplicador para realizar essa potenciação, bem como um módulo calculador de raiz quadrada. Multiplicadores são facilmente encontrados implementados em silício nas FPGAs. Já a raiz quadrada, por ser um cálculo mais específico, requer a utilização de um IP core que faça o cálculo. Para isso pode-se utilizar um IP da plataforma VIVADO da Xilinx, chamado CORDIC CORDIC v6.0, que realiza implementação de cálculo de raiz quadrada. Basicamente esse IP recebe uma variável como entrada, e em sua saída retorna a raiz quadrada aproximada do número de entrada. A configuração das opções desse IP implica em uma certa latência (quantos ciclos de clock, são necessários para um término de operação) da computação.

2.8 FPGA

Quanto mais se conhece a respeito do problema a ser resolvido, mais acertada será a escolha de que tipo de abordagem se deve utilizar, para acomodar a lógica do problema em questão. Como foi pontuado acima, o problema da detecção de ataques em tempo

real, requer duas características principais, desempenho e precisão. Para realizar essa detecção poderíamos utilizar dois tipos de unidade de processamento mais conhecidas ASICs(Circuitos integrados para uma aplicação específica) e as FPGAs(Arranjo de Portas Programáveis em Campo). Porém segundo Hoque, Kashyap e Bhattacharyya (2017), as FPGAs oferecem adaptabilidade dinâmica, que é importante para aplicações que requerem mudanças frequentes em suas configurações, como a detecção de ataques DDoS que evoluem com frequência. Além disso segundo Heath (2002) O tempo de projetos baseados em FPGA é normalmente conhecido com muita precisão a priori, o que trás um certo conforto para a equipe de projeto.

2.9 FPGA'S Xilinx de série 7

As FPGAs da série Xilinx 7 compreendem quatro famílias de FPGA que abordam uma gama completa de requisitos nos sistemas como: baixo custo, pequenos tamanhos, sensíveis ao custo, aplicações de alto volume de largura para altas conectividades, capacidade lógica e capacidade de processamento de sinal para aplicações de alto desempenho (PRZYBUS, 2010).

Artix (Baixo custo) , Kintex (Balanço de alta performance com baixo custo), Virtex (Sistemas de alta performance) e ZYNQ (Aplicações em sistemas embarcados em geral). A diferença entre essas famílias está basicamente em sua performance e baixo consumo , consequentemente para qual finalidade elas deverão ser usadas, vale ressaltar que custo e energia está diretamente ligado a baixo consumo de potência como podemos ver na Figura 2.2. Pra corroborar com isso nota-se que cada família tem variações em relação máxima capacidade dos recursos que ambas possuem, como Células Lógicas, Blocos de RAM , pinos E/S e etc . Cada bloco lógico configurável (CLB) é composto por dois *Slices* que podem ser do tipo M (SLICEM) , podem ser utilizados para memória e lógica, ou do tipo L (SLICEL), podem ser utilizados apenas para lógica e aritmética. Cada *Slice* pode ter como recursos gerenciáveis 4 LUTs(Look-up Tables) de 6 entradas , multiplexadores , *carry chains* e 4 flip-flops/latches. As interfaces E/S, em geral, trabalham pra proporcionar altas velocidades de resposta sem tentar perder integridade no sinal. Além de serem projetadas para diferentes padrões (Voltagens, larguras e protocolos). Nas famílias da 7 series elas podem ser caracterizados por dois tipos : High Range (HR), suportando padrões I/O de tensões de ate 3.3 V, e High Performance (HP), suportando somente padrões I/O de tensões de até 1.8V e projetado para alta performance, por isso dependente da família. Todos os membros das famílias de série 7 tem o mesmo bloco *RAM/FIFO(First in first out)* , operação totalmente síncrona, muitas opções de configurações (*true dual port, simple dual-port, single-port*) e etc (DEMETRIO; CONSTANTE; ARRIGONI,).

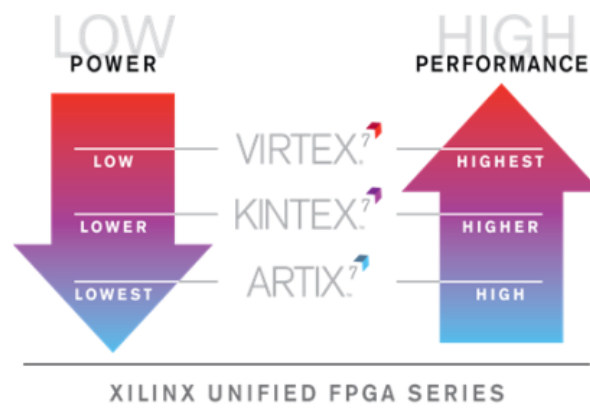


Figura 2.2 – Comparativo de potência e performance entre as famílias da série 7 da Xilinx

3 Módulo Nahid

Para o módulo de correlação em *hardware* receber as instâncias de tráfego no trabalho 6, foi implementado um módulo chamado de pré-processador e ao final da computação, o módulo de hardware envia para outro módulo que irá realizar algum tratamento no sistema, chamado de gerenciador de segurança. Nota-se que os módulos do pré-processador e do gerenciador de segurança são implementados separadamente usando *software*. As máquinas que implementam esses módulos e o *FPGA* podem comunicar-se usando as interfaces de E/S de alta velocidade suportadas pelos *FPGAs* modernos, como *PCI* e *Ethernet*. O módulo de detecção de ataque recebe a instância de tráfego do módulo pré-processador. Além disso, ele recebe o perfil normal e um valor limiar do banco de dados do perfil criado pelo gerenciador de segurança. Cada uma das instâncias de tráfego e o perfil normal são vetores que consistem em três recursos de tráfego. O módulo de detecção de ataque calcula primeiro o NaHiD VERC entre a instância de tráfego de entrada e o perfil normal. O valor de correlação calculado é comparado com o limite para classificar a ocorrência de tráfego recebido como ataque ou normal. O resultado da classificação é armazenado no banco de dados *log* para análise *off-line* pelo gerenciador de segurança. Além disso, um alarme é gerado no caso de a instância ser classificada como um ataque.

O módulo de detecção em hardware é basicamente a implementação da formulação dos passos da correlação proposta, utilizando uma *FPGA*. Para isso, é necessário adequar as operações para os componentes que são disponíveis e escolhidos para realizar a computação de uma dada operação. Além disso, para otimizar o tempo de resposta é necessário uma organização dessas operações em ciclos de *textitclock*, para ter uma referência de tempo para cada computação realizada e assim organizar a sequência das operações. Com isso, a arquitetura de detecção de ataques é proposta com o nome de Nahid. Essa arquitetura possui componentes de diversos níveis, nos quais estão dispostos dependendo da operação que esteja sendo feita no momento, vale ressaltar que essas operações são aritméticas, mudança no tamanho da palavra, registro e seleção. Segue abaixo as operações que o módulo em hardware realiza para efetuar a detecção de ataques.

3.1 Módulo Nahid

Nahid é o componente de mais alto nível, sendo ele quem recebe as entradas (perfil normal e instâncias de tráfego em análise) do módulo pré-processador e envia a saída (resultado da análise) para o gerenciador de segurança. O Nahid é composto basicamente por dois componentes internos, são esses: Datapath e Controller. Além dos vetores de entrada citados anteriormente, o componente Nahid recebe os sinais de controle, *clock*

$(X[3], Y[3]) = \text{Vectors to be measured}$ $TH = \text{Threshold}$ 1. $M_X = \frac{X_1 + X_2 + X_3}{3}, \quad M_Y = \frac{Y_1 + Y_2 + Y_3}{3}$ 2. $(M_X)^2 = M_X \times M_X, \quad (M_Y)^2 = M_Y \times M_Y$ 3. $M_{X^2} = \frac{X_1^2 + X_2^2 + X_3^2}{3}, \quad M_{Y^2} = \frac{Y_1^2 + Y_2^2 + Y_3^2}{3}$ 4. $SD_X = \sqrt{ M_{X^2} - (M_X)^2 }, \quad SD_Y = \sqrt{ M_{Y^2} - (M_Y)^2 }$ 5. $N_1 = X_1 - Y_1 , \quad N_2 = X_2 - Y_2 , \quad N_3 = X_3 - Y_3 $ 6. $D_1 = M_X - SD_X - X_1 + M_Y - SD_Y - Y_1 ,$ $D_2 = M_X - SD_X - X_2 + M_Y - SD_Y - Y_2 ,$ $D_3 = M_X - SD_X - X_3 + M_Y - SD_Y - Y_3 $ 7. $NaHiD_{VERC}(X, Y) = 1 - \frac{N_1}{D_1} + \frac{N_2}{D_2} + \frac{N_3}{D_3} $ 8. $A \Leftrightarrow TH > NaHiD_{VERC}(X, Y)$	$ax_1 = X_1 + X_2, \quad ay_1 = Y_1 + Y_2, \quad M_X = \frac{ax_1 + X_3}{4}$ $M_Y = \frac{ay_1 + Y_3}{4}, \quad mx_1 = X_1^2, \quad mx_2 = X_2^2$ $mx_3 = X_3^2, \quad my_1 = Y_1^2, \quad my_2 = Y_2^2, \quad my_3 = Y_3^2$ $(M_X)^2 = M_X \times M_X, \quad (M_Y)^2 = M_Y \times M_Y$ $amx_1 = mx_1 + mx_2, \quad amy_1 = my_1 + my_2$ $M_{X^2} = \frac{amx_1 + mx_3}{4}, \quad M_{Y^2} = \frac{amy_1 + my_3}{4}$ $V_X = M_{X^2} - (M_X)^2 , \quad V_Y = M_{Y^2} - (M_Y)^2 $ $SD_X = \sqrt{V_X}, \quad SD_Y = \sqrt{V_Y}$ $MSD_X = M_X - SD_X , \quad MSD_Y = M_Y - SD_Y $ $DX_1 = MSD_X - X_1 , \quad DY_1 = MSD_Y - Y_1 $ $DX_2 = MSD_X - X_2 , \quad DY_2 = MSD_Y - Y_2 $ $DX_3 = MSD_X - X_3 , \quad DY_3 = MSD_Y - Y_3 $ $D_1 = DX_1 + DY_1, \quad D_2 = DX_2 + DY_2, \quad D_3 = DX_3 + DY_3$ $N_1 = X_1 - Y_1 , \quad N_2 = X_2 - Y_2 , \quad N_3 = X_3 - Y_3 $ $Q_1 = \frac{N_1}{D_1}, \quad Q_2 = \frac{N_2}{D_2}, \quad Q_3 = \frac{N_3}{D_3}$ $aQ_1 = Q_1 + Q_2, \quad aQ_2 = \frac{aQ_1 + Q_3}{4}$ $NaHiD_{VERC} = 1 - aQ_2 , \quad aT = NaHiD_{VERC} - TH$
--	---

