Automatic Generation of VLSI Architectures for SAD Calculation*

1st Bruna Suemi Nagai
INE - UFSC
Florianópolis, Brasil
brunasunagai@gmail.com

***Abstract*—Devido ao grande número de vezes em que a soma das diferenças absolutas é aplicada e à simplicidade de seu cálculo, torna-se interessante acelerar esse processo com implementações VLSI. O presente trabalho propõe a implementação de diferentes arquiteturas em hardware para o cálculo da SAD que é largamente utilizada por codificadores de vídeo. Diferentes níveis de paralelismo foram analisados para a síntese em uma placa FPGA Cyclone V e as descrições em *hardware* foram geradas por um *script* em Python. Foi obtida a frequência máxima de 84.18 MHz para a arquitetura com somador RP e tamanho de bloco de 2x2. Este trabalho faz parte da avaliação final da disciplina INE410141 - *Selected Topics in Computer Science: Video Coding*, ministrada pelos professores José Luis Guntzel e Mateus Grellert.**

***Index Terms*—video coding, SAD, VLSI, Python**

I. Introdução

O crescente uso de redes sociais, serviços de *streaming* e plataformas de comunicação, fez com que grande parte do tréfego global de dados de internet fossem gastos com conteúdos como vídeos e imagens de alta definição, como previsto em [1]. No entanto, pudemos observar que esse fenômeno se agravou mais nos últimos dois anos devido à pandemia do Covid-19 [2]. Torna-se, portanto, ainda mais urgente a necessidade de garantir a alta qualidade dos vídeos transmitidos.

Um dos principais balanços que se precisa fazer é garantir vídeos de alta qualidade ao mesmo tempo em que o nível de compressão seja suficiente para que seja possível transmití-lo rapidamente pela internet. Outro caso importante atualmente, é o uso das câmeras de *smartphones*, que por possuirem menor espaço de armazenamento e menor tempo de duração da bateria, é desejável que os vídeos possam ser alocados em memória sem precisar de alguns Giga Bytes ou que não gastasse toda a energia do aparelho apenas por gravar um vídeo ou até que não demorasse muito tempo.

Para o caso estudado, foram considerados alguns dos princípios básicos como restrições de arquitetura (i.e. tempo e armazenamento). Sob essa ótica, pode-se analisar o processo de codificação de vídeo de modo a acelerá-lo com implementações em *Very Large Scale Integration* (VLSI).

A Soma das Diferenças Absolutas (SAD) surge como um cálculo simples e eficaz para que a etapa de Estimação de Movimento (ME) consiga determinar qual o bloco mais similar dentro de uma janela de busca. No entanto, a quantidade de cálculos de SAD é muito alta, próximo a bilhões de operações para codificar um vídeo em HD [5]. E conforme a resolução aumenta, cresce a quantidade de operações necessárias para codificação.

Portanto, verifica-se que a SAD é objeto de estudo interessante para implementação de aceleradores em VLSI dada à larga quantidade de vezes em que é utilizada e à sua simplicidade de cálculo, que permite ser implementado facilmente.

Para o presente trabalho, foram implementadas arquiteturas para todos os tamanhos de blocos quadrados, bem como a análise de dois diferentes tipos de somadores que compõem a árvore de somas. Também vale ressaltar que a descrição em VHDL foi feita com um gerador automático em Python. Ao final, os resultados foram sintetizados para uma FPGA Cyclone V utilizando o *software* Quartus Prime 20.1.

II. Fundamentação Teórica

A. Estimação de Movimento (ME)

Na maioria dos codificadores de vídeos modernos, os quadros são divididos em blocos menores, quadrados e/ou retângulos como mostra a Figura 1, que passam por algoritmos de predição intra quadros e inter quadros, onde a primeira explora a redundância espacial e a segunda, redundância temporal. Assim, obtém-se maiores taxas de compressão do que se apenas fosse realizado a codificação entrópica, já que se aproveita das características próprias dos vídeos.

Durante a predição inter quadros, há a etapa de Estimação de Movimento (ME), uma das maiores consumidoras de tempo, podendo chegar a até 80% do tempo total de codificação no modo *full-search* [3]. Tal esforço computacional se deve ao fato de que a ME é responsável por procurar o melhor bloco, por critério de semelhança, será utilizado como referência na Estimação de Movimento (MC) posteriormente [4].

