UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

Marcio Monteiro

ARQUITETURA ENERGETICAMENTE EFICIENTE PARA CÁLCULO DA SATD ATRAVÉS DO REÚSO DE DADOS

Florianópolis

2017

Marcio Monteiro

ARQUITETURA ENERGETICAMENTE EFICIENTE PARA CÁLCULO DA SATD ATRAVÉS DO REÚSO DE DADOS

Trabalho de Conclusão de Curso submetido ao Curso de Bacharelado em Ciências da Computação para a obtenção do Grau de Bacharel em Ciências da Computação.
Orientador: Prof. Dr. José Luís Almada Güntzel
Universidade Federal de Santa Catarina
Coorientador: M.e Ismael Seidel
Universidade Federal de Santa Catarina

Florianópolis

2017

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Monteiro, Marcio
Arquitetura energeticamente eficiente para cálculo da SATD através do reúso de dados / Marcio Monteiro ; orientador, José Luís Almada Güntzel, coorientador, Ismael Seidel, 2017.
148 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Centro Tecnológico, Graduação em Ciências da Computação, Florianópolis, 2017.

Inclui referências.

1. Ciências da Computação. 2. Codificação de Vídeo. 3. Estimação de movimento. 4. SATD. 5. Reúso de cálculos. I. Güntzel, José Luís Almada. II. Seidel, Ismael. III. Universidade Federal de Santa Catarina. Graduação em Ciências da Computação. IV. Título.

Marcio Monteiro

ARQUITETURA ENERGETICAMENTE EFICIENTE PARA CÁLCULO DA SATD ATRAVÉS DO REÚSO DE DADOS

Florianópolis, 08 de Julho 2017.

Prof. Dr. Renato Cislaghi
Universidade Federal de Santa Catarina
Coordenador

Banca Examinadora:

Prof. Dr. José Luís Almada Güntzel
Universidade Federal de Santa Catarina
Orientador

M.e Ismael Seidel
Universidade Federal de Santa Catarina
Coorientador

Prof. Dr. Laércio Lima Pilla
Universidade Federal de Santa Catarina

AGRADECIMENTOS

Gostaria de agradecer aos meus pais, Gilvite e Juvêncio, pelo apoio dado ao longo da minha vida. Aos meus irmãos, Jucemar e Jocelaine, por todos estes anos de convivência.

Ao Seu Willy Sommer e à dona Marisa Sommer por todas as conversas, dicas e apoio ao longo dos anos.

A todos os meus amigos, em especial ao Gustavo Quadros, João Antônio, Murillo Cunha e Rafael Rodrigues pelos longos anos de amizade e diversas aventuras vividas desde o início do ensino médio.

Ao José L. A. Güntzel por orientar este trabalho e ao Ismael Seidel pela coorientação, sem vocês este trabalho não teria sido feito. Ao Laércio L. Pilla por aceitar participar da banca avaliadora e desta forma contribuir com o desenvolvimento deste trabalho.

A todos os integrantes do *Embedded Computer Lab.* (ECL) pelos diversos momentos de zueira ao longo destes anos que faço parte da equipe deste laboratório. Ao Sheiny Almeida pelas suas piadas brilhantemente ruins e ao mesmo tempo geniais. Agradecimento especial ao pessoal de *video coding* André B. Bräscher, Luiz L. Cancellier e ao Ismael Seidel por todas as dicas ao longo do desenvolvimento deste trabalho e também as revisões do texto.

A todos os professores com os quais tive aula nestes muitos anos de vida acadêmica, certamente todos contribuíram para que eu pudesse chegar até aqui.

Ao CNPq/Capes pelas diversas bolsas que possibilitaram o desenvolvimento deste e outros trabalhos.

A todas as bandas que fizeram a trilha sonora das diversas horas que trabalhei no desenvolvimento deste trabalho.

E por último, mas não menos importante, o contribuinte brasileiro pelo financiamento dos meus estudos em uma instituição pública através do pagamento dos impostos.

Se você faz o que todo mundo faz, chega aonde todos chegam. Se você quer chegar aonde a maioria não chega, precisa fazer algo que a maioria não faz.

Roberto Shinyashiki

RESUMO

O contínuo aumento das resoluções usadas em vídeos digitais tornam necessária a adoção de novas técnicas de codificação de vídeo. A Estimação de Movimento (ME) é a etapa mais intensiva em termos de tempo e consumo energético por realizar um elevado número de cálculos de similaridade entre blocos, como por exemplo a SATD. Assim, este trabalho propõe uma arquitetura de SATD com reúso de cálculos, tendo como objetivo diminuir o consumo energético. Após a descrição da arquitetura, a mesma foi sintetizada e simulada com uma ferramenta de uso industrial. Foram utilizados cinco conjuntos de dados para simulação, um gerado a partir de dados aleatórios e quatro a partir de sequências de vídeos. Ao analisar os resultados obtidos, houve uma redução na área de até 80% em relação às arquiteturas do estado da arte. O consumo energético da arquitetura projetada foi até 55% menor do que aqueles apresentados pelas arquiteturas do estado da arte. Portanto, a arquitetura proposta se mostra vantajosa quando é necessário calcular múltiplos tamanhos de blocos.

Palavras-chave: Codificação de Vídeo, Estimação de Movimento, SATD, Reúso de Cálculos, Projeto VLSI

ABSTRACT

The increasing video resolutions bring the need for new video coding techniques. Among several tools, Motion Estimation (ME) is one of the most time and energy demanding due to the large number of distortion computations, such as the SATD. Thus, this work proposes a new SATD architecture aiming to reduce energy consumption through the reuse of calculations. Such architecture was synthesized and simulated with Synopsys tools and five distinct data sets were used as stimuli for simulation; one randomly generated and the remaining four obtained from video samples. The results show that the architecture occupies a smaller area than other SATD architectures from the literature. Moreover, the proposed architecture was up to 55% more energy efficient than its counterparts. Therefore, the proposed architecture shows itself as the right design choice when doing variable block size ME using SATD as distortion metric.

Keywords: Video Coding, Motion Estimation, SATD, reuse of calculations, VLSI project

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

SUMÁRIO

1 INTRODUÇÃO

Um vídeo é uma sequência de imagens, chamadas de quadros, apresentadas rapidamente no tempo. Cada quadro é uma matriz de píxeis, cujo tamanho é chamado de resolução. Consequentemente, quanto maior a resolução, maior o número de píxeis. Assim, torna-se necessária a compressão de vídeos. Este é o papel de um codificador de vídeo. Em codificadores baseados em blocos os quadros são divididos matrizes menores, referenciadas por blocos. O particionamento do quadro contribui para o processamento pois facilita a busca e redução das redundâncias. Redundância, no contexto da codificação de vídeo, é a repetição de informação. Esta pode ser espacial em que repetição ocorre em um mesmo quadro ou temporal em que a repetição acontece entre quadros sucessivos (GHANBARI, 2003).

Um codificador de vídeo pode ser dividido em diversas etapas (SULLIVAN et al., 2012). Na Figura 1 é apresentado um fluxograma com algumas destas etapas. O quadro que está sendo codificado, denominado de quadro original, é particionado em blocos menores chamados de blocos originais. A etapa de predição busca entre vários blocos candidatos ($\mathbf{B}^{\text{can}}$), o mais semelhante ao bloco original ($\mathbf{B}^{\text{ori}}$), o candidato escolhido é chamado de bloco de referência ($\mathbf{B}^{\text{ref}}$).

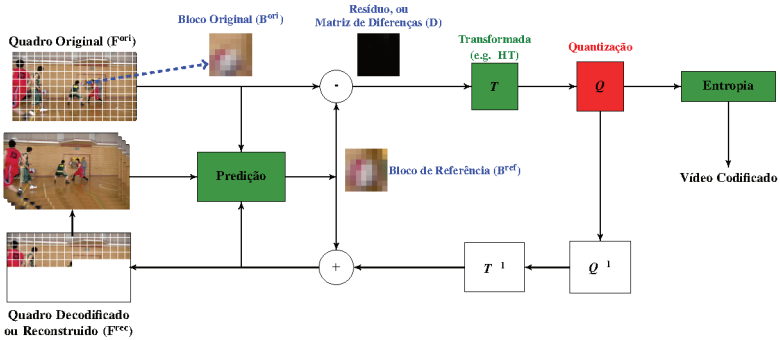


Figura 1: Fluxograma simplificado de um codificador. **T** e **Q** são as etapas de transformação e quantização, respectivamente. $\mathbf{T}^{-1}$ e $\mathbf{Q}^{-1}$ representam as etapas de transformação inversa e quantização inversa. O símbolo de '+' representa a reconstrução de um bloco para ser utilizado como parte de um quadro candidato. Adaptado de Seidel (2016).

A diferença (**D**) entre os blocos original e referência é chamada de resíduo. Na etapa de transformação os resíduos são transformados (T) utili-

zando uma matriz de transformação. Em geral, usa-se a Transformada Discreta dos Cossenos - *Discrete Cosine Transform* (DCT) (AHMED; NATARAJAN; RAO, 1974). Por sua vez, a quantização (Q) divide os valores do resíduo por valores derivados do Parâmetro de Quantização - *Quantization Parameter* (QP). A quantização pode provocar a perda de informação utilizada para representar o bloco, reduzindo assim a qualidade do vídeo. Quanto maior o QP, maior a perda e maior a compressão. Assim, o QP serve como um meio de controlar a taxa de compressão.

Após a quantização, os dados são transmitidos para a codificação de entropia e para a decodificação para alimentar a etapa de predição. A codificação de entropia consiste em reduzir as redundâncias através da análise de frequência dos símbolos e isto não adiciona perdas. Já no processo de decodificação, usado para alimentar a etapa de predição, um bloco codificado é reconstruído pela quantização inversa (Q^{-1}) e pela transformação inversa (T^{-1}). Desta forma, um novo quadro candidato é adicionado para ser utilizado na codificação dos próximos blocos (RICHARDSON, 2002).

Uma das etapas da predição é a Estimação de Movimento - *Motion Estimation* (ME). Tal etapa tem como objetivo reduzir as redundâncias temporais. É também uma das etapas mais intensivas do ponto de vista computacional (BOSSEN et al., 2012), sendo responsável por entre 58% e 86% (LI et al., 2016) do tempo total de codificação no padrão de codificação Codificação de Vídeo de Alta Eficiência - *High Efficiency Video Coding* (HEVC). Com a adoção de resoluções cada vez maiores, como *Full HD* (1920×1080 píxeis) e *Quad HD* (3840×2160 píxeis), a ME requer cada vez mais tempo para realizar os cálculos.

Do tempo total para execução da ME a maior parte é ocupado pelo cálculo da similaridade entre blocos (SILVEIRA et al., 2015; SOARES et al., 2016). Duas das principais métricas de similaridade utilizadas na ME são a Soma das Diferenças Absolutas - *Sum of Absolute Differences* (SAD) e a Soma das Diferenças Transformadas Absolutas - *Sum of Absolute Transformed Differences* (SATD) (RICHARDSON, 2003). Tais métricas de similaridade são compostas por operações aritméticas com baixa complexidade (somas, subtrações e absolutos).

A qualidade da codificação da SATD é melhor quando comparada com a SAD, porém a SATD apresenta um maior custo computacional (MANOEL, 2007) por requerer o cálculo de uma transformada. Além disso, a codificação em tempo real necessita de grande desempenho e ao mesmo tempo ser energeticamente eficiente (CHAKRABARTI; BATTA; CHATTERJEE, 2015). A fim de cumprir com tais requisitos, são necessárias arquiteturas de *hardware* energeticamente eficientes (VANNE et al., 2012).

Algumas arquiteturas de Integração em Larga Escala - *Very-Large*

Scale Integration (VLSI) para o cálculo da SAD exploram o reúso de cálculos para diminuir o tempo gasto e melhorar a eficiência energética da métrica. Este reúso é feito através da soma das SADs dos blocos menores para obter a SAD do bloco maior (KIM; PARK, 2009). Para a SATD, existem implementações que fazem reúso de *hardware*, i.e., o mesmo bloco de *hardware* é utilizado para os cálculos da SATD do bloco menor e do bloco maior (SILVEIRA, 2016). Porém, não foram encontrados trabalhos que fizessem a aplicação do reúso de cálculos na SATD. Assim, neste trabalho é proposto um método para reúso de cálculos e uma arquitetura para cálculo das SATDs de blocos 4×4 e 8×8 com reúso de cálculos. Os tamanhos de blocos suportados foram restringidos aos supracitados para manter a compatibilidade com o Modelo de Testes do HEVC - *HEVC Test Model* (HM).

1.1 OBJETIVOS

O objetivo geral deste trabalho é demonstrar que o reúso de cálculos pode ser aplicado à SATD para reduzir o consumo energético. Para isso, uma arquitetura de *hardware* com baixo consumo de energia para cálculos da SATD com reúso de cálculo foi projetada e sintetizada.

1.1.1 Objetivos Específicos

- Propor e demonstrar como o reúso de cálculos pode ser feito na SATD.
- Projetar e descrever uma arquitetura de SATD com reúso de dados.
- Sintetizar a arquitetura utilizando uma biblioteca de células padrão.
- Simular as arquiteturas de *hardware* projetadas utilizando vetores de testes realistas.
- Obter estimativas precisas de potência.
- Avaliar os resultados de potência e energia obtidos após a simulação da arquitetura sintetizada.
- Comparar os resultados obtidos com aqueles de outras implementações de SATD.

1.2 MÉTODO DE PESQUISA

Com base nos trabalhos correlatos, foi proposto um método para reúso de cálculos na SATD e uma arquitetura. Ambas são detalhadas no Capítulo 4. Os detalhes do método usado para descrição, síntese e simulação da arquitetura proposta são apresentados no Capítulo 5.

1.3 ORGANIZAÇÃO DO TRABALHO

O restante deste trabalho está organizado como segue. No Capítulo 2 são apresentados os conceitos básicos de codificação de vídeo utilizados. A revisão bibliográfica é apresentada no Capítulo 3. No Capítulo 4 é apresentada a arquitetura proposta neste trabalho. O método utilizado para descrição, síntese e simulação da arquitetura é descrito no Capítulo 5. No Capítulo 6 são relatados os resultados obtidos após a síntese e simulação. Finalmente, no Capítulo 7 são apresentadas as conclusões e os trabalhos futuros.

2 CONCEITOS BÁSICOS

Neste capítulo são apresentados os conceitos básicos sobre compressão de vídeo, necessários para compreender este trabalho. Na primeira parte são apresentadas a divisão de blocos e a ME. A seguir são apresentadas duas métricas de similaridade relacionadas a este trabalho e que são utilizadas em codificação de vídeo.

2.1 DIVISÃO DE BLOCOS

No H.264, inicialmente os quadros são particionados em Macroblocos de 16×16 píxeis. Tais MBs podem ser subdivididos em blocos 8×4, 4×8, 8×8. Estes, por sua vez, podem se divididos em blocos menores sendo o menor tamanho permitido 4×4. A Figura 2 apresenta os tamanhos de blocos possíveis no padrão H.264 ao particionar um MB de 16×16.

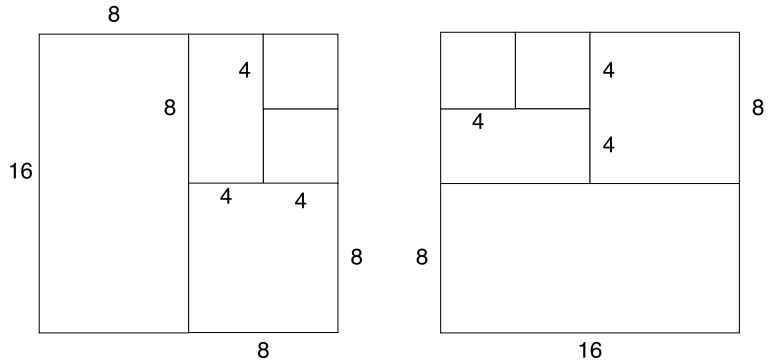


Figura 2: Estes tamanhos de bloco originam os 41 possíveis particionamentos de um MB no H.264/AVC. Adaptado de: ITU-T (2009).