(a) Cálculos da correlação na forma aritmética

(b) Cálculos da correlação na forma de operações em *hardware*

Figura 3.1 – Operações que o Módulo em *hardware* efetua

e *start* (que são os sinais que indicaram o início da detecção e as mudanças de ciclos). Esse componente abstrai toda a combinação lógica que foi implementado nos componentes internos, sendo este de suma importância para realizar a junção de entradas e saídas entre componentes internos, módulos e gerenciadores. O código do módulo Nahid está no apêndice 6.1.

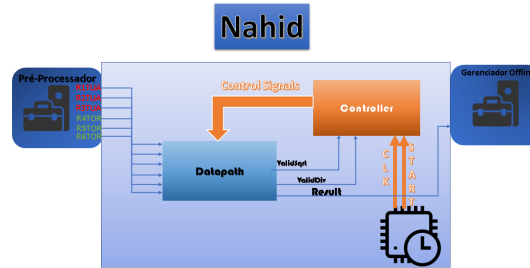


Figura 3.2 – Módulo Nahid

3.1.1 Datapath

O componente responsável por alocar todos os componentes que realizam as operações da detecção é o Datapath. Por isso, o mesmo comporta no mínimo um dos componentes de mais baixo nível, que serão descritos posteriormente. O Datapath recebe os vetores de entradas (perfil normal e instâncias de tráfego em análise) do Nahid, pois esses dados são selecionados, tratados e registrados pelos componentes internos do Datapath. Porém para isso é necessário que seja indicado ao componente quais são as entradas a serem processadas num dado ciclo, por isso o Datapath recebe como entrada seletores provindos do Controller. Entretanto, o Datapath possui saídas para o controller, pois em alguns ciclos é necessário de alguma confirmação de algum componente interno ao

Datapath, além disso a saída do sistema será um resultado registrado num componente interno e será em enviado ao componente de mais alto nível (Nahid). Segue abaixo, uma breve explicação de cada componente do Datapath. O código do Datapath está no apêndice 6.2.

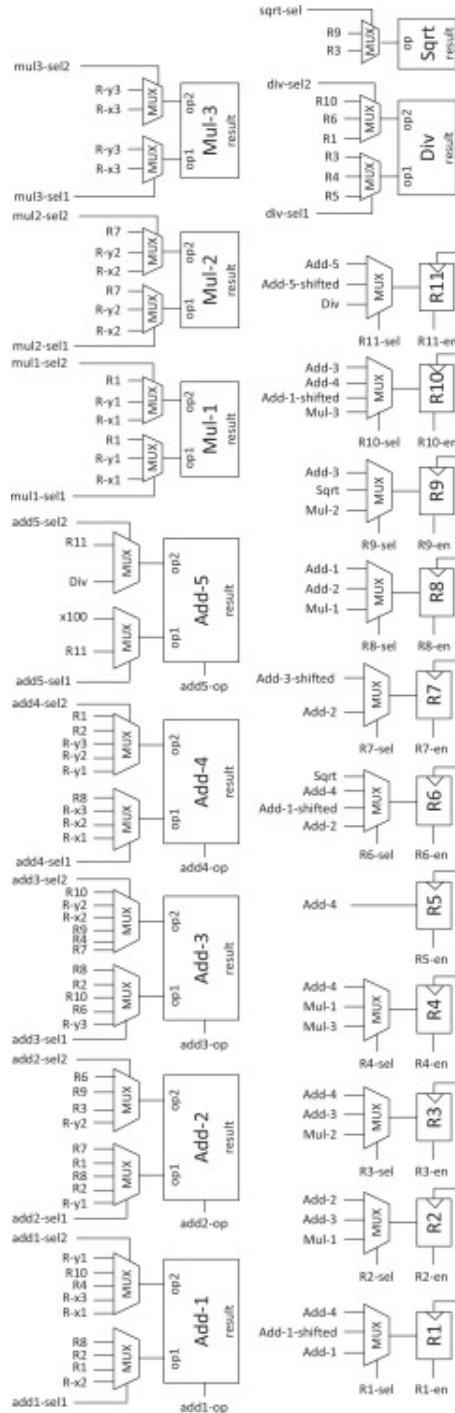


Figura 3.3 – Design do Datapath desenvolvido por (HOQUE; KASHYAP; BHATTACHARYYA, 2017)

3.1.1.1 Extend

Esse componente possui a função de estender o tamanho de uma palavra de bits de qualquer tamanho (menor que 23), para uma palavra de mesmo conteúdo com tamanho de 23 bits. Basicamente, ele completa com 0's o a esquerda da palavra, até o tamanho desta ser de 23 bits. Esse componente é de suma importância nas operações de soma, uma vez que o componentes de soma Adder possuem entradas de tamanho de 23 bits. O Datapath possui 12 componentes do tipo extend. Esse componente é do tipo de mudança no tamanho da palavra e em um ciclo ele completa sua computação. O código do componente estende está no apêndice 6.3.

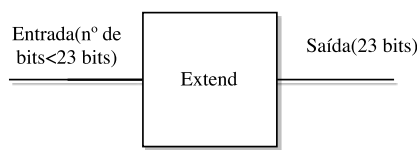


Figura 3.4 – Componente extend

3.1.1.2 Reduce

Esse componente possui a função de reduzir o tamanho de uma palavra de bits de qualquer tamanho (maior que 11), para uma palavra de mesmo conteúdo com tamanho de 11 bits. Basicamente, ele pega os 11 bits mais significativos da palavra. Esse componente é de suma importância nas operações de multiplicação, uma vez que o componentes de multiplicação(Mul) possuem entradas de tamanho de 11 bits. O Datapath possui 4 componentes do tipo reduce. Esse componente é do tipo de mudança no tamanho da palavra e um em ciclo ele completa sua computação. O código do componente estende está no apêndice 6.4.

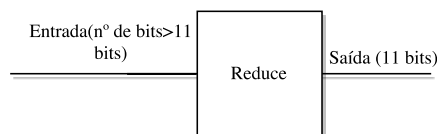


Figura 3.5 – Componente Reduce

3.1.1.3 Mux

Esse componente possui a função de ter na entrada várias opções, porém a cada iteração, existe um seletor que indica que entrada será utilizada num dado momento, levando para saída do componente essa escolha. Esse componente é de suma importância para reutilização de componentes, deixando a arquitetura mais enxuta e coesa, sendo muito utilizado no Controller, pois para cada ciclo existem determinadas escolhas nesses multiplexadores. É importante ressaltar, que no Datapath existem mux de diversos

tamanhos (2 entradas, 4 entradas e 6 entradas), isso está diretamente relacionado com o componente que recebe a saída do mux, pois quanto mais ele pode ser utilizado por entradas diferentes, maior será o número de entradas no multiplexador combinado a esse componente. Devido aos multiplexadores terem grande importância na reutilização de componentes, existem duas principais frentes de alocação de mux no Datapath. Nas operações aritméticas (Multiplicadores e Somadores) e de registro (Registradores). Esse componente é do tipo de seleção de dados e em um ciclo ele completa sua computação. O código do componente mux está no apêndice 6.5.

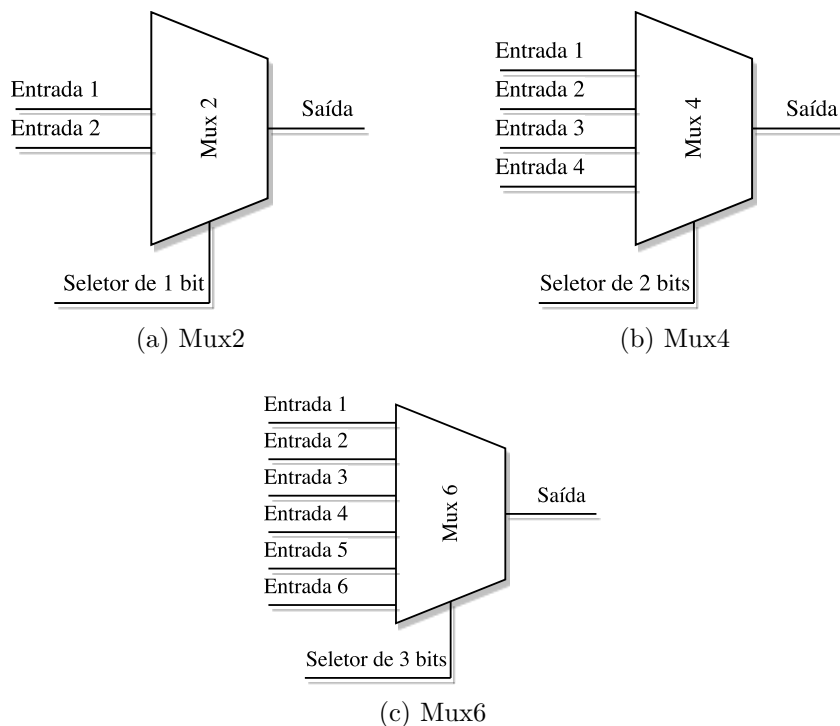


Figura 3.6 – Tipos do componente Mux no módulo

3.1.1.4 Mul

Esse componente possui a função de receber duas entradas, e realizar a multiplicação aritmética das mesmas, gerando uma saída do resultado. Foi utilizado um mul que recebe entradas de 11 bits podendo gerar até 22 bits no resultado dessa multiplicação. Existem 3 multiplicadores no Datapath, que em nosso módulo realizam operação de quadrado de um número, ou seja as multiplicações tem as mesmas entradas num dado momento. Esse componente é do tipo operações aritméticas e em um ciclo ele completa sua computação. O código do componente mul está no apêndice 6.6.