A ME se divide em predição do Vetor de Movimento (MV), a ME inteira (IME) e ME fracionária (FME) [8]. A primeira parte utiliza as informações de movimento dos blocos vizinhos para definir a posição inicial da próxima busca. A IME faz a busca em volta da posição apontada pelo MV e a FME fornece um resultado mais refinado da busca realizada pela IME, pois considera pixels fracionários [8]. O foco do trabalho está na IME, onde a SAD é calculada como uma métrica de distorção.

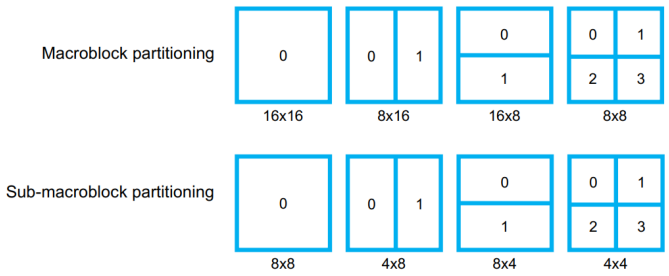

Fig. 1. Exemplos de particionamento de macroblocos e sub-blocos. Fonte: [5]

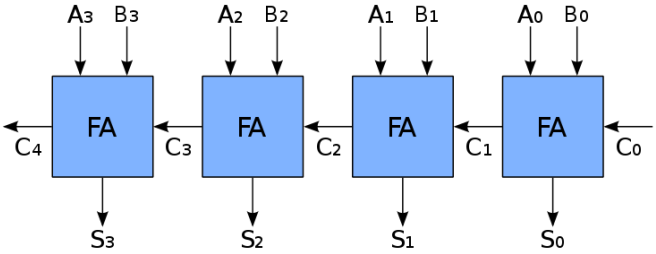

Fig. 2. RCA de quatro bits. Fonte: [7]

B. Soma das Diferenças Absolutas (SAD)

A SAD, como já dito, é um cálculo simples e pode ser representado por (1), onde N é o tamanho do bloco quadrado, O representa o valor do pixel na posição (i, j) no quadro original, enquanto R é o pixel do quadro de referência deslocado de (x, y) em relação a (i, j).

$$\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}|O_{i,j} - R_{i+x,j+y}| \tag{1}$$

Pode-se observar que para este cálculo são necessários apenas três componentes: subtrator, somador e um componente que toma o valor absouto do número. Todos são triviais de implementar em VHDL.

A SAD é a maior consumidora de tempo de computação da IME, pois é necessário que amostras de blocos sejam carregadas da memória para fazer o cálculo [6]. Além disso, todos os blocos da janela de busca de todos os quadro de referência são comparados através da SAD também. Isso é significativo por conta da quantidade de quadros de referência que cada padrão de codificador de vídeo utiliza. O padrão AV1, como exemplo, pode precisar de até 7 quadros de referência, sendo elas anteriores ou posteriores ao quadro original.

C. Somador Ripple Carry (RCA)

O RCA é um somador lento, com relativamente pouco gasto de área. Considera-se utilizar esse somador em casos onde não é requerida altas velocidades para as operações aritméticas. Este pode ser considerado um somador tradicional, já que é implementado em VHDL e Verilog apenas com o uso do símbolo "+". Além disso, o RCA é composto por *full adders* (FA) e *half adders* (HA), que são estruturas básicas de soma bit a bit.

A desvantagem desse somador é que o bit de *carry* deve propagado para cada um dos bits a serem somados, ou seja, se inicia a soma pelos bits menos significativos e conforme se segue para a soma dos bits mais significativos, deve-se esperar o *carry* anterior ser calculado.

A Figura 2 representa um somador RCA de quatro bits, onde o sinal C representa o *carry* que é propagado através de todos os FA's.

D. Somador Carry Lookahead (CLA)

O CLA é um somador mais rápido do que o RCA, pois não depende do resultado do *carry* calculado na posição anterior para avançar. Considerando um CLA de um bit, temos as equações lógicas (2), (3), (4) e (5), sendo A e B os bits de entrada que serão somados, $generate$ (G), $propagate$ (P) e $carry$ (C) são os sinais intermediários, S é o resultado da soma utilizando FA e, por fim, R é o resultado final do CLA de um bit, resultando em dois bits em (6).