Já no padrão de HEVC, o conceito de MB é ampliado para uma estrutura de *quadtrees*, e a primeira divisão do quadro é em Blocos de Codificação em Árvore - *Coding Tree Block*s (CTBs). Tal estrutura pode ter tamanho máximo de 64×64. Cada CTB pode ser dividida usando uma *quadtree* (SULLIVAN et al., 2012) em Blocos de Codificação - *Coding Blocks* (CBs) de tamanho M$\times$M. N é limitado pelos parâmetros chamados de Maior Bloco de Codificação - *Largest Coding Block* (LCB) e Menor Bloco de Codificação - *Smallest Coding Block* (SCB), podendo ser igual ao tamanho do CTB. Na

Figura 3 é apresentado o particionamento de blocos no HEVC.

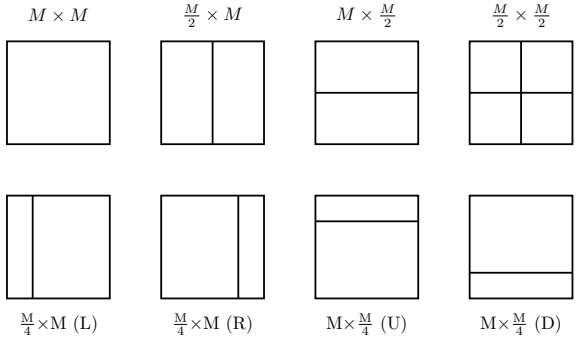


Figura 3: Formas possíveis de particionar uma CB no HM. M é a dimensão da CB, L indica o particionamento vertical assimétrico do lado esquerdo e R no lado direito. U indica o particionamento horizontal na parte superior e D é na parte inferior. Adaptado de: Sullivan et al. (2012).

2.2 ESTIMAÇÃO DE MOVIMENTO

Essa é a etapa responsável por escolher um bloco de referência, dentre um conjunto de candidatos, que tenha a maior similaridade com o bloco que está sendo codificado, chamado de bloco original. Para determinar o quão similar são os blocos é utilizada uma métrica de similaridade.

A ME faz a comparação de uma área do quadro candidato em que estão localizados os blocos candidatos, chamada de Área de Busca - *Search Window* (SW), com o bloco original utilizando uma métrica de similaridade. Na Figura 4 é apresentado o funcionamento da ME com o Algoritmo de Busca Completa - *Fullsearch Block Matching Algorithm* (FBMA) que compara píxel a píxel todos os blocos candidatos da SW ao original. O conjunto *S* é formado por blocos candidatos gerados a partir do percorrimento da SW do quadro candidato e que serão comparados com o bloco original. Após comparar todos os candidatos é obtido o Vetor de Movimento - *Motion Vector* (MV), representado na Figura 4 pela seta que sai do quadro original e vai ao quadro candidato.

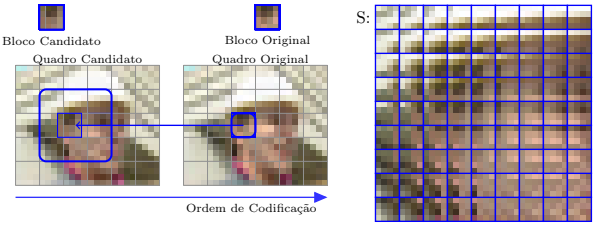


Figura 4: Exemplo de funcionamento da ME utilizando FBMA. O conjunto S representa todos os blocos candidatos considerados durante a ME. Fonte: Seidel (2016).

2.3 MÉTRICAS DE SIMILARIDADE

As métricas de similaridade são utilizadas para comparar o quanto dois blocos são similares entre si. Em geral, as métricas de similaridade são computadas a partir da diferença entre o bloco original ($\mathbf{B}^{\text{ori}}$) e um candidato ($\mathbf{B}^{\text{can}}$), conforme a Equação 2.1. M e N são as dimensões do bloco na forma 2^k para k$\in \mathbb{N}^*$.

$$\mathbf{D}_{M\times N} = \mathbf{B}^{\text{ori}}_{M\times N} - \mathbf{B}^{\text{can}}_{M\times N} \tag{2.1}$$

A Soma das Diferenças Absolutas - *Sum of Absolute Differences* (SAD) é utilizada por sua simplicidade. A métrica consiste em somar todas as diferenças absolutas entre os píxeis dos blocos candidato e original. Na Equação 2.2 é definido o cálculo da SAD, onde $d_{i,j}$ é o elemento na posição i, j de $\mathbf{D}$ (Equação 2.1).

$$\text{SAD}_{M\times N} = \sum_{i=1}^{M}\sum_{j=1}^{N} \mid d_{i,j} \mid \tag{2.2}$$

Uma abordagem utilizada para diminuir o tempo necessário para executar a SAD é o reúso de cálculos. O reúso é feito através do cálculo da SAD dos blocos menores seguida da soma destas para obter a SAD do bloco maior (KIM; PARK, 2009). O reúso com blocos não quadrados é feito através da soma dos blocos menores que compõem o maior. Por exemplo, para calcular a SAD de um bloco 8×4 que foi dividido em dois blocos 4×4, basta calcular a SAD dos blocos menores e somá-los.

A Soma das Diferenças Transformadas Absolutas - *Sum of Absolute Transformed Differences* (SATD) utiliza o mesmo princípio de cálculo da SAD, mas faz uma transformação da matriz de diferenças com a Transformada de Hadamard - *Hadamard Transform* (HT) antes de efetuar a soma ab-

soluta. A SATD obtém melhor qualidade da codificação pois os seus valores calculados tem uma correlação maior com os resíduos transformados.

A SATD é definida como a multiplicação de uma constante de dimensionamento positiva e não nula pelo somatório dos valores absolutos de uma matriz de Diferenças Transformadas (**TD**), como pode ser visto na Equação 2.3.

$$\mathrm{SATD}_{N\times N} = \frac{1}{log_2 N} \times \sum_{i=1}^{N}\sum_{j=1}^{N} \mid td_{i,j} \mid \tag{2.3}$$

onde N é da forma 2^k, com k$\in \mathbb{N}^*$ e $td_{i,j}$ é o elemento na posição i, j de **TD**. A Equação 2.4 apresenta como calcular a **TD**, que consiste em multiplicar a matriz de transformação (**T**) de uma transformada linear inteira pela matriz de diferenças (**D**) e pela **T** transposta.

$$\mathbf{TD}_{N\times N} = \mathbf{T}_{N\times N} \times \mathbf{D}_{N\times N} \times \mathbf{T}^T_{N\times N} \tag{2.4}$$

Uma vez que a matriz de transformação é quadrada, pode-se perceber que a SATD não pode ser aplicada diretamente em blocos não quadrados. Para aplicá-la, inicialmente o bloco não quadrado é dividido em blocos quadrados menores. A SATD é então calculada para esses blocos menores e os resultados são somados para obter a SATD do bloco não quadrado.

Se for utilizada a multiplicação de matrizes usual na Equação 2.4 são necessárias N^3 multiplicações e N^2 somas. Para calcular a **TD** são necessárias duas multiplicações de matrizes, o que implica em dobrar o número de operações. Desta forma são necessárias $2N^3$ multiplicações e $2N^2$ somas. Para calcular a **TD** de matrizes de tamanho 8×8, são necessárias 1024 multiplicações e 128 somas. Com isto nota-se que a complexidade da transformada está na multiplicação das matrizes.

Ao substituir **T** pela matriz de Hadamard (**H**) na Equação 2.4 é obtida a HT. A **H** de dimensões 2^n, com $n \in \mathbb{N}^*$, pode ser construída como segue (AGAIAN et al., 2011):

$$\mathbf{H}_{2^n\times 2^n} = \begin{cases} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} & \text{, se } n = 1 \\ \begin{bmatrix} \mathbf{H}_{2^{n-1}\times 2^{n-1}} & \mathbf{H}_{2^{n-1}\times 2^{n-1}} \\ \mathbf{H}_{2^{n-1}\times 2^{n-1}} & -\mathbf{H}_{2^{n-1}\times 2^{n-1}} \end{bmatrix} & \text{, caso contrário} \end{cases} \tag{2.5}$$

Para diminuir o número de operações, podem ser exploradas duas propriedades da **H**. Uma é o fato dela ser composta apenas por valores 1 e -1 o que permite a multiplicação de matrizes usando apenas somas e subtrações. A outra é a recursividade, que permite o uso de estruturas em *butterfly* para o cálculo da transformada rápida, nesse caso, a Transformada Rápida de Hada-

mard - *Fast Hadamard Transform* (FHT). Também é possível separar as duas multiplicações de matrizes, a qual é chamada de separabilidade (PORTO et al., 2005), contudo não é uma propriedade da **H**.

Ao separar as duas multiplicações de matrizes é possível utilizar um vetor construído a partir de uma coluna de **D** para calcular a **TD**. Anshi, Di e Renzhong (1993) apresentam uma formulação com um vetor genérico (Equação 2.6) que multiplica uma $\mathbf{H}_{4\times4}$ (Equação 2.5 com $n = 2$) e o resultado obtido é apresentado na Equação 2.7. Nota-se que a única diferença entre as equações I e III e as equações II e IV é um sinal. Com isto, é possível subdividir as equações em operações de duas parcelas em que inicialmente são calculadas as somas/subtrações de f_0 com f_1 e f_2 com f_3, para em seguida somar/subtrair os resultados obtidos anteriormente. Pode-se notar que é possível fazer apenas uma vez os cálculos e reutilizá-los para multiplicar o vetor pela **H**.

$$\mathbf{X}_{1\times4} = \begin{bmatrix} f_0 & f_1 & f_2 & f_3 \end{bmatrix}^T \tag{2.6}$$

$$\begin{aligned} &(f_0 + f_1) + (f_2 + f_3)[I] \\ &(f_0 - f_1) + (f_2 - f_3)[II] \\ &(f_0 + f_1) - (f_2 + f_3)[III] \\ &(f_0 - f_1) - (f_2 - f_3)[IV] \end{aligned} \tag{2.7}$$

Ao utilizar *butterflies* para calcular a **TD** são necessárias $2N^2 log_2 N$ operações. Para matrizes de tamanho 8×8, são necessárias apenas 384 operações aritméticas para calcular a **TD**. Nota-se que o uso da *butterfly* para fazer a transformação das matrizes pode reduzir o tempo necessário para calcular as mesmas.

Tomando como base a Figura 5, na qual é apresentado o esquemático de uma *butterfly* de quatro entradas, pode-se definir seu funcionamento como segue. Na primeira camada, para cada par de valores das entradas são feitas uma soma e uma subtração. Para as camadas subsequentes também são feitas uma soma e uma subtração entre dois resultados da camada anterior.

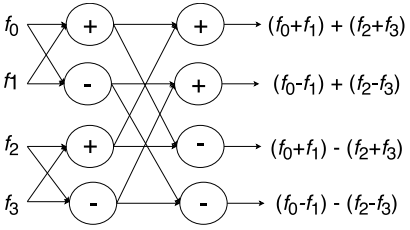


Figura 5: Esquemático de uma *butterfly* para quatro entradas.

3 TRABALHOS CORRELATOS

Jou (2007), Zhu e Xiong (2009) apresentaram um método para cálculo da SATD, chamado de Método Isento de Transformada - *Transform-Exempted* (TE). Este método otimiza o cálculo da SATD através de operações de máximos e absolutos, conforme a Equação 3.1, para substituir a segunda transformada 1-D e o primeiro nível de somas absolutas.

$$| a+b | + | a-b | = 2 \times max(a,b) \tag{3.1}$$

A implementação foi feita no *software* de referência do H.264, chamada de *Joint Model* (JM). Ao comparar o método proposto com o cálculo convencional e com a SATD baseada em FHT houve redução de 38% e 17% do número de operações executadas, respectivamente.

Kim e Park (2009) propõem uma arquitetura para blocos de tamanho variável de até 16×16 com reúso de cálculos na SAD. A entrada é um macrobloco 16×16, dividido em quatro blocos 8×8. Cada um destes blocos é novamente dividido em quatro blocos 4×4, totalizando dezesseis blocos 4×4. Após a divisão dos blocos, é calculada a SAD de cada bloco 4×4. Posteriormente, para obter o valor da SAD de um bloco maior é feita a soma das SADs dos blocos menores. Por exemplo, para calcular a SAD de um bloco 8×8 são necessários 4 pares de blocos originais e candidatos com tamanho 4×4.

Dominges Jr et al. (2011) desenvolveram duas arquiteturas para SATD, uma para blocos 4×4 e outra para blocos 8×8. Ambas são capazes de processar um bloco por ciclo de relógio através do uso de *pipeline*. Na arquitetura para blocos 8×8, a primeira transformada é duplicada para aumentar a capacidade de processamento. Todas as arquiteturas foram descritas em *VHDL* e sintetizadas para *FPGA* da Xilinx. Contudo, não foram apresentados os resultados de energia e potência obtidos pela síntese das arquiteturas. Também foi feita uma avaliação em *software* utilizando o JM para avaliar a SAD, Soma das Diferenças Quadráticas - *Sum of Squared Differences* (SSD) e SATD. Quando os resultados da SATD foram comparados com SAD e SSD, os autores concluíram que a SATD obteve os melhores resultados de qualidade e taxa de compressão.

Em Liu et al. (2011a) é proposta uma arquitetura de SATD para blocos 4×4 e 8×8 com reúso de *hardware* para a Estimação de Movimento Fracionária - *Fractional Motion Estimation* (FME). Segundo os autores, quando comparada com arquiteturas convencionais, a arquitetura proposta consegue diminuir a complexidade computacional e aumentar o paralelismo. A arqui-

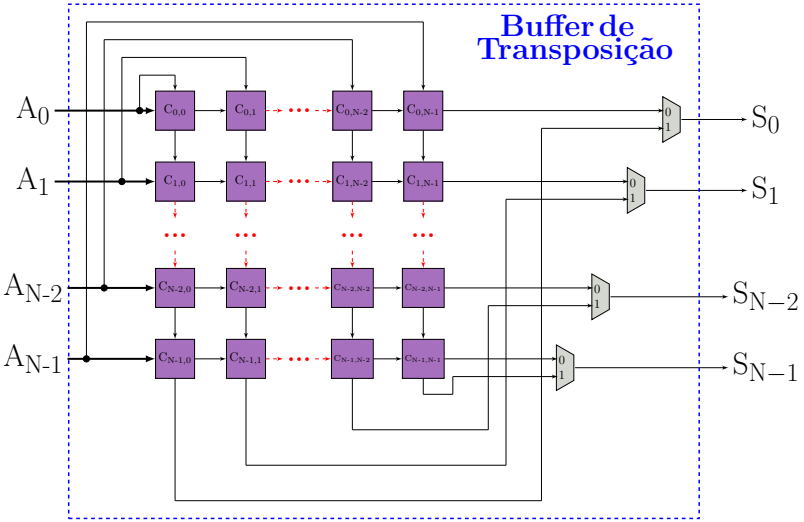
tetura consegue diminuir a latência através do uso de dois interpoladores, alimentando de forma constante o cálculo de SATD. Contudo, não são apresentados valores de energia e potência.