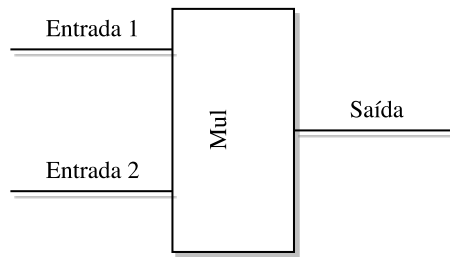


Figura 3.7 – Componente Mul

3.1.1.5 Adder

Esse componente possui a função de receber duas entradas, e realizar a soma aritmética das mesmas, gerando uma saída do resultado. Foi utilizado um adder que recebe entradas de 23 bits podendo gerar até 24 bits no resultado dessa soma. Vale ressaltar que o componente possui 4 modos de operação, o que caracteriza diferentes formas de somar as entradas. Esses modos são:

- O modo 0: A soma padrão de dois números positivos.
- O modo 1: A soma de dois números positivos, com o resultado dividido por 4.
- O modo 2: O módulo de dois números positivos
- O modo 3: O módulo de dois números positivos, com o resultado dividido por 4.

Esses modos de operação, são necessários pelos diversos “cálculos” que são necessários na formulação da correlação que o módulo implementa, por isso é necessário que algum componente implemente essas adições, sendo escolhido o Somador. Existem 5 somadores no Datapath, que no módulo realizam operações de adição necessárias. Vale ressaltar que o seletor de operação, também é uma entrada do adder. Esse componente é do tipo operações aritméticas e em um ciclo ele completa sua computação. O código do componente adder está no apêndice 6.7.

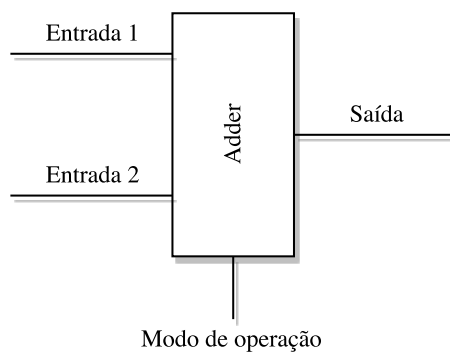


Figura 3.8 – Componente Adder

3.1.1.6 Divider

Esse componente possui a função de receber duas entradas, e realizar a divisão aritmética das mesmas, gerando uma saída do resultado. Esse componente possui certas peculiaridades, pois uma divisão em hardware é mais custosa, pela existência de números fracionários nos resultados de divisões decimais, que interferem diretamente nas operações e no resultado do módulo. Por isso existe a necessidade da utilização de alguma representação numérica que possa trazer resultados fracionários, foi utilizado nesse caso representação em ponto fixo como falado anteriormente. O componente Divider utilizado foi o IP core da *xilinx* “*Divider Generator*”, que foi configurado da seguinte forma:

- Dividendo: 12 bits
- Divisor: 12 bits
- Quociente: 12 bits
- Parte Fracionária: 8 bits

Em outras palavras, temos 2 entradas de 12 bits e uma saída de 20 bits, porém na representação de ponto fixa, tem-se 12.8(12 números na parte inteira e 8 na decimal). Além dessas instâncias, existem dois sinais de controle no módulo, o “*valid in*” (responsável por indicar que o componente está pronto para receber as entradas e iniciar a computação) e o “*valid out*” (responsável por indicar que o componente acabou de realizar a operação por completo e já tem o resultado), esses sinais são de suma importância para a organização do módulo e regulação dos ciclos. Existe apenas um componente do tipo do Divider, que realiza divisões que não possuem divisor diferente de potências de 2 (pois pode-se utilizar mecanismos mais simples, para realizar essas divisões, mantendo o resultado no universo dos inteiros). Esse componente é do tipo operações aritméticas e em 22 ciclos ele completa sua computação.

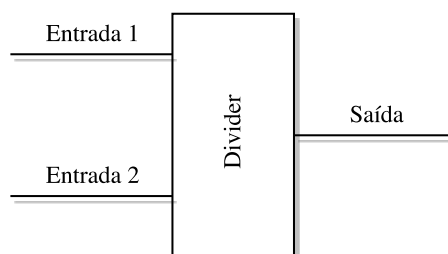


Figura 3.9 – Componente Divider

3.1.1.7 Sqrt

Esse componente possui a função de receber uma entrada, e realizar a raiz quadrada aritmética da mesma, gerando uma saída do resultado. Esse componente possui resultados

inteiros, porém o resultado é um inteiro aproximado para números que não possuem raízes fechadas, pois é possível os números de entradas não serem quadrados perfeitos e o resultado da raiz ser número com casas decimais. Para realizar a busca e aproximação de resposta, é necessário algum algoritmo de busca dessa raiz, gerando um certo custo de ciclos, como no divisor. Para tal o componente *sqrt* utilizado foi o *IP core* da *xilinx* “*Cordic*” (função Raiz), que foi configurado da seguinte forma:

- Entrada: 22 bits
- Saída: 12 bits

Logo, temos 1 entrada de 22 bits e uma saída de 12 bits. Além dessas instâncias, existem dois sinais de controle no módulo, o “*valid in*” (responsável por indicar que o componente está pronto para receber a entrada e iniciar a computação) e o “*valid out*” (responsável por indicar que o componente acabou de realizar a operação por completo e já tem o resultado), esses sinais são de suma importância para a organização do módulo e regulação dos ciclos. Existe apenas um componente do tipo do *sqrt*, esse componente é do tipo operações aritméticas e em 6 ciclos ele completa sua computação.

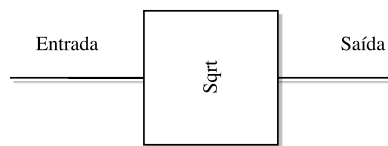


Figura 3.10 – Componente Sqrt

3.1.1.8 Register

Esse componente possui a função de receber uma entrada, e armazenar o valor recebido a partir da próxima subida do *clock* e atualizar o valor que estiver na entrada na próxima subida do *clock*. De forma que pelo menos durante um ciclo o registrador terá o valor recebido num dado momento, por isso podemos considerar o conjunto de registradores como a memória do módulo. Os registradores são de suma importância para a realizar as operações, uma vez que não para realizar todos os passos da correlação, “guardam-se” variáveis de uma operação para utilizar nas próximas operações. O Registrador implementado por padrão recebe entradas de 24 bits e a saída tem o mesmo tamanho, porém existem registradores específicos que possuem tamanhos diferentes do padrão, por serem usados para em fins específicos. Além da entrada, o Registrador recebe o sinal *enable* (responsável para habilitar ou não o registro do componente, no próximo ciclo de *clock*) e *clr* (responsável por zerar o registro do componente, no próximo ciclo de *clock*). Existem 11 registradores no Datapath, que realizam o registro necessários no módulo. Esse componente é do tipo registro e em um ciclo ele completa sua computação. O código do componente register está no apêndice 6.8.

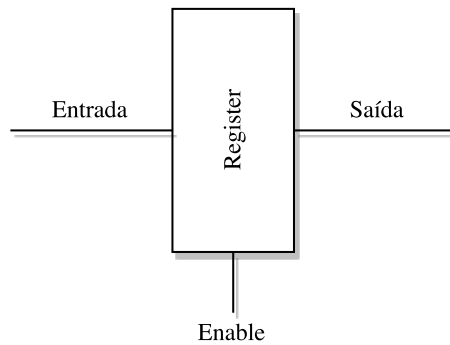


Figura 3.11 – Componente Register

3.1.2 Controller

O componente responsável por organizar os ciclos das operações do Módulo Nahid é o Controller. Quais componentes do Datapath que serão utilizados num determinado ciclo de computação e em que ciclo teremos os resultados de uma dada operação, são as principais funções desse componente. Esse componente, utiliza o conceito de máquina de estados, para realizar uma computação cíclica e prática, afim dos cálculos serem realizados de forma otimizada. O Controller recebe o *clk* (*clock* do sistema) e o *start* (sinal que indica o início do módulo) do Nahid, para que o componente garanta que o sistema está síncrono. Além dessas entradas, como dito anteriormente, recebe os sinais “*valid out*” dos componentes Divider e Sqrt. O Controller possui saídas para o Datapath, para indicar a esse componente o que será utilizado num dado ciclo. A estrutura do controller pode ser dividida em dois cases:

1. Case de operações: Esse case é responsável por indicar todas as operações que serão feitos em todos os ciclos.
2. Case de transições de ciclos: Esse case é responsável por indicar quando haverá as transições de um ciclo para outro.