- $Generate$:

$$G(0) = A\ and\ B \tag{2}$$

- $Propagate$:

$$P(0) = A\ or\ B \tag{3}$$

- $Carry$:

$$C(1) = G(0)\ or\ (P(0)\ and\ C(0)) \tag{4}$$

- Soma:

$$S(0) = A + B + C(0) \tag{5}$$

- Resultado:

$$R = C(1)\ \&\ S(0) \tag{6}$$

Portanto, uma possibilidade é utilizar N CLA's de um bit para gerar um somador de N bits, apesar de não ser o ideal.

III. Implementaçao proposta

A. Gerador de VHDL

Foi desenvolvido um *script* em Python que gera automaticamente os arquivos necessários para a arquitetura completa da SAD, ou seja, está configurado para a escrita do *toplevel*, do pacote com os tipos de entrada (*arrays* de *std_logic_vector*) e do *testbench* correspondente.

Os parâmetros disponíveis para alteração são:

- *block_size*: define o tamanho total do bloco que será processado (4, 16, 64);

- *bit_width*: define a largura de bits para os valores de *pixels* de entrada;
- *files*: lista com os nomes dos arquivos extras a serem adicionados no *toplevel*;
- *adder_name*: define o nome do somador utilizado;
- *pack_name*: define o nome do pacote com tipos de entrada;
- *write_vhd*: se *True*, irá reescrever o arquivo .vhd do *toplevel*;
- *write_pack*: se *True*, irá reescrever o arquivo .vhd do pacote.

B. Arquitetura da SAD

A implementação proposta partiu da ideia de processamento totalmente paralelo e combinacional de todos os *pixels* de um bloco de entrada. Logo, se considerarmos um bloco de 2x2, na verdade estaremos realizando o cálculo da SAD de todos os 4 *pixels* de uma vez, como mostra o exemplo na Figura 3.

Para efetuar o cálculo da SAD em si, foram utilizados subtratores e somadores. O esquemático da Figura 3 pode ser visto como uma estrutura básica, pois para gerar as outras variações de arquitetura com diferentes tamanhos de blocos, deve-se multiplicar a quantidade de tal estrutura e somar as parcelas em novos níveis de soma abaixo. O gerador de VHDL foi desenvolvido a partir dessa lógica, onde a quantidade de níveis de soma abaixo de *ABS* é igual a $log_2\ (block\_size)$. E cada nível possui $block\_width/2^{level}$ somadores.

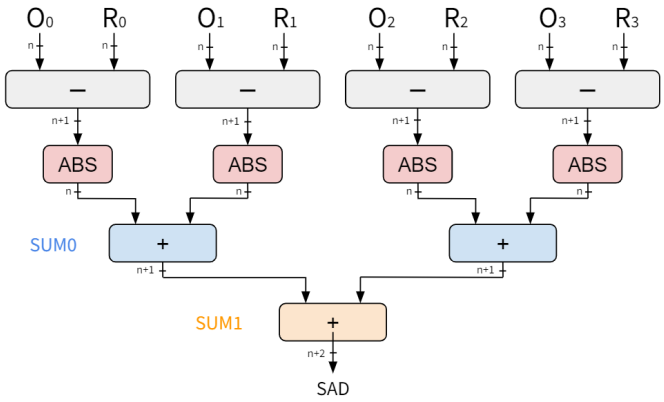

Fig. 3. Esquema de uma arquitetura proposta para a SAD considerando o processamento paralelo de um bloco inteiro 2x2.

Para o maior aproveitamento dos somadores e para facilitar a generalização, o subtrator é o mesmo que o somador, porém as entradas referentes aos *pixels* de entrada do bloco de referência são negativados, restando apenas uma soma entre um número positivo e outro negativo.

C. Somadores

Existem algumas implementações possíveis para o CLA, no entanto como um dos objetivos do presente trabalho era conseguir a automação da escrita do arquivo .vhd, optou-se por utilizar um somador de N bits genérico, onde pode-se definir a quantidade de bits necessária para cada soma no próprio *script* gerador. Assim, se a entrada for de oito bits, o CLA de um bit será replicado oito vezes pelo uso da função *generate*.

No caso do *adder*, foi criado um componente apenas tendo o sinal de saída recebendo a soma com "+", em que a quantidade de bits sendo somada é definida pelo *script*. Isso foi feito dessa maneira para seguir o padrão de componentes de somadores que podem ser alterados conforme a necessidade.