Liu et al. (2011b) propõem uma arquitetura para SATD baseada na recursividade da HT e com truncamento de *bits* para blocos 4×4 e 8×8. A recursividade da HT é feita calculando as SATDs dos blocos 4×4 e utilizando operações sobre matrizes nos resultados parciais das 4×4 para calcular a SATD do bloco 8×8. A arquitetura permite o cálculo simultâneo de apenas dois blocos de tamanho 4×4, uma vez que o *hardware* é utilizado tanto para o cálculo dos blocos 4×4 quanto para 8×8. O truncamento provoca perda de qualidade na codificação ao descartar *bits*. Houve redução entre 13% e 30% na área e o consumo de energia diminuiu entre 12,6% e 32,4%.

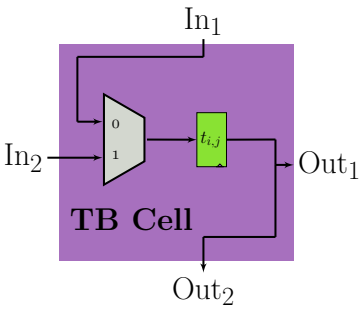
Cancellier et al. (2015) propuseram três arquiteturas para o cálculo da SATD, todas para blocos de tamanho 4×4. Duas utilizam o método TE enquanto a outra implementa a FHT. Segundo os autores, a melhor alternativa para implementar *hardware* dedicado para cálculo da SATD baseado na Hadamard é a arquitetura com TE. Esta consegue reduzir a área e o consumo energético devido à união do último nível da transformada com as primeiras somas absolutas. A arquitetura para TE obteve redução de 26,5% e 19,25% para energia e área, respectivamente, quando comparada com a FHT para TE-SATD na Tree-4stages. Já ao ser comparada a FHT com a TE para *Buffer* de Transposição - *Transpose Buffer* (TB) a energia e a área reduziram em 15,05% e 9,8%, respectivamente.

O *Buffer* de Transposição - *Transpose Buffer* (TB) é utilizado para armazenar e transpor uma matriz de valores. A Figura 6a apresenta o esquemático do TB em que os retângulos com a letra "C" representam as células básicas apresentadas na Figura 6b. A célula básica de um TB é constituída por um multiplexador (*mux*) e um *flip-flop*. O multiplexador é responsável por selecionar o valor utilizado como entrada do *flip-flop*.

O TB funciona de duas formas: movendo os valores por linha ou por coluna. Para cada novo valor inserido, os existentes são movidas para a próxima camada de células e este processo se repete até os primeiros valores inseridos chegarem na última camada de células. Neste momento, o sentido do movimento e inserção dos valores é invertido: se era por linhas, passa a ser por colunas e vice-versa. A mudança de sentido de operação ocorre através de um sinal gerado pelo sistema de controle do TB. No início, existe uma diferença de alguns ciclos entre a inserção de um valor e ele estar disponível na saída. Esta diferença ocorre pela necessidade do valor percorrer os caminhos internos do TB entre a entrada e a saída. Os valores armazenados nas células básicas do TB sempre se referem a uma linha da Hadamard multiplicada por uma coluna das diferenças. A transposição do TB é uma forma de transpor a



(a) *Buffer* de transposição com N quatro entradas (N_0, N_1, A_{N-2} e A_{N-1}) e N saídas (S_0, S_1, S_{N-2} e S_{N-1}). Os quadrados marcados com um "C" são as células básicas.



(b) Célula básica que compõem o *buffer* de transposição. In_1 e In_2 são as entradas e Out_1 e Out_2 são as saídas.

Figura 6: *Datapath* do *buffer* de transposição e das células básicas. Adaptado de Seidel et al. (2016).

matriz resultante de $\mathbf{H} \times \mathbf{D}$ para ser multiplicada por $\mathbf{H}^{\mathrm{T}}$.

Silveira et al. (2015) apresentam uma arquitetura para blocos 8×8 dividida em duas etapas, a horizontal e a vertical. Na horizontal, para cada ciclo, uma linha da matriz de entrada é processada e os resultados parciais são salvos em uma estrutura composta por registradores e multiplexadores. Para a vertical, os valores são escolhidos através dos multiplexadores de forma que seja fornecida uma coluna por vez para a segunda 1-D HT e os resultados são

utilizados no cálculo final da SATD.

Soares et al. (2016) propuseram a computação aproximada da SATD para diminuir o gasto energético. A computação aproximada ocorre através do descarte dos coeficientes menos significativos da 2-D HT, por ser a mais computacionalmente intensiva na SATD. Inicialmente, foi implementado e testado no HM utilizando quatro vídeos das Condições Comuns de Teste - *Common Test Conditions*s (CTCs). A seguir foram desenvolvidas seis arquiteturas para SATD, todas para blocos 4×4 e com quantidade de coeficientes descartados diferentes. As arquiteturas foram sintetizadas com a ferramenta Cadence RTL Compiler (CADENCE, 2015) com o FreePDK para 45 nm da NanGate (2015). A arquitetura que descarta 10 coeficientes obteve 70,7% de economia de energia por operação com perda de 0,008 dB (BD-PSNR), quando comparada com a arquitetura com cálculo preciso da SATD.

Seidel et al. (2016) propuseram duas arquiteturas para cálculo da SATD utilizando a FHT para blocos de tamanho 4×4, 8×8, 16×16 e 32×32, uma com TB e outra com *Buffer* Linear - *Linear Buffer* (LB). Todas as arquiteturas foram sintetizadas para *Low-Vdd/High-Vt* (LH) e Nominal (NN). As arquiteturas com TB foram as mais eficientes em termos de energia para todos os tamanhos de blocos para síntese LH, reduzindo em até 39,1% o consumo energético quando comparada com os resultados da síntese NN.

Seidel, Güntzel e Agostini (2016) propuseram o uso de Eliminação Parcial de Distorção - *Partial Distortion Elimination* (PDE) na fase de particionamento de blocos durante a execução da SATD. A PDE pula a computação da SATD quando o cálculo excede o atual mínimo, e com isto a eficiência de codificação não é alterada. O método proposto foi testado no HM versão 16.9 utilizando as CTCs. Foram obtidos 18,85% de eliminações para 4×4 e 9,99% para 8×8. Também foram implementadas arquiteturas de *hardware* totalmente paralelas para SATD de blocos 4×4 e 8×8. Todas foram sintetizadas utilizando o *Synopsys® Design Compiler®* (DC®) com a biblioteca de células padrão de 45 nm da *Taiwan Semiconductor Manufacturing Company Limited* (TSMC). O aumento da área, no pior caso, foi de 1,72% e a eficiência energética melhorou em 15,9% no melhor caso.

Uma arquitetura configurável para cálculo de tamanhos múltiplos da SATD (2×2, 4×4 e 8×8) é apresentada por Silveira (2016). Para diminuir a área ocupada, os blocos menores são utilizados como parte dos blocos maiores para o cálculo da SATD. Foram sintetizadas três versões da arquitetura: uma com somadores padrão da ferramenta, uma com somadores e subtratores compressores e uma com somadores/compressores propostos no próprio trabalho. A substituição das árvores de somadores por somadores/subtratores compressores provocou uma redução média da potência total. Somadores compressores são estruturas que aceleram a soma, pois é possível fazer a ope-

ração com a mínima propagação de *carry*. A configurabilidade do tamanho de bloco calculado é um ponto positivo, contudo as partes da arquitetura que não são utilizadas para o cálculo de um determinado tamanho de bloco permanecem ligadas, o que provoca aumento do consumo energético. Também, é apenas disponibilizado o valor da SATD do maior bloco calculado. Por exemplo, se for calculada a SATD do bloco 4×4 não é possível ter acesso aos valores das SATDs dos blocos 2×2 e as partes utilizadas apenas para cálculo da 8×8 permanecem ligadas desperdiçando energia. A Figura 7 apresenta o esquemático da arquitetura em que o retângulo tracejado e o pontilhado representam os blocos da arquitetura que geram sobrecusto para o cálculo da SATD dos blocos 2×2 e 4×4, respectivamente.

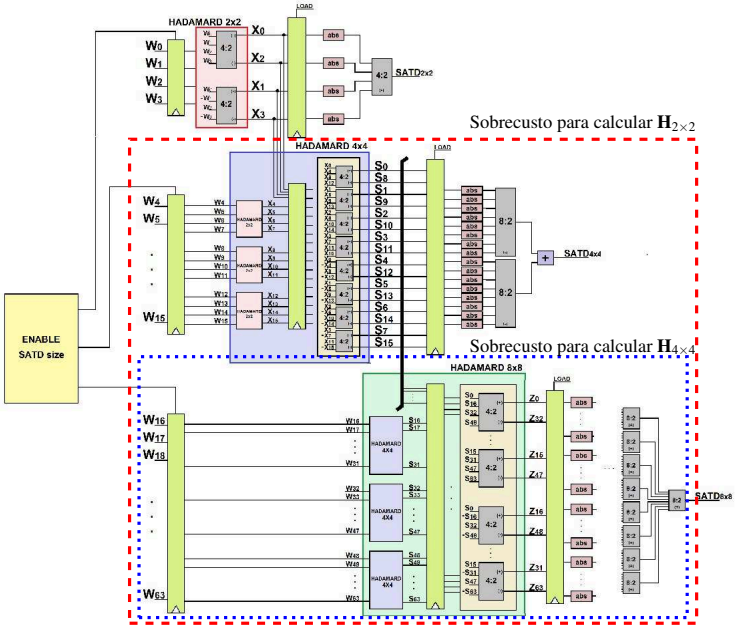


Figura 7: Comparação de área, utilizando como base o valor da arquitetura proposta neste trabalho.

Na Tabela 1 são apresentadas as principais características dos trabalhos descritos neste capítulo.

Tabela 1: Características dos trabalhos relacionados.

Trabalho	Métrica		Tamanho de Bloco						Reúso	
	SAD	SATD	2x2	4x4	8x8	16x16	32x32	NQ[1]	Cálculos	*Hardware*
Kim e Park (2009)	X			X	X	X		X	X	
Jou (2007), Zhu e Xiong (2009)		X		X						
Dominges Jr et al. (2011)		X		X	X					
Liu et al. (2011a)		X		X	X					
Liu et al. (2011b)		X		X	X					X
Cancellier et al. (2015)		X		X						
Silveira et al. (2015)		X		X						
Soares et al. (2016)		X		X						
Seidel, Güntzel e Agostini (2016)		X		X	X	X	X	X		
Seidel et al. (2016)		X		X	X	X	X			
Silveira (2016)		X	X	X	X					X
Este Trabalho		X		X	X				X	

[1] Blocos não quadrados.

4 PROPOSTA

Neste trabalho é apresentada uma proposta para o reúso de cálculos na SATD. Também é proposta uma arquitetura de *hardware* para a SATD utilizando o reúso de cálculos. O reúso na SATD baseia-se na mesma ideia empregada para reúso na SAD, que consiste em calcular a métrica dos blocos menores e utilizar estes valores para obter o valor da métrica do bloco maior. Contudo, não é possível reaproveitar o cálculo completo da métrica, apenas as transformadas. Para tanto, é necessário combinar os valores parciais da segunda transformada dos blocos menores através de uma matriz de ajustes. Em seguida é calculada a soma absoluta das diferenças transformadas. A arquitetura foi desenvolvida com base na arquitetura proposta por Seidel et al. (2016), a qual foi modificada para também suportar o cálculo de blocos de tamanhos 8×8 através do reúso.

Para reusar os cálculos das transformadas, é necessário utilizar uma matriz de ajuste ($\mathbf{A}$) construída a partir da Equação 4.1, onde, $\otimes$ é o produto de Kronecker (SHI, 1997), $n \in \mathbb{N}^*$ e $\mathbf{I}$ é a matriz identidade.

$$\mathbf{A}_{2^n \times 2^n} = \mathbf{H}_{2\times 2} \otimes \mathbf{I}_{2^{n-1} \times 2^{n-1}} \tag{4.1}$$

A matriz Hadamard 8×8 ($\mathbf{H}_{8\times 8}$) será utilizada por ser a maior Hadamard presente no *software* de referência do HEVC. Assim, considerando a matriz $\mathbf{Y}_{8\times 8}$ composta pelas matrizes de diferenças transformadas $\mathbf{TD}_{4\times 4}$, a $\mathbf{TD}_{8\times 8}$ pode ser obtida como segue:

$$\mathbf{TD}_{8\times 8} = \mathbf{A}_{8\times 8} \times \mathbf{Y}_{8\times 8} \times \mathbf{A}^T_{8\times 8} \tag{4.2}$$

onde

$$\mathbf{A}_{8\times 8} = \mathbf{H}_{2\times 2} \otimes \mathbf{I}_{4\times 4} = \begin{bmatrix} \mathbf{I}_{4\times 4} & \mathbf{I}_{4\times 4} \\ \mathbf{I}_{4\times 4} & -\mathbf{I}_{4\times 4} \end{bmatrix} \tag{4.3}$$

4.1 DEMONSTRAÇÃO

Para demonstrar o reúso de cálculos na SATD é necessário utilizar a recursividade, apresentada no Capítulo 2 e a separabilidade da matriz Hadamard que é demonstrada a seguir.

Utilizando a propriedade da separabilidade (associatividade da multiplicação de matrizes) que foi apresentada na Seção 2.3, o cálculo da transformada pode ser dividido em duas multiplicações de matrizes. A primeira entre $\mathbf{H}$ e $\mathbf{D}$ e a segunda entre o resultado da primeira multiplicação e a $\mathbf{H}^{\mathbf{T}}$.

Assim, o primeiro passo para o desenvolvimento da demonstração é construir $\mathbf{H}_{8\times8}$ através da propriedade da recursividade. A matriz $\mathbf{H}_{8\times8}$ é construída pela composição de quatro $\mathbf{H}_{4\times4}$, como é apresentado na Equação 4.4.

$$\mathbf{H}_{8\times8} = \left[\begin{array}{c|c} \mathbf{H}_{4\times4} & \mathbf{H}_{4\times4} \\ \hline \mathbf{H}_{4\times4} & -\mathbf{H}_{4\times4} \end{array}\right] \tag{4.4}$$

O segundo passo consiste em particionar $\mathbf{D}$ em quatro submatrizes de mesmo tamanho, como segue:

$$\mathbf{D}_{8\times8} = \left[\begin{array}{c|c} \mathbf{D}_{1,1} & \mathbf{D}_{1,2} \\ \hline \mathbf{D}_{2,1} & \mathbf{D}_{2,2} \end{array}\right] \tag{4.5}$$

A primeira parte da transformada é calculada através da multiplicação por partições da matriz $\mathbf{H}_{8\times8}$ pela matriz $\mathbf{D}_{8\times8}$ e que resulta na matriz $\mathbf{Y}_{8\times8}$, como está representada na Equação 4.6.

$$\mathbf{Y}_{8\times8} = \mathbf{H}_{8\times8} \times \mathbf{D}_{8\times8} = \left[\begin{array}{c|c} \mathbf{H}_{4\times4} & \mathbf{H}_{4\times4} \\ \hline \mathbf{H}_{4\times4} & -\mathbf{H}_{4\times4} \end{array}\right] \times \left[\begin{array}{c|c} \mathbf{D}_{1,1} & \mathbf{D}_{1,2} \\ \hline \mathbf{D}_{2,1} & \mathbf{D}_{2,2} \end{array}\right] = \left[\begin{array}{c|c} \mathbf{Y}_{1,1} & \mathbf{Y}_{1,2} \\ \hline \mathbf{Y}_{2,1} & \mathbf{Y}_{2,2} \end{array}\right] \tag{4.6}$$

A Equação 4.7 mostra as partições envolvidas em cada multiplicação para obter a matriz $\mathbf{Y}_{8\times8}$ e a disposição de suas partições são apresentadas na Equação 4.6.