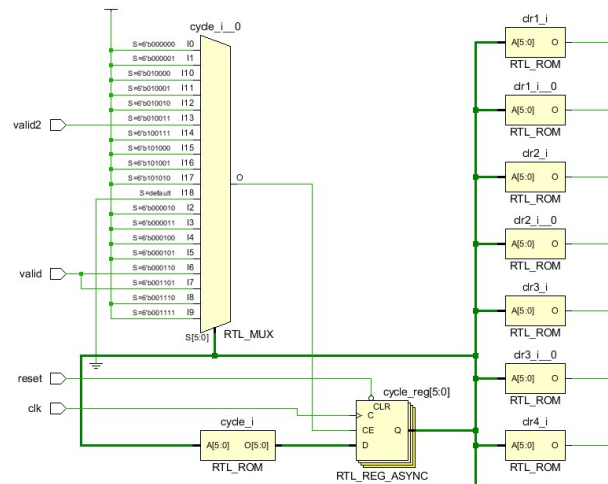


Figura 3.12 – Esquemático do controller, ciclos de entrada e saídas para as entradas do datapath num dado ciclo de *clock*

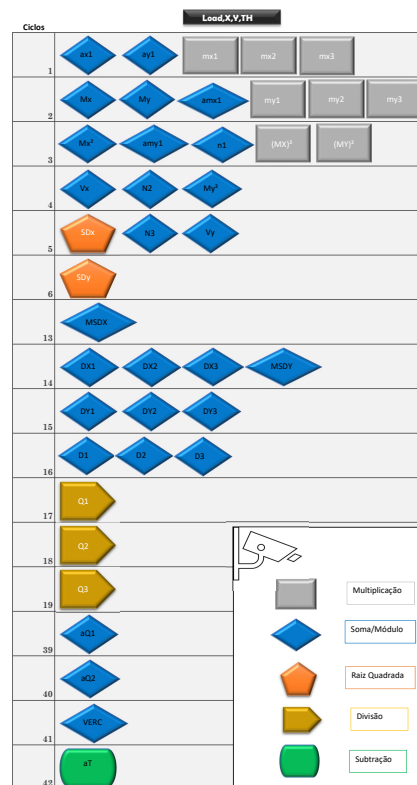


Figura 3.13 – Ciclos do Módulo Nahid 3

A figura 3.13, representa os ciclos que o módulo Nahid segue, para realizar a computação de detecção. Vale ressaltar que existem operações que possuem latência de vários ciclos de *clock*. O código do Controller está no apêndice 6.9.

4 Resultados Experimentais

O *ip soft core* proposto possui duas grandes principais motivações: desempenho e precisão nas detecções de ataques *DDoS*. Para isso foi feita a implementação do core a nível *RTL* (*register-transfer level*), seguido pela a síntese. Diante disso, é possível quantificar a utilização dos componentes do módulo dentro de uma *FPGA* (descrição). As tabelas mostram o relatório de utilização do artigo (HOQUE; KASHYAP; BHATTACHARYYA, 2017) (um trabalho similar) e o trabalho proposto.

Tabela 1 – Relatório de Utilização do artigo de comparação

Tipo	Usado	Disponível	Utilização%
CLB LUTs*	1905	28800	0.60
LUT as Logic	1891	28800	0.60
LUT as Memory	14	7860	0.01
LUT as Shift Register	14		
CLB Registers	1131	28800	0.3
Register as Flip Flop	1255	2386	52
Frequency	118Mhz		

A tabela 1 traz o valores de alguns componentes da *FPGA* utilizada, mostrando estimativas de utilização, uma vez que é possível identificar através da síntese o que seria utilizado na placa escolhida, além de termos conhecimento dos componentes disponíveis. Semelhantemente, A tabela 2 traz valores de utilização, porém para o módulo implementado nesse trabalho. Comparando as duas tabelas, podemos ver uma menor utilização dos componentes, no trabalho proposto. Além de uma maior frequência, o que garante uma detecção em um menor tempo.

Tabela 2 – Relatório de Utilização do trabalho proposto

Tipo	Usado	Disponível	Utilização%
CLB LUTs*	1302	216960	0.60
LUT as Logic	1301	216960	0.60
LUT as Memory	1	99840	<0.01
LUT as Shift Register	1		
CLB Registers	1180	433920	0.27
Register as Flip Flop	1122	433920	0.26
Frequency 120 MHZ			

Outro importante fator de validação do módulo seria o quão os valores se aproximam dos valores reais. Para isso, foi feito um *TestBench* (um teste não sintetizável, no qual verifica o funcionamento, através de simulações) em *systemverilog*. Foi comparado os valores adquiridos nos testes com valores calculados no *Matlab*, com os devidos truncamentos das operações da figura —.

Tabela 3 – Comparativo de resultados do Nahid implementado em software e hardware

Detecção	Matlab	Módulo	Erro (%)
1 (P1=365,P2=252,P3=953,D1=140,D2=200,D3=970)	0,82493	0,82812	1
2 (P1=128,P2=515,P3=852,D1=130,D2=470,D3=970)	0,96874	0,96625	1.02
3 (P1=150,P2=300,P3=853,D1=123,D2=340,D3=876)	0,95585	0,95468	0.9

Na tabela 3 as siglas P1,P2 e P3 são os vetores de perfil normal, já as siglas D1,D2 e D3, são os vetores que foram examinados. Podemos ver que mesmo que as taxas de erros são bem pequenas, o que mostra uma precisão considerável nos resultados do módulo. A figura 4.1, mostra um exemplo de simulação nos seus últimos ciclos, vale a pena ressaltar, que com o *threshold*, pode-se trazer a conclusão se existe um ataque ou não naquela para aqueles valores.

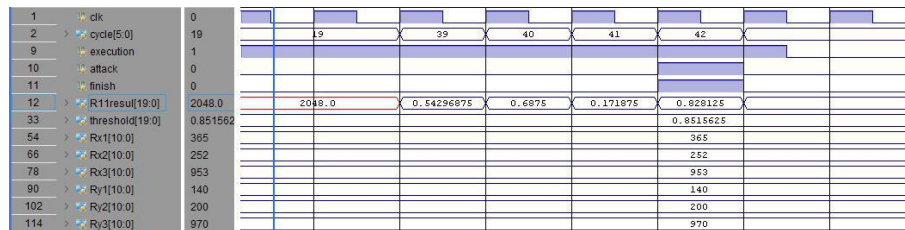


Figura 4.1 – Simulação 1 da tabela 3

Tabela 4 – Tempo de detecção

Detector	Artigo de comparação	Trabalho Proposto	Software(Matlab)
Tempo de Detecção	354 ns	350 ns	296 μ s

A tabela 4 mostra que existe um pequeno ganho, desse trabalho em relação ao artigo de comparação (também implementado em hardware), e um ganho significativo em relação ao detector em software.

5 Conclusões e Trabalhos Futuros

Foi construído um módulo em hardware que se mostrou capaz de detectar em tempo real, uma vez que foi comparado com um trabalho já validado e com soluções em softwares e obteve comportamento similar ou melhor que essas soluções.

Como podemos ver na tabela, 4 temos um ganho de tempo, significativo do módulo em hardware em relação a solução em *software*. Além disso não temos uma precisão com erros pequenos, como podemos ver na tabela 3 .

Como podemos ver nas tabelas 2, 1, 4 e 3 temos um ganho, tanto de utilização mínima de componentes como no tempo de detecção. Vale ressaltar que a *FPGA* que esse trabalho utiliza é a *Artix Série 7* , como vimos no 2 possui baixa potência. Já o artigo de comparação usa uma *Vertex Série 5*, que possui alta performance. Mesmo assim, os resultados desse trabalho foram melhores.

Esses resultados foram alcançados devido a estudos prévios de aritmética de números binários, e a implementação dos mesmos. Além disso foram utilizados métodos de otimização de tempo nas operações aritméticas.

Além disso foram utilizados componentes, da IDE vivado da Xilinx, que foram essenciais para a realização de detecção, pois as operações que esses ip's implementam, são complexas e custosas. Além disso existe uma maior confiabilidade nos resultados, mediante a credibilidade da ferramenta utilizada.

Referências

AHMED, H. A. *et al.* Shifting-and-scaling correlation based biclustering algorithm. **IEEE/ACM Transactions on Computational Biology and Bioinformatics**, v. 11, n. 6, p. 1239–1252, Nov 2014. ISSN 1545-5963.

Citado na página 18.

ALECRIM, E. **Ataques DoS (Denial of Service) e DDoS (Distributed DoS)**. [S.l.]: Disponível na Internet em< <http://www.infowester.com/coll20904.php>> em, 2008. Citado 2 vezes nas páginas 14 e 17.

DEMETRIO, A. *et al.* Arquitetura fpgas e cplds da xilinx.

Citado na página 22.

FEITOSA, E. *et al.* Tráfego internet não desejado: Conceitos, caracterização e soluções. **Proc. VIII SBSeg, SBC**, p. 91–137, 2008.

Citado na página 17.

HANDLEY, M. J.; RESCORLA, E. Internet denial-of-service considerations. 2006.

Citado na página 17.

HEATH, S. **Embedded systems design**. [S.l.]: Newnes, 2002.

Citado na página 22.

HOQUE, N. *et al.* Real-time DDoS Attack Detection Using FPGA. **Computer Communications**, v. 110, n. Supplement C, p. 48 – 58, 2017. ISSN 0140-3664. Disponível em: <<http://www.sciencedirect.com/science/article/pii/S0140366416306442>>. Citado 6 vezes nas páginas 8, 14, 19, 22, 26 e 34.

MANDIA, K.; PROSISE, C. Hackers resposta e contraataque. **Rio de Janeiro: Campus**, 2001. Citado 2 vezes nas páginas 14 e 17.

MIRANDA, L. C. *et al.* O computador na educação do eletrotécnico. Florianópolis, SC, 2002.

Citado na página 19.