IV. Resultados

Todas as seis arquiteturas foram sintetizadas utilizando o *software* Quartus Prime 20.1 para uma FPGA Cyclone V, modelo 5CGXFC7C7F23C8.

Os resultados estão apresentados na tabela da Figura 4, onde o tipo RP se refere a arquitetura com somadores *Ripple Carry* e o tipo CLA, *Carry Lookahead*. Além disso, temos os resultados da quantidade de unidades lógicas da FPGA utilizadas e a quantidade de pinos I/O. Como a quantidade de pinos ultrapassava o limite de 268 para a FPGA escolhida, todas as sínteses foram feitas tornando os pinos como virtuais para que isso não afetasse a comparação de atraso entre as arquiteturas (mesmo que para 2x2 a quantidade de pinos não ultrapasse o limite).

Tipo	Tamanho do bloco	ALM	Pinos	Máx atraso
RP	2x2	65	75	11.879 ns
	4x4	406	269	19.152 ns
	8x8	1100	1039	29.621 ns
CLA	2x2	135	75	17.150 ns
	4x4	565	269	23.987 ns
	8x8	1716	1039	28.665 ns

Fig. 4. Tabela com os resultados da síntese para FPGA Cyclone V.

O atraso foi obtido pela ferramenta *Time Analizer* do Quartus. No entanto, como não foi definido um sinal de *clock*, não havia como analizar o período verdadeiro ou o *slack*, apenas foi possível verificar o caminho crítico entre os sinais de entrada e saída.

Em termos de frequência, temos a maior para o tipo de somador RP e bloco de tamanho 2x2, que é 84.18 MHz e a menor, para o RP também, mas com bloco 8x8, que é 33.76 MHz.

V. Conclusões

Um ponto de destaque nos resultados foram os valores de atraso máximo medidos, já que teoricamente deveríamos obter menor atraso para as arquiteturas utilizando o CLA. Porém o observado foi o contrário para os tamanhos de bloco igual a 4 e 16 e a diferença de tempo para o caso de bloco de 64

é pouca. A partir disso, podemos considerar realmente que a tentativa de generalização do CLA ao replicá-lo gerou um efeito incorreto.

No geral, os resultados foram satisfatórios por ser uma proposta inicial. No entanto, o ideial seria implementar uma lógica sequencial, com uso de *pipelines* entre os níveis de somadores. Além disso, poderia ser considerado o cálculo da SAD não para o bloco todo, mas para linhas ou parte do total de *pixels* do bloco, fazendo a reutilização do *hardware*.

Em relação aos somadores, a generalização do CLA pode ser evitada e outras topologias de somadores, como os compressores pode ser testada. Além disso, a falta de uma síntese lógica que não fosse para a FPGA, impossibilitou a análise correta da máxima frequência obtida, bem como dos resultados de potência e área ocupada.

REFERENCES

[1] Cisco, U. "Cisco annual internet report (2018–2023) white paper." Cisco: San Jose, CA, USA (2020).

[2] Amy Watson, "Consuming media at home due to the coronavirus worldwide", https://www.statista.com/statistics/1106498/homemedia-consumption-coronavirusworldwide-by-country/, 2020.

[3] Zhang, Yongfei, Chao Zhang, and Rui Fan. "Fast motion estimation in HEVC inter coding: An overview of recent advances." 2018 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC). IEEE, 2018.

[4] Seidel, Ismael, et al. "Analysis of Pel Decimation and Technology Choices to Reduce Energy on SAD Calculation." Journal of Integrated Circuits and Systems 9.1 (2014): 48-59.

[5] Guntzel, José and Grellert, Mateus. "Lecture 6 - Inter-frame prediction – Part 2", 2021.

[6] Sant'Anna, Gabriel B., et al. "Relying on a rate constraint to reduce motion estimation complexity." ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, 2021.

[7] Wikimedia Commons, "File:4-bit ripple carry adder-2.svg", https://commons.wikimedia.org/wiki/File:4-bit_ripple_carry_adder-2.svg, 2018.

[8] Y. Zhang, C. Zhang, and R. Fan, "Fast Motion Estimation in HEVC Inter Coding: An Overview of Recent Advances," in 2018 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC), Nov 2018, pp. 49–56.