$$\begin{array}{ll} \mathbf{Y}_{1,1} = \mathbf{H}_{4\times4} \times \mathbf{D}_{1,1} + \mathbf{H}_{4\times4} \times \mathbf{D}_{2,1} & [I] \\ \mathbf{Y}_{1,2} = \mathbf{H}_{4\times4} \times \mathbf{D}_{1,2} + \mathbf{H}_{4\times4} \times \mathbf{D}_{2,2} & [II] \\ \mathbf{Y}_{2,1} = \mathbf{H}_{4\times4} \times \mathbf{D}_{1,1} - \mathbf{H}_{4\times4} \times \mathbf{D}_{2,1} & [III] \\ \mathbf{Y}_{2,2} = \mathbf{H}_{4\times4} \times \mathbf{D}_{1,2} - \mathbf{H}_{4\times4} \times \mathbf{D}_{2,2} & [IV] \end{array} \tag{4.7}$$

Em seguida as partições são multiplicadas pelas partições de $\mathbf{H}_{8\times8}$ resultando na matriz $\mathbf{F}_{8\times8}$ (Equação 4.8). Ao multiplicar a primeira linha de $\mathbf{Y}_{8\times8}$ pela primeira coluna de $\mathbf{H}_{8\times8}$ é obtida a $\mathbf{F}_{1,1}$. A multiplicação da primeira linha de $\mathbf{Y}_{8\times8}$ pela segunda coluna de $\mathbf{H}_{8\times8}$ resulta em $\mathbf{F}_{1,2}$. A $\mathbf{F}_{2,1}$ é obtida a partir da multiplicação da segunda linha de $\mathbf{Y}_{8\times8}$ pela primeira coluna de $\mathbf{H}_{8\times8}$. Por fim, após a multiplicação da segunda linha de $\mathbf{Y}_{8\times8}$ pela segunda coluna de $\mathbf{H}_{8\times8}$ obtém-se $\mathbf{F}_{2,2}$. A Equação 4.9 apresenta as equações resultantes das multiplicações descritas anteriormente.

$$\mathbf{F}_{8\times8} = \mathbf{Y}_{8\times8} \times \mathbf{H}_{8\times8} = \left[\begin{array}{c|c} \mathbf{F}_{1,1} & \mathbf{F}_{1,2} \\ \hline \mathbf{F}_{2,1} & \mathbf{F}_{2,2} \end{array}\right] \tag{4.8}$$

$$\begin{aligned}
\mathbf{F}_{1,1} &= \mathbf{Y}_{1,1} \times \mathbf{H}_{4\times4} + \mathbf{Y}_{1,2} \times \mathbf{H}_{4\times4} \quad [I] \\
\mathbf{F}_{1,2} &= \mathbf{Y}_{1,1} \times \mathbf{H}_{4\times4} - \mathbf{Y}_{1,2} \times \mathbf{H}_{4\times4} \quad [II] \\
\mathbf{F}_{2,1} &= \mathbf{Y}_{2,1} \times \mathbf{H}_{4\times4} + \mathbf{Y}_{2,2} \times \mathbf{H}_{4\times4} \quad [III] \\
\mathbf{F}_{2,2} &= \mathbf{Y}_{2,1} \times \mathbf{H}_{4\times4} - \mathbf{Y}_{2,2} \times \mathbf{H}_{4\times4} \quad [IV]
\end{aligned} \tag{4.9}$$

Ao substituir $\mathbf{Y}_{1,1}, \mathbf{Y}_{1,2}, \mathbf{Y}_{2,1}, \mathbf{Y}_{2,2}$ (Equação 4.9) pelas equações I, II, III e IV apresentadas na Equação 4.7 é obtida a Equação 4.10.

$$\begin{aligned}
\mathbf{F}_{1,1} &= (\mathbf{H}_{4\times4} \times \mathbf{D}_{1,1} + \mathbf{H}_{4\times4} \times \mathbf{D}_{2,1}) \times \mathbf{H}_{4\times4} + (\mathbf{H}_{4\times4} \times \mathbf{D}_{1,2} + \mathbf{H}_{4\times4} \times \mathbf{D}_{2,2}) \times \mathbf{H}_{4\times4} \\
\mathbf{F}_{1,2} &= (\mathbf{H}_{4\times4} \times \mathbf{D}_{1,1} + \mathbf{H}_{4\times4} \times \mathbf{D}_{2,1}) \times \mathbf{H}_{4\times4} - (\mathbf{H}_{4\times4} \times \mathbf{D}_{1,2} + \mathbf{H}_{4\times4} \times \mathbf{D}_{2,2}) \times \mathbf{H}_{4\times4} \\
\mathbf{F}_{2,1} &= (\mathbf{H}_{4\times4} \times \mathbf{D}_{1,1} - \mathbf{H}_{4\times4} \times \mathbf{D}_{2,1}) \times \mathbf{H}_{4\times4} + (\mathbf{H}_{4\times4} \times \mathbf{D}_{1,2} - \mathbf{H}_{4\times4} \times \mathbf{D}_{2,2}) \times \mathbf{H}_{4\times4} \\
\mathbf{F}_{2,2} &= (\mathbf{H}_{4\times4} \times \mathbf{D}_{1,1} - \mathbf{H}_{4\times4} \times \mathbf{D}_{2,1}) \times \mathbf{H}_{4\times4} - (\mathbf{H}_{4\times4} \times \mathbf{D}_{1,2} - \mathbf{H}_{4\times4} \times \mathbf{D}_{2,2}) \times \mathbf{H}_{4\times4}
\end{aligned} \tag{4.10}$$

Aplicando a propriedade da distributividade na Equação 4.10 é obtida a Equação 4.11.

$$\begin{aligned}
\mathbf{F}_{1,1} = \quad & [(\mathbf{H}_{4\times4} \times \mathbf{D}_{1,1} \times \mathbf{H}_{4\times4}) + (\mathbf{H}_{4\times4} \times \mathbf{D}_{2,1} \times \mathbf{H}_{4\times4})] \\
& + [(\mathbf{H}_{4\times4} \times \mathbf{D}_{1,2} \times \mathbf{H}_{4\times4}) + (\mathbf{H}_{4\times4} \times \mathbf{D}_{2,2} \times \mathbf{H}_{4\times4})] \\
\mathbf{F}_{1,2} = \quad & [(\mathbf{H}_{4\times4} \times \mathbf{D}_{1,1} \times \mathbf{H}_{4\times4}) + (\mathbf{H}_{4\times4} \times \mathbf{D}_{2,1} \times \mathbf{H}_{4\times4})] \\
& - [(\mathbf{H}_{4\times4} \times \mathbf{D}_{1,2} \times \mathbf{H}_{4\times4}) + (\mathbf{H}_{4\times4} \times \mathbf{D}_{2,2} \times \mathbf{H}_{4\times4})] \\
\mathbf{F}_{2,1} = \quad & [(\mathbf{H}_{4\times4} \times \mathbf{D}_{1,1} \times \mathbf{H}_{4\times4}) - (\mathbf{H}_{4\times4} \times \mathbf{D}_{2,1} \times \mathbf{H}_{4\times4})] \\
& + [(\mathbf{H}_{4\times4} \times \mathbf{D}_{1,2} \times \mathbf{H}_{4\times4}) - (\mathbf{H}_{4\times4} \times \mathbf{D}_{2,2} \times \mathbf{H}_{4\times4})] \\
\mathbf{F}_{2,2} = \quad & [(\mathbf{H}_{4\times4} \times \mathbf{D}_{1,1} \times \mathbf{H}_{4\times4}) - (\mathbf{H}_{4\times4} \times \mathbf{D}_{2,1} \times \mathbf{H}_{4\times4})] \\
& - [(\mathbf{H}_{4\times4} \times \mathbf{D}_{1,2} \times \mathbf{H}_{4\times4}) - (\mathbf{H}_{4\times4} \times \mathbf{D}_{2,2} \times \mathbf{H}_{4\times4})]
\end{aligned} \tag{4.11}$$

Pode-se observar na Equação 4.11 que as multiplicações dentro dos parênteses são transformadas de blocos menores. Desta forma, é possível calcular a transformada de blocos menores e, através do reúso destes cálculos, obter o valor da transformada maior. Para tal, é necessário utilizar a matriz de ajuste $\mathbf{A}$ como é apresentada na Equação 4.2.

Em seguida, $\mathbf{Y}_{8\times8}$ é aplicada na Equação 4.12 e obtida a matriz $\mathbf{F}_{8\times8}$.

$$\mathbf{F}_{8\times8} = \mathbf{A}_{8\times8} \times \mathbf{Y}_{8\times8} \times {\mathbf{A}_{8\times8}}^{T} \tag{4.12}$$

Para obter a SATD do bloco 8×8, basta calcular o valor absoluto de todos os elementos de $\mathbf{F}_{8\times8}$ e em seguida somá-los.

Com esta demonstração, é possível notar que algumas partes do cálculo da transformada podem ser paralelizados e até mesmo reutilizados para o cálculo de transformadas de tamanho maior.

4.1.1 Exemplo

Para melhor compreensão da demonstração, a seguir é apresentado um exemplo de cálculo da SATD com reúso de cálculos. No exemplo será utilizada uma matriz de diferenças (**D**) de tamanho 4×4 e matrizes Hadamard de 2×2 ($\mathbf{H}_{2\times2}$).

Inicialmente, **D** é dividida em quatro submatrizes de tamanho 2×2, conforme apresentada na Equação 4.13.

$$\mathbf{D}_{4\times4} = \left[\begin{array}{c|c} \mathbf{D}_{1,1} & \mathbf{D}_{1,2} \\ \hline \mathbf{D}_{2,1} & \mathbf{D}_{2,2} \end{array}\right] = \left[\begin{array}{cc|cc} 1 & 2 & 1 & 2 \\ 3 & 4 & 3 & 4 \\ \hline 1 & 2 & 1 & 2 \\ 3 & 4 & 3 & 4 \end{array}\right] \tag{4.13}$$

Em seguida, é calculada a Equação 4.14 e obtida a submatriz de valores parciais representada pela Equação 4.15.

$$\mathbf{Y}_{1,1} = \mathbf{H}_{2\times2} \times \mathbf{D}_{1,1} \times \mathbf{H}_{2\times2} \tag{4.14}$$

$$\mathbf{Y}_{1,1} = \begin{bmatrix} 10 & -2 \\ -4 & 0 \end{bmatrix} \tag{4.15}$$

O mesmo cálculo é feito com $\mathbf{D}_{1,2}$, $\mathbf{D}_{2,1}$ e $\mathbf{D}_{2,2}$. Gerando, ao todo, quatro matrizes de resultados parciais $\mathbf{Y}_{1,1}$, $\mathbf{Y}_{1,2}$, $\mathbf{Y}_{2,1}$ e $\mathbf{Y}_{2,2}$. Para calcular a SATD 2×2, basta aplicar a operação de absoluto nos elementos das submatrizes e em seguida fazer o somatório dos mesmos. Neste exemplo, as submatrizes serão iguais pois todas contém os mesmos elementos e assim as SATDs 2×2 serão iguais a 16.

Para calcular a SATD 4×4 é necessário utilizar a matriz de ajuste construída segundo a Equação 4.1 com $n = 4$ resultando na matriz representada na Equação 4.16. Em seguida, é construída a matriz $\mathbf{Y}_{8\times8}$ através da composição das matrizes $\mathbf{Y}_{i,j}$ conforme apresentado na Equação 4.17.

$$\mathbf{A}_{4\times4} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix} \tag{4.16}$$

$$\mathbf{Y}_{4\times4} = \left[\begin{array}{c|c} \mathbf{Y}_{1,1} & \mathbf{Y}_{1,2} \\ \hline \mathbf{Y}_{2,1} & \mathbf{Y}_{2,2} \end{array}\right] \tag{4.17}$$

A seguir, $\mathbf{Y}_{4\times4}$ é aplicada na Equação 4.18 resultando na Equação 4.19.

$$\mathbf{F}_{4\times4} = \mathbf{A}_{4\times4} \times \mathbf{Y}_{4\times4} \times {\mathbf{A}_{4\times4}}^{T} \tag{4.18}$$

$$\mathbf{F}_{4\times4} = \begin{bmatrix} 40 & -8 & 0 & 0 \\ -16 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{4.19}$$

Para obter a SATD 4×4, basta aplicar a operação de absoluto na matriz $\mathbf{F}_{4\times4}$ e somar os elementos. No presente exemplo, o valor obtido é 64.

4.2 ARQUITETURA PROPOSTA

A arquitetura proposta é baseada na demonstração apresentada na Seção 4.1 e no trabalho de Seidel et al. (2016). Está é dividida em duas partes: o bloco operativo e o bloco de controle.

4.2.1 Bloco Operativo

Na Figura 8 é apresentado o bloco operativo da arquitetura proposta, e esta pode ser dividida em três partes principais. Duas estruturas paralelas para cálculo da SATD 4×4 (Figura 9) e uma estrutura para cálculo da SATD 8×8 (Figura 10). A capacidade de processamento é de quatro blocos 4×4 em paralelo, sendo dois originais ($\mathbf{B}^{\text{ori}}$) e dois candidatos ($\mathbf{B}^{\text{can}}$). A entrada dos dados é feita linha-a-linha, ou seja, a cada ciclo de relógio são inseridos novos valores.

Os *buffers* de transposição utilizados na arquitetura foram construídos com base nos apresentados na Figura 6. A primeira e a segunda transformadas são compostas por *butterflies*, conforme a Figura 5.

4.2.1.1 SATD 4×4

Na Figura 9 é apresentado o esquemático da parte responsável pelo cálculo dos blocos 4×4. Na arquitetura proposta existem duas destas estruturas operando de forma simultânea. Dessa forma, é possível calcular dois blocos 4×4 em paralelo. Estas duas estruturas alimentam a parte responsável pelo cálculo dos blocos 8×8. Os passos realizados pelas duas estruturas são os mesmos e são descritos a seguir.

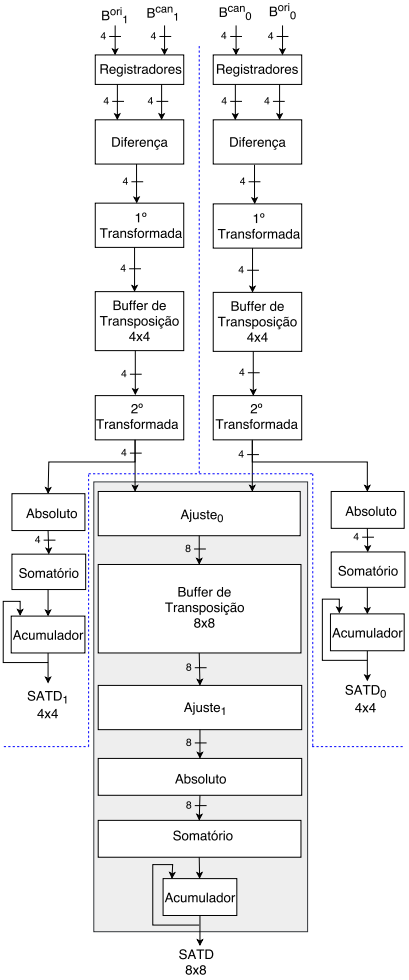


Figura 8: Esquemático da arquitetura proposta para cálculo de SATD com reúso de cálculos. As linhas tracejadas separam as duas estruturas para o cálculo das SATDs 4×4 e no retângulo sombreado está a estrutura para cálculo da 8×8. Os números assinalados na arquitetura representam a quantidade de dados transmitidos entre as partes. Dos oito pares de valores de entrada da arquitetura, $\mathbf{B}_0^{\text{ori}}$ e $\mathbf{B}_0^{\text{can}}$ recebem quatro pares e os demais são recebidos por $\mathbf{B}_1^{\text{ori}}$ e $\mathbf{B}_1^{\text{can}}$.