PINHEIRO, J. M. dos S. Ameaças e ataques aos sistemas de informação: Prevenir e antecipar. **Cadernos UniFOA**, v. 3, n. 5, p. 11–21, 2017.

Citado na página 17.

PRZYBUS, B. Xilinx redefines power, performance, and design productivity with three new 28 nm fpga families: Virtex-7, kintex-7, and artix-7 devices. **Xilinx White Paper**, 2010.

Citado na página 22.

STEIN, L. D. **World Wide Web Security FAQ**. [S.l.]: Lincoln D. Stein., 2002.

Citado na página 18.

WOODS, J. B. R. *et al.* Reconfigurable computing: Architectures, tools and applications. Springer, 2008.

Citado na página 20.

YU, S. *et al.* Discriminating ddos attacks from flash crowds using flow correlation coefficient. **IEEE Transactions on Parallel and Distributed Systems**, IEEE, v. 23, n. 6, p. 1073–1080, 2012.

Citado na página 18.

6 Códigos do Módulo

6.1 Código Nahid

```

1  module Nahid(clk,reset,Rx1,Rx2,Rx3,Ry1,Ry2,Ry3,R11resul,threshold);
2  //entrada do controller
3  input  clk,reset;
4  //entrada do datapath
5  input  [10:0] Rx1,Rx2,Rx3,Ry1,Ry2,Ry3;
6  output[19:0] R11resul;
7  input  [19:0] threshold;
8  reg  selmul3_1,selmul3_2,seladd5_2,seladd5_1,selr7,selsqrt,clr1,clr2,
9  clr3,clr4,clr5,clr6,clr7,clr8,clr9,clr10,clr11,enable1,enable2,
10 enable3,enable4,enable5,enable6,enable7,enable8,enable9,enable10,
11 enable11,valid,insqrt,valid2,inbottom,intop;
12 reg  clrx1,clrx2,clrx3,clry1,clry2,clry3,enablex1,enablex2,enablex3,
13 enabley1,enabley2,enabley3;
14 reg  [1:0] selmul2_2,selmul2_1,selmul1_2,selmul1_1,seladd2_2,seladd1_1,
15 seldiv_2,seldiv_1,selr11,selr10,selr9,selr8,selr6,selr4,selr3,
16 selr2,selr1;
17 reg  [2:0] seladd4_2,seladd4_1,seladd3_2,seladd3_1,seladd2_1,seladd1_2;
18 reg  [1:0] opadd1,opadd2,opadd3,opadd4,opadd5;
19 reg  [19:0] R11resul;
20 reg  attack,finish;
21 Datapath data1 (.Rx1(Rx1),.Rx2(Rx2),.Rx3(Rx3),.Ry1(Ry1),.Ry2(Ry2),
22 .Ry3(Ry3),.selmul3_1(selmul3_1),.selmul3_2(selmul3_2),
23 .seladd5_2(seladd5_2),.seladd5_1(seladd5_1),.selr7(selr7),
24 .selsqrt(selsqrt),.selmul2_2(selmul2_2),.selmul2_1(selmul2_1),
25 .selmul1_2(selmul1_2),.selmul1_1(selmul1_1),.seladd2_2(seladd2_2),
26 .seladd1_1(seladd1_1),.seldiv_2(seldiv_2),.seldiv_1(seldiv_1),
27 .selr11(selr11),.selr10(selr10),.selr9(selr9),.selr8(selr8),
28 .selr6(selr6),.selr4(selr4),.selr3(selr3),.selr2(selr2),
29 .selr1(selr1),.seladd4_2(seladd4_2),.seladd4_1(seladd4_1),
30 .seladd3_2(seladd3_2),.seladd3_1(seladd3_1),.seladd2_1(seladd2_1),
31 .seladd1_2(seladd1_2),.clk(clk),.clr1(clr1),.clr2(clr2),
32 .clr3(clr3),.clr4(clr4),.clr5(clr5),.clr6(clr6),.clr7(clr7)
33 ,.clr8(clr8),.clr9(clr9),.clr10(clr10),.clr11(clr11),
34 .enable1(enable1),.enable2(enable2),.enable3(enable3),
35 .enable4(enable4),.enable5(enable5),.enable6(enable6),

```



```

36 .enable7(enable7),.enable8(enable8),.enable9(enable9),
37 .enable10(enable10),.enable11(enable11), .opadd1(opadd1),.opadd2(opadd2),
38 .opadd3(opadd3),.opadd4(opadd4),.opadd5(opadd5),
39 .clrx1(clrx1),.clrx2(clrx2),.clrx3(clrx3),
40 .clry1(clry1),.clry2(clry2),
41 .clry3(clry3),.enablex1(enablex1),.enablex2(enablex2),
42 .enablex3(enablex3),.enabley1(enabley1),.enabley2(enabley2),
43 .enabley3(enabley3),.insqrt(insqrt),.valid(valid),
44 .valid2(valid2),.inbottom(inbottom),
45 .intop(intop),.R11resul(R11resul));
46
47 controller c1 (.clk(clk),.reset(reset),.selmul3_1(selmul3_1),
48 .selmul3_2(selmul3_2),.seladd5_2(seladd5_2),.seladd5_1(seladd5_1),
49 .selr7(selr7),.selsqrt(selsqrt),.selmul2_2(selmul2_2),
50 .selmul2_1(selmul2_1),.selmul1_2(selmul1_2),.selmul1_1(selmul1_1),
51 .seladd2_2(seladd2_2),.seladd1_1(seladd1_1),.seldiv_2(seldiv_2),
52 .seldiv_1(seldiv_1),.selr11(selr11),.selr10(selr10),.selr9(selr9),
53 .selr8(selr8),.selr6(selr6),.selr4(selr4),.selr3(selr3),
54 .selr2(selr2),.selr1(selr1),.seladd4_2(seladd4_2),
55 .seladd4_1(seladd4_1),.seladd3_2(seladd3_2),
56 .seladd3_1(seladd3_1),.seladd2_1(seladd2_1),
57 .seladd1_2(seladd1_2),.clr1(clr1),.clr2(clr2),
58 .clr3(clr3),.clr4(clr4),.clr5(clr5),.clr6(clr6),
59 .clr7(clr7),.clr8(clr8),.clr9(clr9),.clr10(clr10),
60 .clr11(clr11),.enable1(enable1),.enable2(enable2),
61 .enable3(enable3),.enable4(enable4),.enable5(enable5),
62 .enable6(enable6),.enable7(enable7),.enable8(enable8),
63 .enable9(enable9),.enable10(enable10),.enable11(enable11),
64 .opadd1(opadd1),.opadd2(opadd2),.opadd3(opadd3),.opadd4(opadd4)
65 ,.opadd5(opadd5),.clrx1(clrx1),.clrx2(clrx2),.clrx3(clrx3),
66 .clry1(clry1),.clry2(clry2),.clry3(clry3),.enablex1(enablex1),
67 .enablex2(enablex2),.enablex3(enablex3),.enabley1(enabley1),
68 .enabley2(enabley2),.enabley3(enabley3),
69 .insqrt(insqrt),.valid(valid),.valid2(valid2),.inbottom(inbottom),
70 .intop(intop),.finish(finish));
71
72 always_comb
73     begin
74         begin
75             attack=0;
76             if (finish)
77                 if(threshold>R11resul)

```

```

78                                     attack=1;
79                                     else
80                                     attack=0;
81                                     end
82                             end
83     endmodule

```

6.2 Código Datapath

```

1
2  module Datapath(Rx1,Rx2,Rx3,Ry1,Ry2,Ry3,
3  selmul3_1,selmul3_2,seladd5_2,seladd5_1,selr7,selsqrt,selmul2_2
4  ,selmul2_1,selmul1_2,selmul1_1,seladd2_2,seladd1_1,seldiv_2
5  ,seldiv_1,selr11,selr10,selr9,selr8,selr6,selr4,selr3,selr2
6  ,selr1,seladd4_2,seladd4_1,seladd3_2,seladd3_1,seladd2_1
7  ,seladd1_2,clk,clr1,clr2,clr3,clr4,clr5,clr6,clr7,clr8,clr9
8  ,clr10,clr11,clrx1,clrx2,clrx3,clry1,clry2,clry3,enable1
9  ,enable2,enable3,enable4,enable5,enable6,enable7,enable8
10 ,enable9,enable10,enable11,enabllex1,enabllex2,enabllex3,enabley1
11 ,enabley2,enabley3,datasqrt,datadiv,datar11,datar10,datar9
12 ,datar8,datar7,datar6,datar5,datar4,datar3,datar2,datar1
13 ,opadd1,opadd2,opadd3,opadd4,opadd5,R11resul
14 ,valid,insqrt,valid2,inbottom,intop);
15 input  [10:0] Rx1,Rx2,Rx3,Ry1,Ry2,Ry3;
16 //Seletores
17 input  selmul3_1,selmul3_2,seladd5_2,seladd5_1,selr7,selsqrt,clk,clr1
18 ,clr2,clr3,clr4,clr5,clr6,clr7,clr8,clr9,clr10,clr11,clrx1
19 ,clrx2,clrx3,clry1,clry2,clry3,enable1,enable2,enable3,enable4
20 ,enable5,enable6,enable7,enable8,enable9,enable10,enable11
21 ,enabllex1,enabllex2,enabllex3,enabley1,enabley2,enabley3,datasqrt
22 ,datadiv,datar11,datar10,datar9,datar8,datar7,datar6,datar5
23 ,datar4,datar3,datar2,datar1,insqrt,inbottom,intop;
24 input  [1:0] selmul2_2,seladd4_1,selmul2_1,selmul1_2,selmul1_1,seladd2_2
25 ,seladd1_1,seldiv_2,seldiv_1,selr11,selr10,selr9,selr8
26 ,selr6,selr4,selr3,selr2,selr1,opadd1,opadd2,opadd3,opadd4,opadd5;
27 input  [2:0] seladd4_2,seladd3_2,seladd3_1,seladd2_1,seladd1_2;
28 output valid,valid2;
29 output [19:0] R11resul ;
30 wire  clrx1,clrx2,clrx3,clry1,clry2,clry3,enabllex1,enabllex2,enabllex3
31 ,enabley1,enabley2,enabley3,valid,valid2;
32 wire  [23:0] op2add5,op1add5,op2add4,op1add4,op2add3,op1add3
33 ,op2add2,op1add2,op2add1,op1add1;

