Escrita Artigo

Otimizando o design utilizando Vivado HLS

Esta seção descreve as várias otimizações e técnicas que você pode usar para direcionar o Vivado HLS para produzir uma microarquitetura que satisfaça os objetivos desejados de desempenho e área.

Diretivas de otimização fornecidas pelo Vivado HLS:

ALLOCATION: Especifica um limite para o número de operações, núcleos ou funções utilizadas. Isso pode forçar o compartilhamento ou hardware recursos e pode **aumentar a latência**

ARRAY_MAP: Combina vários arrays menores em um único array grande para ajudar a reduzir os recursos de RAM do bloco.

ARRAY_PARTITION: Particiona grandes matrizes em múltiplos arrays menores ou em registros individuais, para melhorar o acesso aos dados e remover bloquear gargalos de RAM.

ARRAY_RESHAPE: Remodela um array de um com muitos elementos para um com maior largura de palavra. Útil para melhorar a RAM do bloco acessa sem usar mais RAM de bloco.

CLOCK: Para projetos do SystemC, vários clocks nomeados podem ser especificado usando o comando create_clock e aplicado para SC_MODULEs individuais usando essa diretiva.

DATA_PACK: Empacota os campos de dados de uma estrutura em um único escalar com um largura de palavra mais ampla.

DATAFLOW: Permite o pipeline de nível de tarefa, permitindo funções e loops para executar simultaneamente. Usado para minimizar o intervalo.

DEPENDENCE: Usado para fornecer informações adicionais que podem superar dependências de loop-carry e permitir que os loops sejam canalizados(ou pipeline com intervalos mais baixos).

EXPRESSION_BALANCE: Permite que o balanceamento automático de expressões seja desativado.

FUNCTION_INSTANTIATE: Permite que instâncias diferentes da mesma função sejam localmente otimizado.

INLINE: Inline uma função, removendo toda a hierarquia de funções. Costuma habilitar a otimização lógica através dos limites das funções e **melhora a latência** / intervalo reduzindo a chamada de função a sobrecarga.

INTERFACE: Especifica como as portas RTL são criadas a partir da função descrição.
LATENCY: **Permite que uma restrição de latência mínima e máxima seja especificadas.**

LOOP_FLATTEN: Permite que os loops aninhados sejam recolhidos em um único loop com melhor latência.

LOOP_MERGE: Mesclar loops consecutivos para reduzir a latência geral, aumentar compartilhar e melhorar a otimização lógica.

LOOP_TRIPCOUNT: Usado para loops que possuem limites de variáveis. Fornece um estimativa para a contagem de iteração do loop. Isso não tem impacto sobre síntese, apenas em relatórios.
OCCURRENCE: Usado quando pipelining funções ou loops, para especificar que o código em um local é executado em uma taxa menor do que o código na função ou loop envolvente.

PIPELINE: Reduz o intervalo de inicialização, permitindo que o concorrente execução de operações dentro de um loop ou função.

PROTOCOL: Este comando especifica uma região do código para ser um região de protocolo. Uma região de protocolo pode ser usada manualmente especifique um protocolo de interface.

RESET: Essa diretiva é usada para adicionar ou remover redefinições em um variável de estado (global ou estática).

RESOURCE: Especifique que um recurso de biblioteca específico (core) é usado para implementar uma variável (matriz, operação aritmética ou argumento de função) na RTL.
entrevista com chamusca
STREAM: Especifica que uma matriz específica deve ser implementada como FIFO ou canal de memória

RAM durante o fluxo de dados otimização. Ao usar o hls :: stream, o STREAM diretiva de otimização é usada para substituir a configuração do hls :: stream.

TOP: A função de nível superior para síntese é especificada no configurações do projeto. Esta diretiva pode ser usada para especificar funcionar como o nível superior para síntese. Isso então permite diferentes soluções dentro do mesmo projeto a ser especificado como a função de nível superior para a síntese sem a necessidade de crie um novo projeto.

UNROLL: Unroll for-loops para criar múltiplas operações independentes em vez de uma única coleção de operações.

Comentários:

Fonte: Vivado Design Suite User Guide *High-Level Synthesis*, UG902 (v2017.4) February 2, 2018.

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	5.00	5.27	0.62

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
57	57	43	43	dataflow

Detail

Instance

Loop

Dataflow

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	5.00	4.38	0.62

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
102	102	103	103	none

Inline

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	5.00	4.29	0.62

Latency (clock cycles)

Summary

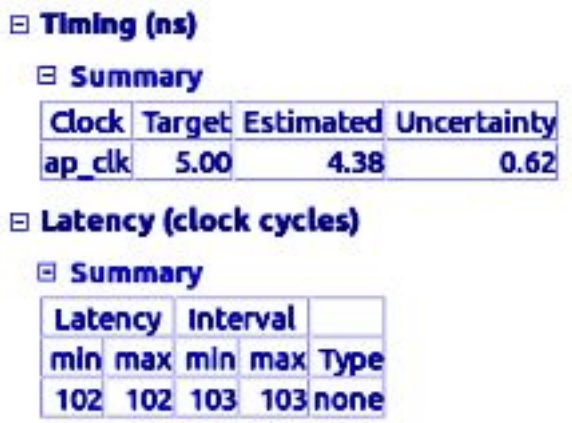
Latency		Interval		
min	max	min	max	Type
84	84	1	1	function

Detail

Instance

Loop

Pipeline



Unroll

Introdução

serviços são disponibilizados por meio da rede mundial de computadores. Serviços tais 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 quaisquer eventos 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 e 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 o serviço (HOQUE; KASHYAP; BHATTACHARYYA, 2017). Ataques do tipo DoS distribuidos 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 ASICs, 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.

Devido a complexidade das aplicações modernas, sistemas de tempo real são cada vez mais necessários no contexto atual. Tais sistemas estão incluídos nas mais variadas áreas de conhecimento: Aviação, medicina, astronomia e etc. Sabendo que esses sistemas requerem velocidade e desempenho, além alto grau de processamento é importante novos métodos e tecnologias para otimização das aplicações de tempo real.
Os métodos de otimização de sistemas de tempo real são baseados na metodologia em que esses sistemas são implementados(algoritmos,métricas e fluxo). Além disso, existem estratégias que buscam realizar otimização em sistemas sem afetar a estrutura do mesmo. Vale ressaltar, que a otimização ideal ocorre para todos os requisitos da aplicação, porém em sistemas reais devem existir requisitos mais prioritários para que haja um maior ganho específico nos fins que a aplicação deseja realizar.
Para realizar tarefas de alto grau de processamento é necessário algumas estratégias podem ser utilizadas como utilização de funções matemáticas consolidadas, métodos estatísticos tradicionais e novos algoritmos. Porém para esse processamento ser em tempo real ele deve ser dinâmico e veloz por isso em pouco tempo ele deve processar um conjunto de dados em um sistema. Por isso um método simples de otimização seria a redução variáveis de entrada para a realização dessa tarefa, assim alocando o máximo de tempo possível para o processamento dessas entradas no sistema.
As tecnologias 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 soluções de sistema de tempo real. 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 poder de processamento são as FPGAs e ASICs, porém as FPGAs oferecem adaptabilidade dinâmica, que é importante para aplicações que requerem mudanças frequentes em suas configurações.