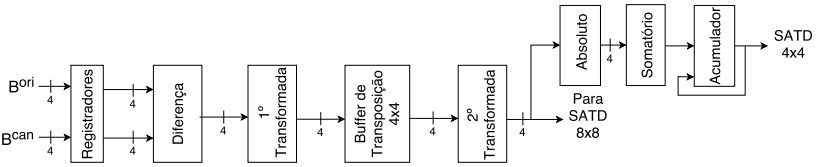


Figura 9: Esquemático para o cálculo da SATD 4×4 da arquitetura proposta. Os números assinalados na arquitetura representam a quantidade de valores transmitidos entre as partes.

Inicialmente, são inseridos dois vetores de tamanho 4×1, sendo um de $\mathbf{B}^{\text{ori}}$ e outro de $\mathbf{B}^{\text{can}}$, no *Registradores*, que é uma barreira de registradores para manter estáveis os valores por um ciclo de relógio. A seguir, a *Diferença* subtrai os valores do vetor de $\mathbf{B}^{\text{ori}}$ dos valores do vetor de $\mathbf{B}^{\text{can}}$ e obtém um vetor $\vec{v}_0$ com as diferenças. A primeira transformada multiplica a $H_{4\times4}$ por $\vec{v}_0$ através de uma *butterfly* e o resultado é armazenado no *Buffer de Transposição* 4×4. A segunda transformada multiplica os valores da saída do *buffer* pela $H_{4\times4}$. Neste ponto, os valores da saída da segunda transformada são enviados para dois blocos distintos, para o *Absoluto* e para o cálculo da SATD 8×8.

O *Absoluto* obtém os valores absolutos dos resultados da segunda transformada e os disponibiliza na sua saída. O *Somatório* utiliza uma árvore de somadores para obter a soma de todas as saídas do *Absoluto* e enviar para o *Acumulador*. No *Acumulador* é somada a saída do *Somatório* com o valor armazenado no registrador interno dele e o resultado é disponibilizado na saída, aqui chamada de SATD 4×4.

4.2.1.2 SATD 8×8

Na Figura 10 é apresentada a parte da arquitetura responsável pelo cálculo da SATD 8×8. Cada entrada, E_0 e E_1, recebe um vetor de dimensões 4×1 das estruturas apresentadas na Subseção 4.2.1.1. O $Ajuste_0$ é necessário para compor os dois vetores de entrada em um vetor de dimensões 8×1 para ser utilizado no cálculo da SATD 8×8. A construção do vetor 8×1 é feita através da multiplicação dos dois vetores 4×1 pela matriz apresentada na Equação 4.20 e os valores resultantes são armazenadas no TB.

O bloco $Ajuste_1$ é feito através da implementação em *hardware* da Equação 4.3. Tal bloco multiplica as saídas do TB pela matriz da Equação 4.20 para completar os cálculos da transformada do bloco 8×8. O *Absoluto*, o *Somatório* e o *Acumulador* são estruturas similares as utilizadas para 4×4,

com a diferença que são utilizados mais operadores e o número de ciclos é maior. Ao término do cálculo do bloco, é disponibilizado o valor da SATD 8×8 como saída.

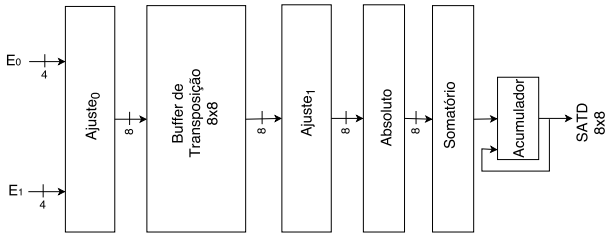


Figura 10: Esquemático para o cálculo da SATD 8×8 da arquitetura proposta. Os números assinalados na arquitetura representam a quantidade de dados transmitidos em paralelo entre as partes.

$$\mathbf{A}_{8\times 8} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 \end{bmatrix} \tag{4.20}$$

4.2.2 Bloco de Controle

O comportamento do bloco operativo (Figura 8) foi projetado para seguir o comportamento da Máquina de Estados Finitos - *Finite State Machine* (FSM) apresentada na Figura 11. Cada estado tracejado representa um conjunto de estados que foram agrupados e que são descritos a seguir. Foram adotadas as seguintes convenções, *N* é o índice que indica o número do estado dentro de um agrupamento e as setas tracejadas representam as conexões entre um estado de origem e um de destino. O estado $\mathtt{I}_{N-1}$ representa dezenove estados($\mathtt{I}_1$ até $\mathtt{I}_{19}$) utilizados para popular a arquitetura. O estado $\mathtt{C}_{N-1}$ representa quatro estados ($\mathtt{C}_3$ à $\mathtt{C}_7$) responsáveis pelo controle da arquitetura enquanto esta estiver operando no laço. E finalmente, o estado $\mathtt{T}_{N-1}$ representa quatro estados ($\mathtt{T}_3$ até $\mathtt{T}_7$) que descarregam o TB 8×8 e calculam as SATDs dos últimos blocos.

As duas primeiras SATDs 4×4 são liberadas no estado $\mathtt{I}_9$ e para cada

quatro ciclos de relógio são liberadas mais duas. Já a primeira SATD 8×8 fica pronta somente no estado $\texttt{PC}_1$ e a cada oito ciclos é liberado um novo valor. Os últimos valores para os blocos 4×4 são liberados no estado $\texttt{T}_2$, já para o 8×8 é liberado no estado $\texttt{L}_1$.

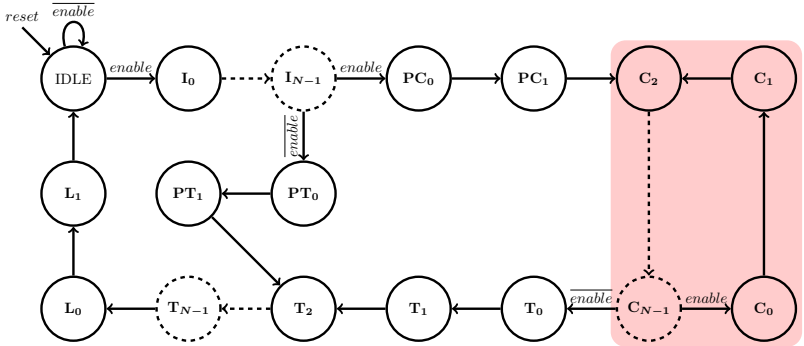


Figura 11: FSM utilizada para controlar a arquitetura proposta. Os estados tracejados representam agrupamentos de estados e as setas tracejadas representam as transições entre os estados do agrupamento. Similar ao trabalho de Seidel et al. (2016).

A FSM segue os seguintes passos:

- O sinal `reset` é assíncrono.
- Estando em *IDlE*, se o sinal de `enable` permanecer em "0", a FSM permanece no estado `IDLE` e os sinais de `reset` das SATDs 4×4 e 8×8 e dos TBs permanecem ativados.
- Se o sinal de `enable` muda e permanece no estado lógico "1", ocorre a transição para o estado $\texttt{I}_0$ e as entradas da arquitetura e os *buffers* de transposição 4×4 são habilitados.
- No estado $\texttt{I}_{19}$, caso o sinal de `enable` esteja em zero significa que existem apenas quatro blocos 8×8 para serem calculados e deve ocorrer a transição para o estado $\texttt{PT}_0$.
- Se `enable` for "1" então existem mais de quatro blocos e ocorre a transição para o estado $\texttt{PC}_0$.
- Se no estado $\texttt{C}_7$, o sinal de `enable` estiver em estado lógico "1", significa que existem mais blocos para serem calculados e o laço deve ser repetido.

- Caso *enable* esteja em zero, não existem mais blocos para serem computados e a FSM deve fazer a transição para o estado T_0, além de desativar as entradas de dados da arquitetura.

- Nos demais estados, não existem condições de transição.

A seguir é feita a análise do número de operações feitas pela arquitetura e uma comparação com a FHT sem aplicar o reúso. A *Diferença* e o *Absoluto* fazem 4 operações cada um, totalizando 8 operações. Para a primeira e segunda transformadas são feitas 16 operações, sendo 8 para cada transformada. O *Somatório* executa 3 operações. Desta forma, para calcular um par de linhas, sendo uma de $\mathbf{B}^{\text{ori}}$ e outra de $\mathbf{B}^{\text{can}}$, são necessárias 27 operações. Para obter a SATD 4×4 é necessário calcular 4 linhas (108 operações) mais 4 operações do *Acumulador*, o que totaliza 112 operações.

Caso seja utilizada a FHT para o cálculo da SATD 8×8 são necessárias 576 operações, sendo 8 do *Acumulador* e as demais 568 para calcular os 8 pares de linhas dos blocos de entrada. Para cada par de linhas são feitas 71 operações, divididas em 8 operações na *Diferença* e no *Absoluto*, 48 operações para as transformadas (24 para cada *butterfly*) e 7 operações para o *Somatório*.

Ao utilizar o reúso continuam sendo necessárias 112 operações para o cálculo da SATD 4×4. Porém, para a 8×8 são necessárias apenas 256 operações, sendo 8 para o *Acumulador* e as demais 248 operações para calcular as 8 partes da SATD 8×8. Para cada parte são feitas 16 operações para os *Ajustes* (8 para cada um), 8 para o *Absoluto* e 7 para o *Somatório*. Desta forma, ao utilizar o reaproveitamento dos cálculos das SATDs 4×4, a SATD 8×8 precisa de apenas 256 operações, economizando 320 operações em relação ao método sem o reaproveitamento.

5 MÉTODO

Inicialmente foi desenvolvido um *software* baseado no modelo implementado por Bräscher (2016) para validação da arquitetura proposta. Para isto, foram utilizadas as linguagens C++ para desenvolver o *software* e *Verilog* para descrever a arquitetura. O cálculo das quatro SATDs dos blocos 4×4 e a SATD do bloco 8×8 são feitos a partir de um bloco 8×8. Os blocos utilizados e as SATDs calculadas são salvas em arquivos.

Para fazer a descrição em *Verilog* e o *testbench* da arquitetura foi utilizado o gerador desenvolvido por Seidel et al. (2016). Inicialmente, foram geradas as descrições e os *testbenchs* de uma arquitetura para blocos 4×4 e uma de blocos 8×8. Em seguida, na arquitetura para blocos 4×4 foram adicionadas as saídas extras para a segunda transformada. Já na descrição para blocos 8×8, ambas as matrizes de transformação foram substituídas por uma matriz de ajuste. Por fim, as saídas extras da segunda transformada para blocos 4×4 foram conectadas com as entradas da 8×8. Ao término das modificações, foram feitos testes utilizando os dados gerados pelo modelo de referência para detecção e correção de eventuais não conformidades.

O *testbench* gerado para a arquitetura de blocos 8×8 foi modificado para suportar a nova arquitetura. Para tal, foram adicionadas as leituras dos arquivos com os resultados das SATDs 4×4 e descrito o comportamento esperado para cada um dos estados da FSM. Os resultados das SATDs foram utilizados para verificar se os valores calculados pela arquitetura estavam corretos. Caso um valor incorreto fosse detectado a simulação era interrompida e gerada uma mensagem sobre a falha.

Após finalizar o desenvolvimento do modelo de referência e a descrição da arquitetura, ambas foram testadas para verificar se o funcionamento estava dentro do esperado. Ao término desta avaliação, ambas foram sintetizadas e simuladas usando a biblioteca de células padrão de 45 nm da TSMC e período de relógio de 15,625 *ns*, a mesma configuração utilizada por Bräscher (2016). Para a síntese lógica foi utilizada a ferramenta DC® (SYNOPSYS, 2016a) em modo topográfico. A síntese lógica foi executada para se obter estimativas de área e potência da arquitetura.

Para todas as sínteses, os atrasos de entrada e saída foram limitados a 60% do período de relógio. A capacitância máxima das entradas foram configuradas para 10 vezes o valor da entrada de uma porta AND de duas entradas. Já as saídas foram configuradas para 30 vezes a saída de uma porta lógica AND de duas entradas. Também, foi utilizado o valor de 32 milhões de blocos 4×4 por segundo como *throughput*, o dobro de Walter e Bampi (2011). Para os blocos 8×8, o *throughput* foi de 8 milhões de blocos por

segundo.

Para gerar os arquivos com os dados realistas, o HM foi modificado de forma que pudesse salvar os blocos candidatos e referências de tamanho 8×8 em arquivos. O arquivo de configuração utilizado foi o `encoder_lowdelay_P_main.cfg` com QP 32. Para cada vídeo foram sorteadas aleatoriamente 1% das janelas de buscas dos primeiros cinco quadros para serem salvas.

Foram feitas 4 codificações, usando o HM modificado, com dados realistas obtidos a partir das sequências de vídeos apresentados na Tabela 2. Estes vídeos são os mesmos utilizados por Soares et al. (2016) e cada uma faz parte de uma classe definida nas Condições Comuns de Teste - *Common Test Conditions* (CTC)(BOSSEN, 2012).

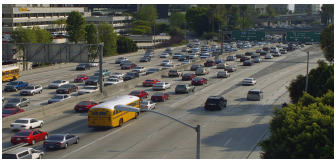
Tabela 2: Características dos vídeos avaliados. Nas duas últimas colunas são apresentados os tempos para codificar 5 quadros e 1 segundo dos vídeos utilizando o HM. Vídeos utilizados por Soares et al. (2016).

Características					Tempo exec. no HM	
Nome do Vídeo	**Resolução**	**NQ**[1]	**FPS**	**Classe**	**5 Quadros**	**1 s de Vídeo**
Traffic	2560x1600	150	30	A	256m 40s	25h 40m
BQTerrace	1920x1080	600	60	B	130m	26h
RaceHorses	832x480	300	30	C	27m 53s	2h 48m
BlowingBubbles	416x240	500	50	D	6m 15s	1h

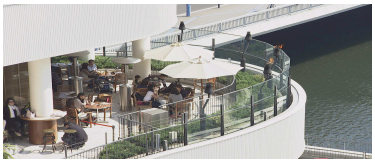
[1]Número de quadros do vídeo.

O primeiro vídeo, *Traffic*, mostra uma avenida com veículos passando (Figura 12a) e tem duração de cinco segundos. *BQTerrace* mostra um terraço com um rio ao fundo (Figura 12b). O vídeo *RaceHorses* mostra uma corrida de cavalos (Figura 12c). O último vídeo é chamado de *BlowingBubbles* (Figura 12d) e mostra duas crianças brincando com bolhas de sabão.

A partir das *netlists* obtidas na síntese lógica foram realizadas simulações com o *Synopsys*® *Verilog Compiler Simulator* (VCS®) versão L-2016.03-SP1(SYNOPSYS, 2016b) para validar as arquiteturas sintetizadas. Para a simulação foram utilizados 5 conjuntos de dados, sendo um com dados gerados de forma aleatória e os demais gerados pela HM. A arquitetura foi simulada com dados realistas para obter as atividades de chaveamento e com isso estimativas de potência mais próximas da realidade. Após sintetizar e simular a arquitetura proposta, os resultados foram comparados com os resultados apresentados nos artigos das arquiteturas do estado da arte.



(a) *Traffic*



(b) *BQTerrace*

(c) *RaceHorses*

(d) *BlowingBubbles*

Figura 12: Exemplo de quadros dos vídeos utilizados na simulação da arquitetura. Adaptado de Bossen (2012).

5.1 LIMITAÇÕES

No decorrer do desenvolvimento deste trabalho foram encontradas limitações em relação ao espaço de armazenamento, o tempo de execução das simulações e aos trabalhos correlatos. As duas primeiras limitações impediram a simulação da arquitetura proposta com todos os blocos dos vídeos utilizados na simulação. Já a última, ocorreu pelo fato de serem utilizadas ferramentas diferentes e em muitos casos não serem feitas simulações.