```

```

34 wire [23:0] add5resul,add4resul,add3resul,add2resul,add1resul,add5shifted
35 ,add4shifted,add3shifted,add2shifted,add1shifted;
36 wire [23:0] mul3resul,mul2resul,mul1resul,
37 op2div,op1div,sqrtresul,
38 div;
39 wire [10:0] op2mul3,op1mul3,op2mul2,op1mul2,op2mul1,op1mul1,x1r,x2r,x3r
40 ,y1r,y2r,y3r;
41 wire [22:0] Rx1new,Rx2new,Rx3new,Ry1new,Ry2new,Ry3new,R7ex,R5ex
42 ,R11ex,divnew;
43 wire [19:0] divresul;
44 wire [23:0] opR1,opR2,opR3,opR4,opR5,opR6,opR7,opR8,opR9,opR10,opR11,
45 opsqrt,sqrtresulnew;
46 wire [23:0] R10resul,R9resul,R8resul,R7resul,R6resul,R5resul,R4resul,
47 R3resul,R2resul,R1resul;
48 wire [10:0] R7resulnew,R1resulnew;
49 wire [11:0] op1divnew,op2divnew;
50 //extends (Modulo para ajustar entradas de 11 para 23 bits)
51 extend ex1 (.a(x1r),.y(Rx1new));
52 extend ex2 (.a(x2r),.y(Rx2new));
53 extend ex3 (.a(x3r),.y(Rx3new));
54 extend ey1 (.a(y1r),.y(Ry1new));
55 extend ey2 (.a(y2r),.y(Ry2new));
56 extend ey3 (.a(y3r),.y(Ry3new));
57 extend #(12) er7 (.a(R7resul),.y(R7ex));
58 extend #(16) sqrt (.a(sqrtresul),.y(sqrtresulnew));
59 extend er5 (.a(R5resul),.y(R5ex));
60 extend er1 (.a(R1resul),.y(R1ex));
61 extend #(20) er11 (.a(R11resul),.y(R11ex));
62 extend #(20) ediv (.a(divresul),.y(divnew));
63 //Reduce (Modulo para ajustar Saidas de 24 para 12 bits)
64
65 reduce rr7 (.a(R7resul),.y(R7resulnew));
66 reduce rr1 (.a(R1resul),.y(R1resulnew));
67
68
69 reduce rop1 (.a(op1div),.y(op1divnew));
70 reduce rop2 (.a(op2div),.y(op2divnew));
71
72
73
74 /*Multiplexadores parte 1
75 Primeira coluna da arquitetura

```

```

76  -- 0 que recebe?
77  Entradas dos dados e a
78  realimentacao de registradores e variaveis,
79  alem disso coloca as entradas em nos valores dos adders.
80  */
81
82  //Mux dos multiplicadores
83  //Multiplexadores do Multiplicador 3
84  mux2 mux2mul3(.a(y3r),.b(x3r),.sel(selmul3_2),.y(op2mul3));
85  mux2 mux1mul3(.a(y3r),.b(x3r),.sel(selmul3_1),.y(op1mul3));
86  //Multiplexadores do Multiplicador 2
87  mux4 mux2mul2(.a(R7resulnew),.b(y2r),.c(x2r),.d(0),.sel(selmul2_2)
88  ,.y(op2mul2));
89  mux4 mux1mul2(.a(R7resulnew),.b(y2r),.c(x2r),.d(0),.sel(selmul2_1)
90  ,.y(op1mul2));
91  //Multiplexadores do Multiplicador 1
92  mux4 mux2mul1(.a(R1resulnew),.b(y1r),.c(x1r),.d(0),.sel(selmul1_2)
93  ,.y(op2mul1));
94  mux4 mux1mul1(.a(R1resulnew),.b(y1r),.c(x1r),.d(0),.sel(selmul1_1)
95  ,.y(op1mul1));
96
97
98
99  //Mux dos Somadores
100
101  //Multiplexadores do Somador 5
102  mux2 mux2add5(.a(R11ex),.b(divnew),.sel(seladd5_2),.y(op2add5));
103  mux2 mux1add5(.a(24'b0000000000000000100000000),.b(R11ex)
104  ,.sel(seladd5_1),.y(op1add5));
105  //Multiplexadores do Somador 4
106  mux6 mux2add4(.a(R1resul),.b(R2resul),.c(Ry3new),.d(Ry2new)
107  ,.e(Ry1new),.f(0),.g(0),.h(0),.sel(seladd4_2),.y(op2add4));
108  mux4 mux1add4(.a(R8resul),.b(Rx3new),.c(Rx2new),.d(Rx1new)
109  ,.sel(seladd4_1),.y(op1add4));
110  //Multiplexadores do Somador 3
111  mux6 mux2add3(.a(R10resul),.b(Ry2new),.c(Rx2new),.d(R9resul)
112  ,.e(R4resul),.f(R7ex),.g(0),.h(0),.sel(seladd3_2),.y(op2add3));
113  mux6 mux1add3(.a(R8resul),.b(R2resul),.c(R10resul),.d(R6resul)
114  ,.e(Ry3new),.f(0),.g(0),.h(0),.sel(seladd3_1),.y(op1add3));
115  //Multiplexadores do Somador 2
116  mux4 mux2add2(.a(R6resul),.b(R9resul),.c(R3resul),.d(Ry2new)
117  ,.sel(seladd2_2),.y(op2add2));

```

```

118 mux6 mux1add2(.a(R7ex),.b(R1resul),.c(R8resul),.d(R2resul)
119 ,.e(Ry1new),.f(0),.g(0),.h(0),.sel(seladd2_1),.y(op1add2));
120 //Multiplexadores do Somador 1
121 mux6 mux2add1(.a(Ry1new),.b(R10resul),.c(R4resul),.d(Rx3new)
122 ,.e(Rx1new),.f(0),.g(0),.h(0),.sel(seladd1_2),.y(op2add1));
123 mux4 mux1add1(.a(R8resul),.b(R2resul),.c(R1resul),.d(Rx2new)
124 ,.sel(seladd1_1),.y(op1add1));
125
126 /*Multiplicadores e Somadores
127 segunda coluna da arquitetura
128 -- 0 que recebe?
129 Os dados que os multiplexadores parte 1 selecionam.
130 */
131 mul mul3 (.a(op1mul3),.b(op2mul3),.mul(mul3resul));
132 mul mul2 (.a(op1mul2),.b(op2mul2),.mul(mul2resul));
133 mul mul1 (.a(op1mul1),.b(op2mul1),.mul(mul1resul));
134
135 adder add5(.a(op1add5),.b(op2add5),.op(opadd5),.o(add5resul));
136 adder add4(.a(op1add4),.b(op2add4),.op(opadd4),.o(add4resul));
137 adder add3(.a(op1add3),.b(op2add3),.op(opadd3),.o(add3resul));
138 adder add2(.a(op1add2),.b(op2add2),.op(opadd2),.o(add2resul));
139 adder add1(.a(op1add1),.b(op2add1),.op(opadd1),.o(add1resul));
140
141 /*Multiplexadores parte 2
142 terceira coluna da arquitetura
143 -- 0 que recebe?
144 Saida dos modulos da parte 2 e variaveis.
145 */
146
147 //mux raiz quadrada
148 mux2r muxsqrt(.a(R9resul),.b(R3resul),.sel(selsqrt),.y(opsqrt));
149 //mux divisor
150 mux4r mux2div(.a(R10resul),.b(R6resul),.c(R1resul),.sel(seldiv_2)
151 ,.y(op2div));
152 mux4r mux1div(.a(R3resul),.b(R4resul),.c(R5ex)
153 ,.sel(seldiv_1),.y(op1div));
154 //mux dos registradores
155 mux4r muxr11(.a(add5resul),.b(add5resul),.c(divresul),.sel(selr11)
156 ,.y(opR11));
157
158 mux4r muxr10(.a(add3resul),.b(add4resul),.c(add1resul),.d(mul3resul)
159 ,.sel(selr10),.y(opR10));

```

```

160
161 mux4r muxr9(.a(add3resul),.b(sqrtresulnew),.c(mul2resul),.sel(selr9)
162 ,.y(opR9));
163
164 mux4r muxr8(.a(add1resul),.b(add2resul),.c(mul1resul),.sel(selr8)
165 ,.y(opR8));
166
167 mux2r muxr7(.a(add3resul),.b(add2resul),.sel(selr7),.y(opR7));
168
169 mux4r muxr6(.a(sqrtresulnew),.b(add4resul),.c(add1resul),.d(add2resul)
170 ,.sel(selr6),.y(opR6));
171
172 mux4r muxr4(.a(add4resul),.b(mul1resul),.c(mul3resul),.sel(selr4)
173 ,.y(opR4));
174
175 mux4r muxr3(.a(add4resul),.b(add3resul),.c(mul2resul),.d(0)
176 ,.sel(selr3),.y(opR3));
177
178 mux4r muxr2(.a(add2resul),.b(add3resul),.c(mul1resul),.sel(selr2)
179 ,.y(opR2));
180
181 mux4r muxr1(.a(add4resul),.b(add1shifted),.c(add1resul),.sel(selr1)
182 ,.y(opR1));
183
184 /*Registradores, div e sqrt
185 quarta coluna da arquitetura
186 -- 0 que recebe?
187 Os dados que os multiplexadores parte 3 selecionam.
188 */
189 //div div1 (.a(op1divnew),.b(op2divnew),.div(divresul));
190
191 div_gen_1 divider (clk,inbottom,op2div,intop,op1div,valid2,divresul);
192
193 //sqrt sqrt1(.b(opsqrt),.square(sqrtresul));
194
195 cordic_1 sqrtcalc (clk,insqrt,opsqrt,valid,sqrtresul);
196
197
198 register r1 (.in(opR1),.clk(clk),.clr(clr1)
199 ,.enable(enable1),.o(R1resul));
200 register r2 (.in(opR2),.clk(clk),.clr(clr2)
201 ,.enable(enable2),.o(R2resul));