A limitação do número de janelas de busca utilizadas ocorreu por dois motivos: o espaço necessário para armazenar os arquivos gerados e o tempo necessário para executar a codificação. Por exemplo, para o vídeo *Traffic* (2560×1600) se todos os quadros fossem utilizados na simulação seriam necessários 600 GB. Tal espaço seria dividido em 300 GB para o arquivo com os quadros originais e 300 GB para o arquivo com os candidatos, uma vez que para os primeiros cinco quadros cada arquivo ocupou 10 GB. Outro ponto relevante foi o tempo necessário para executar cada codificação, que pode variar de alguns minutos até mesmo dias, como pode ser visto nas duas últimas colunas da Tabela 2.

Na Tabela 3 é apresentado o número total de blocos salvos pela HM e também o total de blocos que foram simulados. A simulação com dados aleatórios utilizou todos os 81920 blocos gerados. Já as simulações com dados realistas não utilizaram todos os blocos devido à grande demanda de tempo para a execução. Para o vídeo *Traffic* foram utilizados 0,02% do total de blo-

cos salvos. Já para o *BQTerrace* foram simulados 0,09% dos. *RaceHorses* e *BlowingBubbles* ultrapassaram a marca de 1 milhão de blocos simulados e foram os com o maior percentual de blocos simulados, sendo respectivamente 0,86% e 4,32% do total.

Tabela 3: Número total de blocos salvos pela HM e o número total de blocos simulados.

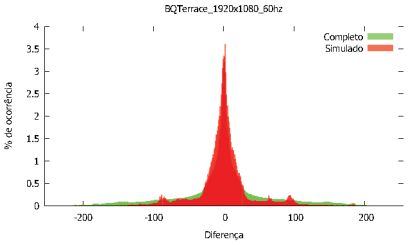
Vídeo	Blocos salvos	Blocos simulados
Traffic	1663762464	278528
BQTerrace	854809480	803840
RaceHorses	170198032	1456128
BlowingBubbles	37314296	1611776

Os resultados obtidos neste trabalho não podem ser comparados de forma direta com as arquiteturas dos trabalhos correlatos. A comparação somente seria justa se todas as arquiteturas fossem sintetizadas e simuladas utilizando as mesmas ferramentas de síntese e simulação e os mesmos conjuntos de dados, o que não ocorre.

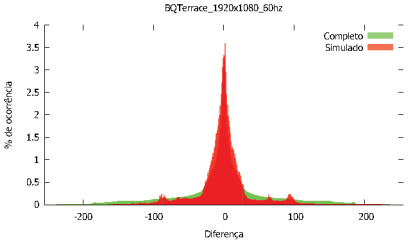
A seguir são apresentados os gráficos com o percentual de ocorrência das diferenças entre os píxeis dos blocos candidatos e originais para todos os vídeos utilizados na simulação. Estes gráficos são apresentados para mostrar que os dados utilizados na simulação tem distribuição de resíduos (diferenças) similar aos arquivos completos. Cada gráfico apresenta as ocorrências para o arquivo completo e para todos os blocos utilizados na simulação da arquitetura. O eixo das diferenças varia de -255 até 255, que são os valores possíveis para a diferença entre dois píxeis.

Os vídeos *Traffic* (Figura 13a) e *BQTerrace* (Figura 13b) possuem a maior diferença entre as regiões das diferenças dos blocos simulados e do arquivo completo. Isso pode ter ocorrido devido a estes dois vídeos terem o menor número de blocos simulados, 278528 e 803840, respectivamente. Tal fato aponta para a necessidade de fazer simulação com mais blocos.

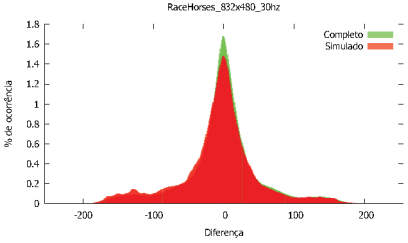
RaceHorses (Figura 13c) possuem alto grau de semelhança entre os gráficos, contudo apresentam alguns pontos discrepantes. O vídeo *Blowing Bubbles* (Figura 13d) apresenta uma grande semelhança entre os gráficos para o vídeo completo e para o que foi simulado. Nota-se que para todos, tanto para o vídeo completo quanto para o simulado, as diferenças se concentram próximo de zero.



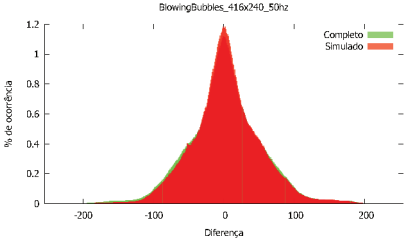
(a) *Traffic*.



(b) *BQTerrace*.



(c) *RaceHorses*



(d) *BlowingBubbles*

Figura 13: Distribuição das diferenças dos píxeis dos vídeos utilizados neste trabalho.

6 RESULTADOS E DISCUSSÃO

A partir do método descrito no Capítulo 5, foram obtidos os resultados apresentados a seguir.

A área obtida após fazer a síntese da arquitetura proposta foi de 14981 μm^2. A Tabela 4 apresenta os resultados estimados de potência. A simulação utilizando dados aleatórios obteve os maiores resultados para as potências. O vídeo *BQTerrace* obteve o menor valor para potência estática (154,3 μW). Esta diferença entre as potências estáticas ocorre pois o cálculo da potência estática dependente da atividade de chaveamento, como pode ser visto na biblioteca utilizada (TSMC, 2011). Ao considerar apenas as simulações com dados realistas, o vídeo *BlowingBubbles* obteve 156,1 μW para potência estática, sendo este o maior valor. Os maiores valores de potência dinâmica e total foram respectivamente 1163,4 μW e 1319,4 μW, obtidos pelo vídeo *RaceHorses*.

Tabela 4: Resultados de potência por vídeo após a síntese e simulação da arquitetura. A primeira linha apresenta os resultados da síntese, porém sem simulação, o que assume uma atividade de chaveamento de 10% (SYNOPSYS, 2016c). Na segunda linha é reportado os resultados da simulação com os dados aleatórios, nas quatro linhas seguintes estão os valores obtidos com as simulações com dados realistas. Nas últimas duas linhas são apresentadas a média (μ) e desvio padrão (σ) para potência e energia considerando apenas os resultados da síntese após a simulação com os dados realistas.

Dados	Potência (μW)		
	Estática	Dinâmica	Total
Sem Simulação	143,8	540,3	684,2
Dados aleatórios	157,5	1176,1	1333,6
Traffic	155,8	1027,7	1183,5
BQTerrace	154,3	1023,5	1177,8
RaceHorses	156,0	1163,4	1319,4
BlowingBubbles	156,1	1130,4	1286,5
Média (μ)	**155,6**	**1086,3**	**1241,8**
Desvio Padrão (σ)	**0,9**	**71,3**	**71,9**

Ao comparar o valor de potência total obtido para a síntese com a média dos resultados das simulações com dados realistas, nota-se que a diferença é de 44,9%. Isso dá indícios de que apenas utilizar os valores obtidos pela síntese sem simulação para obter a energia e potência não é o mais in-

dicado. Desta forma, simular a arquitetura com dados realistas e sintetizar considerando a atividade de chaveamento se mostra necessária para obter valores mais precisos para potência. Ainda, ao analisar as duas últimas linhas da Tabela 4, nota-se que os valores de desvio padrão são pequenos, o que indica que os valores são próximos.

Para o cálculo da energia é utilizada a Equação 6.1, em que $\text{Energia}_{\text{SATD}}$ é a energia para cada $\text{SATD}_{8\times8}$ calculada, Potência_{tot} é a potência total, #Ciclos é o número de ciclos para calcular cada SATD e Período é o tempo de relógio entre duas subidas de *clock*.

$$\text{Energia}_{\text{SATD}} = \text{Potência}_{tot} \times \#\text{Ciclos} \times \text{Período} \tag{6.1}$$

Para a arquitetura proposta não é calculada a energia para as SATDs dos blocos 4×4, pois este custo já está incluso no cálculo da energia para blocos 8×8. Os ciclos necessários para o preenchimento inicial e esvaziar a arquitetura não são considerados nos cálculos de energia, dado que estes ciclos não influenciam de forma significativa no resultado final. Para calcular uma SATD 8×8 são necessários oito ciclos, conforme explicado na Subseção 4.2.2, e o período de relógio é de 15,625 *ns*.

A Figura 14 apresenta o gráfico para energia. Nota-se que existe uma variação para as simulações com dados realistas, indicando que o conteúdo dos vídeos influencia no valor de energia.

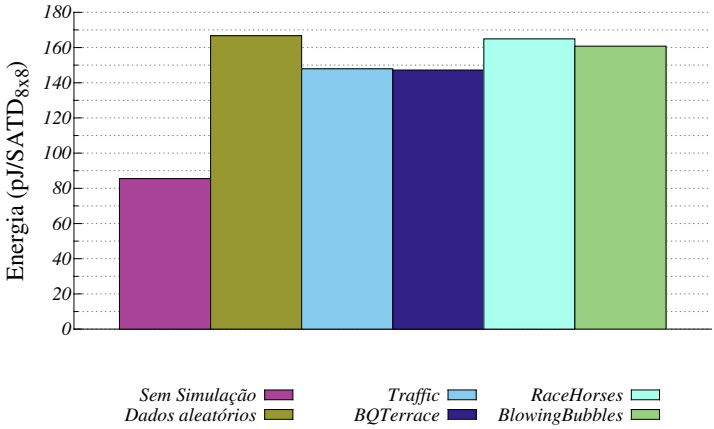


Figura 14: Resultados de energia para a síntese e as simulações.

6.1 COMPARAÇÃO COM TRABALHOS CORRELATOS

Na Figura 15 são apresentadas as comparações de área dos trabalhos correlatos, em que o valor de base utilizado foi o obtido pela arquitetura proposta neste trabalho.

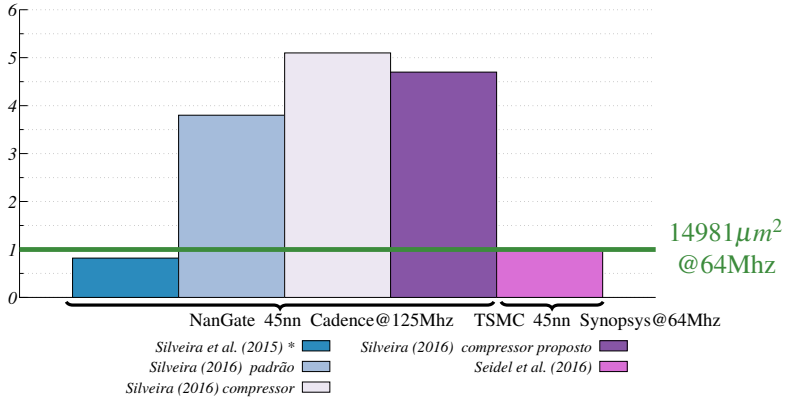


Figura 15: Comparação de área, utilizando como base o valor da arquitetura proposta neste trabalho. A arquitetura de Silveira et al. (2015) calcula a SATD apenas para blocos 8×8.

A arquitetura proposta por Silveira et al. (2015) ocupa uma área de 18,35% menor que a proposta neste trabalho. Porém, a arquitetura de Silveira et al. (2015) calcula somente blocos 8×8 e sintetizou a sua arquitetura com a biblioteca NanGate (2015) de 45 nm na ferramenta Cadence RTL Compiler tool (CADENCE, 2015).

Silveira (2016) propôs três arquiteturas para SATD, sendo uma com somadores padrão da ferramenta, uma com somadores e subtratores compressores e a última com somadores/compressores propostos no próprio trabalho. O resultado de área obtido para a primeira arquitetura foi de 57074 μm^2, já para a segunda foi de 76190 μm^2 e a última foi de 70321 μm^2. A arquitetura proposta neste trabalho ocupou 74%, 80% e 79% menos área que as arquiteturas de Silveira (2016). As configurações utilizadas na síntese das três arquiteturas propostas por Silveira (2016) são as mesmas de Silveira et al. (2015). Ao fazer a comparação entre os resultados deste trabalho com os dois anteriores, deve-se considerar que as ferramentas utilizadas são de fornecedores diferentes e cada ferramenta possui as suas peculiaridades, o que pode gerar resultados divergentes.

Ao comparar com as arquiteturas para 8×8 e 4×4 propostas por Seidel et al. (2016), a arquitetura deste trabalho apresentou uma redução de 2,44% na área. Já, Seidel et al. (2016) fizeram a síntese das arquiteturas com a ferramenta DC® e a biblioteca de células padrão de 45 nm da TSMC para dois cenários: NN e LH. Para uma comparação justa, foram utilizados os resultados NN, uma vez que foram obtidas com as mesmas configurações do presente trabalho.

Não foi possível fazer a comparação de energia com Silveira et al. (2015) pois sua arquitetura faz somente o cálculo de blocos 8×8. De maneira similar, não é possível comparar com Soares et al. (2016) pois é calculada somente a SATD para blocos 4×4. Também não foram feitas comparações de energia com o trabalho de Silveira (2016), uma vez que não foram disponibilizados os valores de energia. Além disso, não é apresentada a quantidade de ciclos para o cálculo da SATD na arquitetura para múltiplos tamanhos de HT. Sem esta informação não é possível aplicar a Equação 6.1 e obter os valores de energia.

No trabalho de Seidel et al. (2016) foram reportados 16,4 $pJ/$SATD para 4×4 e 60,5 $pJ/$SATD para 8×8, totalizando 125,9 $pJ/$SATD. A arquitetura proposta no presente trabalho é 32% mais energeticamente eficiente que as arquiteturas de Seidel et al. (2016). Já em Seidel, Güntzel e Agostini (2016), a arquitetura para 4×4 obteve 8,1 $pJ/$SATD e para 8×8 161 $pJ/$SATD totalizando 193,8 $pJ/$SATD para calcular quatro $\text{SATDs}_{4\times4}$ e uma $\text{SATD}_{8\times8}$. Ou seja, a arquitetura do presente trabalho é 55% mais energeticamente eficiente que as de Seidel, Güntzel e Agostini (2016). Ambas as comparações são relativas ao valor de energia obtido sem simulação, pois tais trabalhos não fazem simulação.

7 CONCLUSÕES E TRABALHOS FUTUROS

Neste trabalho foi proposto, demonstrado e apresentado como fazer o reúso de cálculos no contexto da SATD. Em seguida foi projetada e desenvolvida uma arquitetura para a ME baseada na SATD com reúso de cálculos. Os tamanhos de blocos suportados pela arquitetura são aqueles implementados no HM: 4×4 e 8×8. A arquitetura foi sintetizada e simulada com ferramentas de uso industrial. Foram utilizados cinco conjuntos de dados para simulação, um gerado a partir de dados aleatórios e quatro a partir de sequências de vídeos.

A área ocupada pela arquitetura proposta ficou próxima das arquiteturas do estado da arte. Ao examinar os resultados de potência e energia obtidos na síntese sem simulação, observa-se que estes valores são aproximadamente a metade dos obtidos após as simulações. A única exceção é a potência estática que os valores foram próximos. Desta forma pode-se notar que os resultados da síntese sem a simulação são estimativas otimistas e que se distanciam dos valores reais, no contexto da codificação de vídeo. Uma vez que o número de candidatos utilizados nas simulações pode interferir nos resultados obtidos.

Por conta de restrições de tempo de execução e armazenamento, neste trabalho não foram utilizados todos os candidatos dos vídeos. Contudo, sempre que possível, deve-se utilizar todos os candidatos dos vídeos selecionados para simulação da arquitetura, uma vez que dos trabalhos correlatos, apenas dois realizam a simulação com dados realistas. Contudo, não foi possível comparar o consumo energético desses trabalhos com os obtidos pela arquitetura proposta. Isso porque em um não é fornecido tal valor e faltam informações para poder calcular. No outro, os resultados são apenas para SATD 4×4. Os demais trabalhos não fazem a simulação das arquiteturas e prejudicando assim a comparação com este trabalho.