```

```

202 register r3 (.in(opR3),.clk(clk),.clr(clr3)
203 ,.enable(enable3),.o(R3resul));
204 register r4 (.in(opR4),.clk(clk),.clr(clr4)
205 ,.enable(enable4),.o(R4resul));
206 register #(11) r5 (.in(add4resul),.clk(clk),.clr(clr5)
207 ,.enable(enable5),.o(R5resul));
208 register r6 (.in(opR6),.clk(clk),.clr(clr6)
209 ,.enable(enable6),.o(R6resul));
210 register #(12) r7 (.in(opR7),.clk(clk),.clr(clr7)
211 ,.enable(enable7),.o(R7resul));
212 register r8 (.in(opR8),.clk(clk),.clr(clr8)
213 ,.enable(enable8),.o(R8resul));
214 register r9 (.in(opR9),.clk(clk),.clr(clr9)
215 ,.enable(enable9),.o(R9resul));
216 register r10 (.in(opR10),.clk(clk),.clr(clr10)
217 ,.enable(enable10),.o(R10resul));
218 register #(20) r11 (.in(opR11),.clk(clk),.clr(clr11)
219 ,.enable(enable11),.o(R11resul));
220
221 register #(11) x1 (.in(Rx1),.clk(clk),.clr(clrx1)
222 ,.enable(enablex1),.o(x1r));
223 register #(11) x2 (.in(Rx2),.clk(clk),.clr(clrx2)
224 ,.enable(enablex2),.o(x2r));
225 register #(11) x3 (.in(Rx3),.clk(clk),.clr(clrx3)
226 ,.enable(enablex3),.o(x3r));
227 register #(11) y1 (.in(Ry1),.clk(clk),.clr(clry1)
228 ,.enable(enabley1),.o(y1r));
229 register #(11) y2 (.in(Ry2),.clk(clk),.clr(clry2)
230 ,.enable(enabley2),.o(y2r));
231 register #(11) y3 (.in(Ry3),.clk(clk),.clr(clry3)
232 ,.enable(enabley3),.o(y3r));
233 endmodule

```

6.3 Código Extend

```

1 module extend #(parameter WIDTH=11) (input [WIDTH-1:0] a
2 ,output reg [23:0] y);
3
4     always_comb
5
6         y = {0, a};
7

```

```
8 endmodule
```

6.4 Código Reduce

```
1 module reduce(input [23:0] a,  
2 output reg [10:0] y);  
3  
4     always_comb  
5  
6         y =a[10:0];  
7  
8 endmodule
```

6.5 Código Mux

```
1 module mux2(input [22:0] a,b,  
2 input sel,  
3 output reg [22:0] y );  
4     always_comb  
5         begin  
6             case(sel)  
7                 0: y =a;  
8                 1: y =b;  
9                 default: y=23'bx;  
10            endcase  
11        end  
12  
13 endmodule
```

6.6 Código Mul

```
1 module mul(a,b,mul);  
2 input [10:0] a,b;  
3 output [23:0]mul;  
4  
5     assign mul=a*b;  
6  
7 endmodule
```

6.7 Código Adder

```
1 module adder #(parameter WIDTH=23)
```



```

2  (input  [WIDTH-1:0] a,b,
3  input  [1:0]op,
4  output reg [WIDTH:0] o);
5      always_comb
6          begin
7              case (op)
8                  2'b00:o = a+b;
9                  2'b01:o = (a+b)>>2;
10                 2'b10:
11                     if(a>b)
12                         begin
13                             o=a-b;
14                         end
15                     else
16                         o=b-a;
17                 2'b11:o=$unsigned(b-a)>>2;
18                 default:o=24'bx;
19             endcase
20         end
21 endmodule

```

6.8 Código Register

```

1
2  module register #(parameter WIDTH=24) (input  [WIDTH-1:0] in
3  , input clk,clr,enable, output reg [WIDTH-1:0] o);
4
5      always@(posedge clk)
6
7          if(enable)
8              begin
9                  if(clr)
10                     o=0;
11                 else
12                     o=in;
13             end

```

6.9 Código Controller

```

1
2  module controller(clk,reset,valid,valid2,selmul3_1,selmul3_2,seladd5_2,sel
3  clrx1,clrx2,clrx3,clry1,clry2,clry3,enablex1,enablex2,enablex3,enabley1,en

```

```

4  selmul2_2,selmul2_1,selmul1_2,selmul1_1,seladd2_2,seladd1_1,seldiv_2,seldiv_1,
5  seladd4_2,seladd4_1,seladd3_2,seladd3_1,seladd2_1,seladd1_2,
6  opadd1,opadd2,opadd3,opadd4,opadd5,
7  finish
8  );
9
10 //Parametros da maquina de estado
11 input  clk,reset,valid,valid2;
12 reg [5:0] cycle;
13 //Parametros de controle do datapath
14 output reg finish;
15 output reg selmul3_1,selmul3_2,seladd5_2,seladd5_1,
16 selr7,selsqrt,clr1,clr2,clr3,clr4,clr5,clr6,clr7,
17 clr8,clr9,clr10,clr11,enable1,enable2,enable3,
18 enable4,enable5,enable6,enable7,enable8,enable9,
19 enable10,enable11,insqrt,inbottom,intop;
20 output reg clrx1,clrx2,clrx3,clry1,clry2,clry3,enablex1,
21 enablex2,enablex3,enabley1,enabley2,enabley3;
22 output reg [1:0] selmul2_2,selmul2_1,selmul1_2,selmul1_1,
23 seladd2_2,seladd1_1,seldiv_2,seldiv_1,selr11,selr10,
24 selr9,selr8,selr6,selr4,selr3,selr2,selr1;
25 output reg [2:0] seladd4_2,seladd4_1,seladd3_2,seladd3_1,seladd2_1,seladd1_1;
26 output reg [1:0] opadd1,opadd2,opadd3,opadd4,opadd5;
27
28
29 always @(*)
30 begin
31     case(cycle)
32         6'b0:
33             begin
34                 enablex1=1;
35                 enablex2=1;
36                 enablex3=1;
37                 enabley1=1;
38                 enabley2=1;
39                 enabley3=1;
40                 finish=0;
41             end
42         6'b000001:
43             begin
44                 //disabled
45                 enable5=0;