Quando comparada com as arquiteturas do estado da arte, a arquitetura proposta neste trabalho conseguiu reduzir o consumo energético sem aumentar a área ocupada. Desta forma, utilizar a SATD com reúso de cálculos se mostra vantajosa quando existe a necessidade do cálculo de múltiplos tamanhos de blocos.

7.1 TRABALHOS FUTUROS

Pode-se fazer a síntese física da arquitetura proposta. A aplicação de técnicas *low-power* também é uma opção para melhorar a eficiência energé-

tica.

Além disso, pode ser alterada a arquitetura para suportar tamanhos de blocos não quadrados, por exemplo 4×8 e 8×4. Adicionar tal característica exigiria mudanças na estrutura da arquitetura. Outra opção é o uso de implementações alternativas para os componentes da arquitetura, como utilizar somador/compressor no lugar dos somadores convencionais.

Ainda, pode-se explorar a possibilidade do uso da TE no reúso de cálculos da SATD junto ao ajuste dos valores da transformada do bloco menor para o bloco maior. Pode ser analisado o impacto nos resultados de potência e energia da síntese com atividades de chaveamento geradas a partir de simulações com dados aleatórios. Finalmente, o impacto do valor do QP nos valores da síntese após a simulação também pode ser avaliado.

REFERÊNCIAS

AGAIAN, S. et al. **Hadamard Transforms**. [S.l.]: SPIE Press, 2011. (SPIE Press Monograph Vol. PM207).

AHMED, N.; NATARAJAN, T.; RAO, K. R. Discrete cosine transform. **IEEE Trans. on Computers**, C-23, n. 1, p. 90–93, jan 1974.

ANSHI, C.; DI, L.; RENZHONG, Z. A research on fast Hadamard transform (FHT) digital systems. **TENCON '93. Proceedings. Computer, Communication, Control and Power Engineering.1993 IEEE Region 10 Conference on**, v. 3, p. 541–545 vol.3, out 1993.

BOSSEN, F. **Common test conditions and software reference configurations**. Shanghai, out 2012.

BOSSEN, F. et al. HEVC complexity and implementation analysis. **IEEE Trans. Circuits Syst. Video Technol.**, IEEE, v. 22, n. 12, p. 1685–1696, 2012.

BRÄSCHER, A. B. Trabalho de Conclusão de Curso, **Arquiteturas Energeticamente Eficientes para SATD com Tamanho de Bloco Varíavel no HEVC**. 2016.

CADENCE. **Cadence Encounter RTL Compiler v.8.10**. [S.l.], 2015. Último acesso em 10/01/2017. Disponível em: <www.cadence.com>.

CANCELLIER, L. H. et al. Exploring optimized Hadamard methods to design energy-efficient SATD architectures. **Journal of Integrated Circuits and Systems**, v. 10, n. 2, p. 113–122, Ago 2015.

CHAKRABARTI, I.; BATTA, K. N. S.; CHATTERJEE, S. K. **Motion Estimation for Video Coding: Efficient Algorithms and Architectures**. 1. ed. [S.l.]: Springer International Publishing, 2015. (Studies in Computational Intelligence 590).

DOMINGES JR, J. S. et al. High throughput 4x4 and 8x8 SATD similarity criteria architectures for video coding applications. In: **2011 VII Designer Forum (DF)**. Còrdoba, Argentina: [s.n.], 2011. p. 115.

GHANBARI, M. **Standard Codecs Image Compression to Advanced Video Coding**. [S.l.]: The Institution of Engineering and Technology, 2003. (IET Telecommunications Series).

ITU-T. **H.264 Corrigendum 1**. jan 2009.

JOU, F. **Method for fast SATD estimation**. [S.l.], aug 2007. US Patent App. 11/360,552. Último acesso em 10/01/2017. Disponível em: <https://www.google.com/patents/US20070198622>.

KIM, J.; PARK, T. A novel VLSI architecture for full-search variable block-size motion estimation. **IEEE Transactions on Consumer Electronics**, v. 55, n. 2, p. 728–733, May 2009.

LI, Y. et al. HEVC fast FME algorithm using IME RD-costs based error surface fitting scheme. In: **2016 Visual Communications and Image Processing (VCIP)**. Chengdu, China: [s.n.], 2016. p. 1–4.

LIU, J. et al. A full-mode FME VLSI architecture based on 8x8/4x4 adaptive Hadamard transform for QFHD H.264/AVC encoder. In: **2011 IEEE/IFIP 19th International Conference on VLSI and System-on-Chip**. Kowloon, Hong Kong: [s.n.], 2011. p. 434–439.

LIU, Z. et al. Register length analysis and VLSI optimization of VBS Hadamard transform in H.264/AVC. **IEEE Trans. Circuits Syst. Video Technol.**, v. 21, n. 5, p. 601–610, May 2011.

MANOEL, E. T. M. **Codificação de Vídeo H.264 - Estudo de Codificação Mista de Macroblocos**. Dissertação (Trabalho de Conclusão de Curso) — UFSC, 2007.

NANGATE. **NanGate 45nm Open Cell Library**. [S.l.], 2015. Último acesso em 10/01/2017. Disponível em: <http://www.nangate.com/?page_id=22>.

PORTO, M. et al. Design space exploration on the H.264 4x4 Hadamard transform. In: **NORCHIP Conference, 2005. 23rd**. Oulu, Finland: [s.n.], 2005. p. 188–191.

RICHARDSON, I. E. G. **Video codec design: developing image and video compression systems**. [S.l.]: John Wiley and Sons, 2002.

RICHARDSON, I. E. G. **H. 264 and MPEG-4 video compression: video coding for next-generation multimedia**. [S.l.]: John Wiley & Sons Inc, 2003.

SEIDEL, I. **Redução de Complexidade e Energia em Codificadores de Vídeo Digital Preservando a Eficiência de Codificação: Exploração de Propriedades das Métricas de Distorção Aplicadas no Casamento de Blocos.** Dissertação (Seminário de Andamento (doutorado)) — UFSC, 2016.

SEIDEL, I. et al. Energy-efficient SATD for beyond HEVC. In: **2016 IEEE International Symposium on Circuits and Systems (ISCAS)**. Montreal, Canada: [s.n.], 2016. p. 802–805.

SEIDEL, I.; GÜNTZEL, J. L.; AGOSTINI, L. Coarse grain partial distortion elimination for Hadamard ME in HEVC. In: **2016 IEEE International Conference on Electronics, Circuits and Systems (ICECS)**. Monte Carlo, Monaco: [s.n.], 2016. p. 704–707.

SHI, W.-H. S. in collaboration with T. K. **Matrix calculus and Kronecker product with applications and C++ programs**. [S.l.]: World Scientific, 1997.

SILVEIRA, B. **Exploração Arquitetural nas Métricas de Similaridade para Codificadores de Vídeo do Padrão HEVC.** Dissertação (Dissertação de Mestrado) — UCPel, 2016.

SILVEIRA, E. et al. SATD hardware architecture based on 8x8 Hadamard transform for HEVC encoder. In: **2015 IEEE International Conference on Electronics, Circuits, and Systems (ICECS)**. Cairo, Egypt: [s.n.], 2015. p. 576–579.

SOARES, L. B. et al. A novel pruned-based algorithm for energy-efficient SATD operation in the HEVC coding. In: **2016 29th Symposium on Integrated Circuits and Systems Design (SBCCI)**. Belo Horizonte, Brazil: [s.n.], 2016. p. 1–6.

SULLIVAN, G. et al. Overview of the high efficiency video coding (HEVC) standard. **IEEE Trans. Circuits Syst. Video Technol.**, v. 22, n. 12, p. 1649–1668, dez 2012.

SYNOPSYS. **Synopsys Design Compiler, Version L-2016.03-SP1.** [S.l.], 2016. Último acesso em 10/01/2017.

SYNOPSYS. **Synopsys VCS, Version L-2016.03-SP1.** 2016. Último acesso em 10/01/2017.

SYNOPSYS. **Synopsys's Design Compiler User Guide, Version L-2016.03-SP1.** 2016.

TSMC. **TSMC STANDARD CELL Library TCBN45GSBWPTC**. [S.l.], 2011.

VANNE, J. et al. Comparative rate-distortion-complexity analysis of HEVC and AVC video codecs. **IEEE Trans. on Circuits Syst. Video Technol.**, v. 22, n. 12, p. 1885–1898, Dec 2012.

WALTER, F.; BAMPI, S. Synthesis and comparison of low-power architectures for SAD calculation. **26th South Symposium on Microelectronics**, p. 45 – 48, Abril 2011.

ZHU, C.; XIONG, B. Transform-exempted calculation of sum of absolute Hadamard transformed differences. **IEEE Trans. Circuits Syst. Video Technol.**, v. 19, n. 8, p. 1183–1188, 2009.

APÊNDICE A – Código fonte

A.1 *SOFTWARE* DE REFERÊNCIA - HM

Listing A.1: Alterações em “TComRdCost.cpp”

```
169c169
<   m_afpDistortFunc[DF_SATDTCC] = TComRdCost::xGetHADTCC;
---
>
237,264d236
< // Setting the Distortion Parameter for Inter (ME)
< Void TComRdCost::setDistParamTCC( const TComPattern* const
    pcPatternKey, const Pel* piRefY, Int iRefStride, DistParam&
    rcDistParam, bool salvaNoHD )
< {
<   // set Original & Curr Pointer / Stride
<   rcDistParam.pOrg = pcPatternKey->getROIY();
<   rcDistParam.pCur = piRefY;
<
<   rcDistParam.iStrideOrg = pcPatternKey->getPatternLStride();
<   rcDistParam.iStrideCur = iRefStride;
<
<   // set Block Width / Height
<   rcDistParam.iCols    = pcPatternKey->getROIYWidth();
<   rcDistParam.iRows    = pcPatternKey->getROIYHeight();
<
<   if (rcDistParam.iCols == 8 && rcDistParam.iRows == 8 &&
    salvaNoHD)
<   {
<       // verificar se vai gravar no hd ou nÃ£o
<     rcDistParam.DistFunc = m_afpDistortFunc[DF_SATDTCC ];
<   } else {
<     rcDistParam.DistFunc=m_afpDistortFunc[DF_HADS];
<   }
<
<   rcDistParam.m_maximumDistortionForEarlyExit = std::
    numeric_limits<Distortion>::max();
<
<   // initialize
<   rcDistParam.iSubShift  = 0;
< }
<
1853,1975d1824
<
< //
< //
< //
< Distortion TComRdCost::xGetHADTCC( DistParam* pcDtParam )
< {
<   if ( pcDtParam->bApplyWeight )
<   {
<     return TComRdCostWeightPrediction::xGetHADsw( pcDtParam );
<   }
<   const Pel* piOrg       = pcDtParam->pOrg;
<   const Pel* piCur       = pcDtParam->pCur;
<   const Int  iRows       = pcDtParam->iRows;
<   const Int  iCols       = pcDtParam->iCols;
<   const Int  iStrideCur  = pcDtParam->iStrideCur;
<   const Int  iStrideOrg  = pcDtParam->iStrideOrg;
<   const Int  iStep       = pcDtParam->iStep;
```

```
52 <
53 <   Int  x, y;
54 <
55 <   Distortion uiSum = 0;
56 <
57 <   if( ( iRows % 8 == 0) && (iCols % 8 == 0) )
58 <   {
59 <     Int  iOffsetOrg = iStrideOrg<<3;
60 <     Int  iOffsetCur = iStrideCur<<3;
61 <     for ( y=0; y<iRows; y+= 8 )
62 <     {
63 <       for ( x=0; x<iCols; x+= 8 )
64 <       {
65 <         uiSum += xCalcHADs8x8 ( &piOrg[x], &piCur[x*iStep],
       iStrideOrg, iStrideCur, iStep
66 < #if VECTOR_CODING__DISTORTION_CALCULATIONS && (
       RExt__HIGH_BIT_DEPTH_SUPPORT==0)
67 <           , pcDtParam->bitDepth
68 < #endif
69 <                                );
70 <         // /*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*
71 <         // printar o bloco aqui
72 <         printTCC( &piOrg[x], &piCur[x*iStep], iStrideOrg,
       iStrideCur, iStep);
73 <       }
74 <       piOrg += iOffsetOrg;
75 <       piCur += iOffsetCur;
76 <     }
77 <   }
78 <   else if( ( iRows % 4 == 0) && (iCols % 4 == 0) )
79 <   {
80 <     Int  iOffsetOrg = iStrideOrg<<2;
81 <     Int  iOffsetCur = iStrideCur<<2;
82 <
83 <     for ( y=0; y<iRows; y+= 4 )
84 <     {
85 <       for ( x=0; x<iCols; x+= 4 )
86 <       {
87 <         uiSum += xCalcHADs4x4( &piOrg[x], &piCur[x*iStep],
       iStrideOrg, iStrideCur, iStep );
88 <       }
89 <       piOrg += iOffsetOrg;
90 <       piCur += iOffsetCur;
91 <     }
92 <   }
93 <   else if( ( iRows % 2 == 0) && (iCols % 2 == 0) )
94 <   {
95 <     Int  iOffsetOrg = iStrideOrg<<1;
96 <     Int  iOffsetCur = iStrideCur<<1;
97 <     for ( y=0; y<iRows; y+=2 )
98 <     {
99 <       for ( x=0; x<iCols; x+=2 )
100 <       {
101 <         uiSum += xCalcHADs2x2( &piOrg[x], &piCur[x*iStep],
       iStrideOrg, iStrideCur, iStep );
102 <       }
103 <       piOrg += iOffsetOrg;
104 <       piCur += iOffsetCur;
105 <     }
106 <   }
107 <   else
```

```
108 <   {
109 <     assert(false);
110 <   }
111 <
112 <   return ( uiSum >> DISTORTION_PRECISION_ADJUSTMENT(pcDtParam->
        bitDepth-8) );
113 < }
114 <
115 <
116 <
117 < void TComRdCost::printTCC( const Pel *piOrg, const Pel *piCur, Int
      iStrideOrg, Int iStrideCur, Int iStep)
118 < {
119 <   FILE *ori = fopen("ori.txt", "a");
120 <   FILE *can = fopen("can.txt", "a");
121 <   if (ori == NULL || can == NULL)
122 <   {
123 <     printf("Error opening file!\n");
124 <     exit(1);
125 <   }
126 <
127 <   Int k;
128 <   assert( iStep == 1 );
129 <   for( k = 0; k < 64; k += 8 )
130 <   {
131 <     fprintf(ori, "%x\n",piOrg[0]);
132 <     fprintf(ori, "%x\n",piOrg[1]);
133 <     fprintf(ori, "%x\n",piOrg[2]);
134 <     fprintf(ori, "%x\n",piOrg[3]);
135 <     fprintf(ori, "%x\n",piOrg[4]);
136 <     fprintf(ori, "%x\n",piOrg[5]);
137 <     fprintf(ori, "%x\n",piOrg[6]);
138 <     fprintf(ori, "%x\n",piOrg[7]);
139 <
140 <     fprintf(can, "%x\n",piCur[0]);
141 <     fprintf(can, "%x\n",piCur[1]);
142 <     fprintf(can, "%x\n",piCur[2]);
143 <     fprintf(can, "%x\n",piCur[3]);
144 <     fprintf(can, "%x\n",piCur[4]);
145 <     fprintf(can, "%x\n",piCur[5]);
146 <     fprintf(can, "%x\n",piCur[6]);
147 <     fprintf(can, "%x\n",piCur[7]);
148 <
149 <     piCur += iStrideCur;
150 <     piOrg += iStrideOrg;
151 <   }
152 <   fclose(ori);
153 <   fclose(can);
154 <   return;
155 < }
156 <
157 <
```