```

```
46         enable6=0;
47         enable8=0;
48         enable9=0;
49         enable10=0;
50         enable11=0;
51         //ax1//
52         //add
53         seladd1_2=3'b100;
54         seladd1_1=2'b11;
55         opadd1=2'b00;
56         //register
57         selr1=2'b10;
58         clr1=0;
59         enable1=1;
60         //ay1//
61         //add
62         seladd2_2=2'b11;
63         seladd2_1=3'b100;
64         opadd2=2'b0;
65         //register
66         selr7=2'b01;
67         clr7=0;
68         enable7=1;
69         //mx1//
70         //mult
71         selmul1_1=2'b10;
72         selmul1_2=2'b10;
73         //register
74         selr2=2'b10;
75         clr2=0;
76         enable2=1;
77         //mx2//
78         //mult
79         selmul2_1=2'b10;
80         selmul2_2=2'b10;
81         //register
82         selr3=2'b10;
83         clr3=0;
84         enable3=1;
85         //mx3//
86         //mult
87         selmul3_1=1'b1;
```

```
88         selmul3_2=1'b1;
89         //register
90         selr4=2'b10;
91         clr4=0;
92         enable4=1;
93     end
94     6'b000010:
95     begin
96         //disabled
97         enable3=0;
98         enable4=0;
99         enable5=0;
100        enable6=0;
101        enable11=0;
102        //Mx//
103        //add
104        seladd1_2=3'b011;
105        seladd1_1=2'b10;
106        opadd1=2'b01;
107        //register
108        clr1=0;
109        enable1=1;
110        selr1=2'b10;
111        //My//
112        //add
113        seladd3_2=3'b101; //ok
114        seladd3_1=3'b100; //ok
115        opadd3=2'b01; //ok
116        //register
117        clr1=0;
118        enable1=1;
119        selr7=0;
120        //amx1//
121        //add
122        seladd2_2=2'b10;
123        seladd2_1=3'b011;
124        opadd2=2'b00;
125        //register
126        clr2=0;
127        enable2=1;
128        selr2=2'b00;
129        //my1//
```

```
130         //mult
131         selmul1_1=2'b01;
132         selmul1_2=2'b01;
133         //register
134         clr8=0;
135         enable8=1;
136         selr8=2'b10;
137         //my2//
138         //mult
139         selmul2_1=2'b01;
140         selmul2_2=2'b01;
141         //register
142         clr9=0;
143         enable9=1;
144         selr9=2'b10;
145         //my3//
146         //mult
147         selmul3_1=0;
148         selmul3_2=0;
149         //register
150         clr10=0;
151         enable10=1;
152         selr10=2'b11;
153         end
154         6'b000011:
155         begin
156             // disabled
157             enable1=0;
158             enable2=0;
159             enable3=0;
160             enable7=0;
161             enable10=0;
162             enable11=0;
163             //Mx2
164             //mult
165             selmul1_2=2'b00;
166             selmul1_1=2'b00;
167             //register
168             enable4=1;
169             selr4=2'b01;
170             //amy1
171             //add
```

```
172         seladd2_2=2'b01;
173         seladd2_1=3'b010;
174         opadd2=0;
175         //register
176         selr8=2'b01;
177         clr8=0;
178         enable8=1;
179         //N1
180         //add
181         seladd4_2=3'b100;
182         seladd4_1=2'b11;
183         opadd4=2'b10;
184         //register
185         clr5=0;
186         enable5=1;
187         //Mx2
188         //add
189         seladd1_2=3'b010;
190         seladd1_1=2'b01;
191         opadd1=2'b01;
192         //register
193         selr6=2'b10;
194         clr6=0;
195         enable6=1;
196         //My2
197         //mult
198         selmul2_2=2'b00;
199         selmul2_1=2'b00;
200         //register
201         selr9=2'b10;
202         clr9=0;
203         enable9=1;
204         end
205     6'b000100:
206         begin
207             enable1=0;
208             enable2=0;
209             enable4=0;
210             enable5=0;
211             enable6=0;
212             enable7=0;
213             enable8=0;
```

```
214         enable9=0;
215         enable11=0;
216         //Vx
217         //add
218         seladd3_2=3'b100;
219         seladd3_1=3'b011;
220         opadd3=2'b10;
221         //register
222         selr3=2'b01;
223         clr3=0;
224         enable3=1;
225         //N2
226         seladd4_2=3'b011;
227         seladd4_1=2'b10;
228         opadd4=10;
229         //register
230         selr4=2'b00;
231         clr4=0;
232         enable4=1;
233         //M2
234         seladd1_2=3'b001;
235         seladd1_1=2'b00;
236         opadd1=2'b01;
237         //register
238         selr10=2'b10;
239         clr10=0;
240         enable10=1;
241         end
242         6'b000101:
243         begin
244             enable1=0;
245             enable2=0;
246             enable4=0;
247             enable5=0;
248             enable7=0;
249             enable8=0;
250             enable10=0;
251             enable11=0;
252             //SDx
253             //sqrt
254             selsqrt=1;
255             insqrt=1;
```

```
256         //register
257         selr6=2'b00;
258         clr6=0;
259         enable6=1;
260         //N3
261         //add
262         seladd4_2=3'b010;
263         seladd4_1=2'b01;
264         opadd4=2'b10;
265         //register
266         selr3=2'b00;
267         clr3=0;
268         enable3=1;
269         //Vy
270         //add
271         seladd3_2=3'b011;
272         seladd3_1=3'b010;
273         opadd3=2'b10;
274         //register
275         selr9=2'b00;
276         clr9=0;
277         enable9=1;
278         end
279     6'b000110:
280         begin
281             enable1=0;
282             enable2=0;
283             enable3=0;
284             enable4=0;
285             enable5=0;
286             enable7=0;
287             enable8=0;
288             enable10=0;
289             enable11=0;
290             //SDy
291             //sqrt
292             selsqrt=0;
293             insqrt=1;
294             //register
295             selr9=2'b01;
296             clr9=0;
297             enable9=1;
```



```
298         end
299         6'b001101:
300         begin
301             enable1=0;
302             enable3=0;
303             enable4=0;
304             enable5=0;
305             enable6=0;
306             enable7=0;
307             enable8=0;
308             enable10=0;
309             enable11=0;
310             //MSDx
311             //add
312             seladd2_2=2'b00;
313             seladd2_1=3'b001;
314             opadd2=2'b10;
315             //register
316             selr2=2'b0;
317             clr2=0;
318             enable2=1;
319         end
320         6'b001110:
321
322         begin
323             enable3=0;
324             enable4=0;
325             enable5=0;
326             enable7=0;
327             enable9=0;
328             enable10=0;
329             enable11=0;
330             //DX1
331             //add
332             seladd1_2=3'b100;
333             seladd1_1=2'b01;
334             opadd1=2'b10;
335             //register
336             selr1=2'b10;
337             clr1=0;
338             enable1=1;
339             //DX2
```

```
340 //add
341 seladd3_2=3'b010;
342 seladd3_1=3'b001;
343 opadd3=2'b10;
344 //register
345 selr2=2'b01;
346 clr2=0;
347 enable2=1;
348 //DX3
349 //add
350 seladd4_2=3'b001;
351 seladd4_1=2'b01;
352 opadd4=2'b10;
353 //register
354 selr6=2'b01;
355 clr6=0;
356 enable6=1;
357 //MSDY
358 //add
359 seladd2_2=2'b01;
360 seladd2_1=3'b000;
361 opadd2=2'b10;
362 //register
363 selr8=2'b01;
364 clr8=0;
365 enable8=1;
366 end
367 6'b001111:
368 begin
369 enable1=0;
370 enable2=0;
371 enable3=0;
372 enable4=0;
373 enable5=0;
374 enable6=0;
375 enable7=0;
376 enable11=0;
377 //DY
378 //add
379 seladd1_2=3'b000;
380 seladd1_1=2'b00;
381 opadd1=2'b10;
```

```
382         //register
383         selr8=2'b00;
384         clr8=0;
385         enable8=1;
386         //DY2
387         //add
388         seladd3_2=3'b001;
389         seladd3_1=3'b000;
390         opadd3=2'b10;
391         //register
392         selr9=2'b00;
393         clr9=0;
394         enable9=1;
395         //DY3
396         //add
397         seladd4_2=3'b010;
398         seladd4_1=2'b00;
399         opadd4=2'b10;
400         //register
401         selr10=2'b01;
402         clr10=0;
403         enable10=1;
404         end
405         6'b010000:
406         begin
407             enable2=0;
408             enable3=0;
409             enable4=0;
410             enable5=0;
411             enable7=0;
412             enable8=0;
413             enable9=0;
414             enable11=0;
415             //D1
416             //add
417             seladd4_2=3'b0;
418             seladd4_1=2'b0;
419             opadd4=2'b0;
420             //register
421             selr1=2'b00;
422             clr1=0;
423             enable1=1;
```

```
424         //D2
425         //add
426         seladd2_2=2'b01;
427         seladd2_1=3'b011;
428         opadd2=2'b0;
429         //register
430         selr6=2'b11;
431         clr6=0;
432         enable6=1;
433         //D3
434         //add
435         seladd3_2=3'b000;
436         seladd3_1=3'b011;
437         opadd3=2'b0;
438         //register
439         selr10=2'b00;
440         clr10=0;
441         enable10=1;
442         end
443         6'b010001:
444         begin
445             enable10=0;
446             enable1=0;
447             //Q1
448             inbottom=1;
449             intop=1;
450             //div
451             seldiv_2=2'b10;
452             seldiv_1=2'b10;
453             //register
454             selr11=2'b10;
455             clr11=0;
456             enable11=1;
457         end
458         6'b010010:
459         begin
460             //Q2
461             inbottom=1;
462             intop=1;
463             //div
464             seldiv_2=2'b01;
465             seldiv_1=2'b01;
```

```
466         end
467     6'b010011:
468         begin
469             //Q3
470             inbottom=1;
471             intop=1;
472             //div
473             seldiv_2=2'b00;
474             seldiv_1=2'b00;
475         end
476     6'b100111:
477         begin
478             //aQ1
479             //add
480             seladd5_2=1;
481             seladd5_1=1;
482             opadd5=2'b00;
483             //register
484             selr11=2'b00;
485             clr11=0;
486             enable11=1;
487         end
488     6'b101000:
489         begin
490             //aQ2
491             seladd5_2=1;
492             seladd5_1=1;
493             opadd5=2'b01;
494             //register
495             selr11=2'b01;
496             clr11=0;
497             enable11=1;
498         end
499     6'b101001:
500         begin
501             //Verc
502             seladd5_2=0;
503             seladd5_1=0;
504             opadd5=2'b10;
505             //register
506             selr11=2'b01;
507             clr11=0;
```

```
508         enable11=1;
509     end
510     6'b101010:
511     begin
512         finish =1;
513         clr1=1;
514         clr2=1;
515         clr3=1;
516         clr4=1;
517         clr5=1;
518         clr6=1;
519         clr7=1;
520         clr8=1;
521         clr9=1;
522         clr10=1;
523         clr11=1;
524     end
525 endcase
526 end
527
528 always@(posedge clk, negedge reset)
529
530     if(~reset)
531         cycle <= 6'b0;
532     else
533         case(cycle)
534             6'b0:
535                 cycle <= 6'b000001;
536             6'b000001:
537                 cycle <= 6'b000010;
538             6'b000010:
539                 cycle <= 6'b000011;
540             6'b000011:
541                 cycle <= 6'b000100;
542             6'b000100:
543                 cycle <= 6'b000101;
544             6'b000101:
545                 cycle <= 6'b000110;
546             6'b000110:
547                 if(valid)
548                     cycle <= 6'b001101;
549                     6'b001101
```

```
550                                                     if(valid)
551                 cycle <= 6'b001110;
552         6'b001110:
553                 cycle <= 6'b001111;
554         6'b001111:
555                 cycle <= 6'b010000;
556         6'b010000:
557                 cycle <= 6'b010001;
558         6'b010001:
559                 cycle <= 6'b010010;
560         6'b010010:
561                 cycle <= 6'b010011;
562         6'b010011:
563                 if(valid2)
564                 cycle <= 6'b100111;
565         6'b100111:
566                 cycle <= 6'b101000;
567         6'b101000:
568                 cycle <= 6'b101001;
569         6'b101001:
570                 cycle <= 6'b101010;
571         6'b101010:
572                 cycle <= 6'b000000;
573         endcase
574 endmodule
```