Listing A.2: Alterações em “TComRdCost.h”

```
1 154d153
2 <   Void    setDistParamTCC( const TComPattern* const pcPatternKey,
        const Pel* piRefY, Int iRefStride, DistParam& rcDistParam, bool
        salvaNoHD );
3 200d198
4 <   static Distortion xGetHADTCC    ( DistParam* pcDtParam );
```

```
203d200
<    static Void printTCC( const Pel *piOrg, const Pel *piCur, Int
     iStrideOrg, Int iStrideCur, Int iStep);
```

Listing A.3: Alterações em “TEncSearch.cpp”

```
3803d3802
<    bool        salvaNoHD = false;
3805,3807d3803
<    if((rand()%100) <= 1) {
<      salvaNoHD = true;
<    }
3809c3805
<    m_pcRdCost->setDistParamTCC( pcPatternKey, piRefY, iRefStride,
     m_cDistParam, salvaNoHD );
---
>    m_pcRdCost->setDistParam( pcPatternKey, piRefY, iRefStride,
     m_cDistParam );
```

Listing A.4: Alterações em “TEncSearch.h”

```
53d52
< #include <stdlib.h>
```

Listing A.5: Alterações em “TypeDef.h”

```
404d403
<    DF_SATDTCC          = 63,
```

A.2 *TESTBENCH*

Listing A.6: Código fonte do *testbench*

```
`timescale 1ns/1ps
//
//----------------------------------------
// Design Name : SATD_UV
// File Name   : tb.v
// Function    : Testbench of the SATD 4x4/8x8 with reuse of
     calculations
// Coder       : Ismael Seidel and Marcio Monteiro
//----------------------------------------
//
module testbenchSATD;
    // parameters
    parameter TOTAL_EXECS = TOTAL_EXECUTIONS_TO_RUN;
    parameter H_PERIOD = 8;
    parameter CLK_PERIOD = H_PERIOD*2;
    parameter WIDTH=8;
    parameter SIZE=3;
    parameter TWO_POW_SIZE = 8;
    parameter MEMSIZE=1024;
```

```
19      reg [7:0] original_vector [MEMSIZE*TWO_POW_SIZE*TWO_POW_SIZE
            -1:0];
20      reg [7:0] candidate_vector [MEMSIZE*TWO_POW_SIZE*TWO_POW_SIZE
            -1:0];
21      reg [19:0] results_vector4x4_0 [TOTAL_EXECS-1:0];
22      reg [19:0] results_vector4x4_1 [TOTAL_EXECS-1:0];
23      reg [19:0] results_vector4x4_2 [TOTAL_EXECS-1:0];
24      reg [19:0] results_vector4x4_3 [TOTAL_EXECS-1:0];
25      reg [19:0] results_vector [TOTAL_EXECS-1:0];
26
27      //QBT
28      reg [19:0] result8x8_m2;
29      reg [19:0] result8x8_m1;
30      reg [19:0] result0_m1;
31      reg [19:0] result1_m1;
32      reg [19:0] result2_m1;
33      reg [19:0] result3_m1;
34
35      // control signals
36      reg [31:0] i;
37      reg [31:0] reloadMemory;
38      reg [31:0] controlExecution;
39      reg [31:0] address;
40      reg clk;
41      reg reset;
42      reg enable;
43      // inputs of architecture
44      reg [7:0] original_0, original_1, original_2, original_3,
            original_4, original_5, original_6, original_7;
45      reg [7:0] candidate_0, candidate_1, candidate_2, candidate_3,
            candidate_4, candidate_5, candidate_6, candidate_7;
46      // signal to indicate when the SATD's are done
47      wire done, doneA, doneB;
48      // regitster to store the result
49      wire [19:0] satd8x8;
50      wire [15:0] satd4x4A, satd4x4B;
51      // design under verification
52      SATD SATD_UV(
53          .clock (clk),//1
54          .reset (reset),//2
55          .enable (enable),//3
56          .original_0 (original_0),//4
57          .original_1 (original_1),//5
58          .original_2 (original_2),//6
59          .original_3 (original_3),//7
60          .original_4 (original_4),//8
61          .original_5 (original_5),//9
62          .original_6 (original_6),//10
63          .original_7 (original_7),//11
64          .candidate_0 (candidate_0),//12
65          .candidate_1 (candidate_1),//13
66          .candidate_2 (candidate_2),//14
67          .candidate_3 (candidate_3),//15
68          .candidate_4 (candidate_4),//16
69          .candidate_5 (candidate_5),//17
70          .candidate_6 (candidate_6),//18
71          .candidate_7 (candidate_7),//19
72          .satd8x8 (satd8x8),//20
73          .satd4x4A (satd4x4A),//21
74          .satd4x4B (satd4x4B),//22
75          .done (done),//23
```

```
76          .doneA (doneA),//24
77          .doneB (doneB)//25
78      );
79      // files with the tests vector
80      initial $readmemh("PATH_OF_VECTORS/canmain.txt",
            candidate_vector,0,(MEMSIZE*TWO_POW_SIZE*TWO_POW_SIZE - 1));
81      initial $readmemh("PATH_OF_VECTORS/orimain.txt", original_vector
            ,0,(MEMSIZE*TWO_POW_SIZE*TWO_POW_SIZE - 1));
82      initial $readmemh("PATH_OF_VECTORS/results_8x8.txt",
            results_vector,0,(MEMSIZE*TWO_POW_SIZE*TWO_POW_SIZE - 1));
83      initial $readmemh("PATH_OF_VECTORS/results_4x4_0.txt",
            results_vector4x4_0,0,(MEMSIZE*TWO_POW_SIZE*TWO_POW_SIZE -
            1));
84      initial $readmemh("PATH_OF_VECTORS/results_4x4_1.txt",
            results_vector4x4_1,0,(MEMSIZE*TWO_POW_SIZE*TWO_POW_SIZE -
            1));
85      initial $readmemh("PATH_OF_VECTORS/results_4x4_2.txt",
            results_vector4x4_2,0,(MEMSIZE*TWO_POW_SIZE*TWO_POW_SIZE -
            1));
86      initial $readmemh("PATH_OF_VECTORS/results_4x4_3.txt",
            results_vector4x4_3,0,(MEMSIZE*TWO_POW_SIZE*TWO_POW_SIZE -
            1));
87      always
88          #H_PERIOD clk = !clk;
89      initial begin
90          $set_gate_level_monitoring("on");
91          $set_toggle_region(SATD_UV);
92          $toggle_start;
93          i=0;
94          reloadMemory=0;
95          controlExecution=0;
96          clk = 0;
97          reset = 0;
98          enable = 0;
99          original_0 = 0;
100         original_1 = 0;
101         original_2 = 0;
102         original_3 = 0;
103         original_4 = 0;
104         original_5 = 0;
105         original_6 = 0;
106         original_7 = 0;
107         candidate_0 = 0;
108         candidate_1 = 0;
109         candidate_2 = 0;
110         candidate_3 = 0;
111         candidate_4 = 0;
112         candidate_5 = 0;
113         candidate_6 = 0;
114         candidate_7 = 0;
115
116         //QBT
117         result8x8_m2 = 0;
118         result8x8_m1 = 0;
119         result0_m1 = 0;
120         result1_m1 = 0;
121         result2_m1 = 0;
122         result3_m1 = 0;
123
124         #CLK_PERIOD
125             @ (negedge clk);
```

```
126                 reset = 1;
127             @ (negedge clk);
128                 reset = 0;
129         #CLK_PERIOD
130             address = 0;
131         #CLK_PERIOD enable = 1;
132         #CLK_PERIOD;//state INITIALIZE_0
133             original_0 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 0) + address];
134             original_1 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 1) + address];
135             original_2 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 2) + address];
136             original_3 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 3) + address];
137             original_4 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 4) + address];
138             original_5 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 5) + address];
139             original_6 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 6) + address];
140             original_7 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 7) + address];
141             candidate_0 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 0) + address];
142             candidate_1 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 1) + address];
143             candidate_2 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 2) + address];
144             candidate_3 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 3) + address];
145             candidate_4 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 4) + address];
146             candidate_5 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 5) + address];
147             candidate_6 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 6) + address];
148             candidate_7 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 7) + address];
149             address=address+1;
150         #CLK_PERIOD;//state INITIALIZE_1
151             original_0 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 0) + address];
152             original_1 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 1) + address];
153             original_2 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 2) + address];
154             original_3 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 3) + address];
155             original_4 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 4) + address];
156             original_5 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 5) + address];
157             original_6 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 6) + address];
158             original_7 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 7) + address];
159             candidate_0 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 0) + address];
160             candidate_1 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 1) + address];
```

```
161             candidate_2 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 2) + address];
162             candidate_3 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 3) + address];
163             candidate_4 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 4) + address];
164             candidate_5 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 5) + address];
165             candidate_6 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 6) + address];
166             candidate_7 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 7) + address];
167             address=address+1;
168         #CLK_PERIOD;//state INITIALIZE_2
169             original_0 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 0) + address];
170             original_1 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 1) + address];
171             original_2 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 2) + address];
172             original_3 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 3) + address];
173             original_4 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 4) + address];
174             original_5 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 5) + address];
175             original_6 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 6) + address];
176             original_7 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 7) + address];
177             candidate_0 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 0) + address];
178             candidate_1 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 1) + address];
179             candidate_2 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 2) + address];
180             candidate_3 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 3) + address];
181             candidate_4 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 4) + address];
182             candidate_5 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 5) + address];
183             candidate_6 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 6) + address];
184             candidate_7 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 7) + address];
185             address=address+1;
186         #CLK_PERIOD;//state INITIALIZE_3
187             original_0 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 0) + address];
188             original_1 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 1) + address];
189             original_2 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 2) + address];
190             original_3 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 3) + address];
191             original_4 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 4) + address];
192             original_5 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 5) + address];
```

```
193             original_6 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 6) + address];
194             original_7 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 7) + address];
195             candidate_0 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 0) + address];
196             candidate_1 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 1) + address];
197             candidate_2 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 2) + address];
198             candidate_3 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 3) + address];
199             candidate_4 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 4) + address];
200             candidate_5 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 5) + address];
201             candidate_6 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 6) + address];
202             candidate_7 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 7) + address];
203             address=address+1;
204         #CLK_PERIOD;//state INITIALIZE_4
205             original_0 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 0) + address];
206             original_1 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 1) + address];
207             original_2 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 2) + address];
208             original_3 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 3) + address];
209             original_4 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 4) + address];
210             original_5 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 5) + address];
211             original_6 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 6) + address];
212             original_7 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 7) + address];
213             candidate_0 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 0) + address];
214             candidate_1 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 1) + address];
215             candidate_2 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 2) + address];
216             candidate_3 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 3) + address];
217             candidate_4 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 4) + address];
218             candidate_5 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 5) + address];
219             candidate_6 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 6) + address];
220             candidate_7 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 7) + address];
221             address=address+1;
222         #CLK_PERIOD;//state INITIALIZE_5
223             original_0 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 0) + address];
224             original_1 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 1) + address];
```

```
225                 original_2 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 2) + address];
226                 original_3 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 3) + address];
227                 original_4 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 4) + address];
228                 original_5 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 5) + address];
229                 original_6 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 6) + address];
230                 original_7 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 7) + address];
231                 candidate_0 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 0) + address];
232                 candidate_1 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 1) + address];
233                 candidate_2 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 2) + address];
234                 candidate_3 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 3) + address];
235                 candidate_4 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 4) + address];
236                 candidate_5 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 5) + address];
237                 candidate_6 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 6) + address];
238                 candidate_7 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 7) + address];
239                 address=address+1;
240             #CLK_PERIOD;//state INITIALIZE_6
241                 original_0 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 0) + address];
242                 original_1 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 1) + address];
243                 original_2 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 2) + address];
244                 original_3 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 3) + address];
245                 original_4 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 4) + address];
246                 original_5 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 5) + address];
247                 original_6 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 6) + address];
248                 original_7 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 7) + address];
249                 candidate_0 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 0) + address];
250                 candidate_1 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 1) + address];
251                 candidate_2 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 2) + address];
252                 candidate_3 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 3) + address];
253                 candidate_4 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 4) + address];
254                 candidate_5 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 5) + address];
255                 candidate_6 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 6) + address];
```

```
256             candidate_7 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 7) + address];
257             address=address+1;
258         #CLK_PERIOD;//state INITIALIZE_7
259             original_0 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 0) + address];
260             original_1 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 1) + address];
261             original_2 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 2) + address];
262             original_3 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 3) + address];
263             original_4 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 4) + address];
264             original_5 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 5) + address];
265             original_6 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 6) + address];
266             original_7 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 7) + address];
267             candidate_0 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 0) + address];
268             candidate_1 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 1) + address];
269             candidate_2 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 2) + address];
270             candidate_3 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 3) + address];
271             candidate_4 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 4) + address];
272             candidate_5 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 5) + address];
273             candidate_6 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 6) + address];
274             candidate_7 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 7) + address];
275             address=0;
276             i=i+1;
277             controlExecution=controlExecution+1;
278         #CLK_PERIOD;//state INITIALIZE_8
279             original_0 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 0) + address];
280             original_1 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 1) + address];
281             original_2 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 2) + address];
282             original_3 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 3) + address];
283             original_4 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 4) + address];
284             original_5 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 5) + address];
285             original_6 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 6) + address];
286             original_7 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 7) + address];
287             candidate_0 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 0) + address];
288             candidate_1 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 1) + address];
```

```
289                 candidate_2 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 2) + address];
290                 candidate_3 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 3) + address];
291                 candidate_4 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 4) + address];
292                 candidate_5 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 5) + address];
293                 candidate_6 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 6) + address];
294                 candidate_7 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 7) + address];
295                 address=address+1;
296             #CLK_PERIOD;//state INITIALIZE_9
297                 original_0 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 0) + address];
298                 original_1 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 1) + address];
299                 original_2 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 2) + address];
300                 original_3 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 3) + address];
301                 original_4 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 4) + address];
302                 original_5 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 5) + address];
303                 original_6 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 6) + address];
304                 original_7 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 7) + address];
305                 candidate_0 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 0) + address];
306                 candidate_1 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 1) + address];
307                 candidate_2 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 2) + address];
308                 candidate_3 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 3) + address];
309                 candidate_4 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 4) + address];
310                 candidate_5 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 5) + address];
311                 candidate_6 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 6) + address];
312                 candidate_7 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 7) + address];
313                 address=address+1;
314                 if(results_vector4x4_0[i-1] === satd4x4A &&
                        results_vector4x4_1[i-1] === satd4x4B) begin
315                     // $display("Correct\n");
316                 end else begin
317                     $display("ERROR:\n");
318                     $display("calc1: %d\tcalc2: %d\t4x4_0: %d\t4x4_1
                            : %d\t4x4_2: %d\t4x4_3: %d\t",satd4x4A,
                            satd4x4B, results_vector4x4_0[i-1],
                            results_vector4x4_1[i-1],
                            results_vector4x4_2[i-1],
                            results_vector4x4_3[i-1]);
319                     $finish;
320                 end
321             #CLK_PERIOD;//state INITIALIZE_10
```

```
322             original_0 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 0) + address];
323             original_1 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 1) + address];
324             original_2 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 2) + address];
325             original_3 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 3) + address];
326             original_4 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 4) + address];
327             original_5 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 5) + address];
328             original_6 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 6) + address];
329             original_7 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 7) + address];
330             candidate_0 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 0) + address];
331             candidate_1 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 1) + address];
332             candidate_2 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 2) + address];
333             candidate_3 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 3) + address];
334             candidate_4 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 4) + address];
335             candidate_5 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 5) + address];
336             candidate_6 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 6) + address];
337             candidate_7 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 7) + address];
338             address=address+1;
339         #CLK_PERIOD;//state INITIALIZE_11
340             original_0 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 0) + address];
341             original_1 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 1) + address];
342             original_2 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 2) + address];
343             original_3 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 3) + address];
344             original_4 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 4) + address];
345             original_5 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 5) + address];
346             original_6 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 6) + address];
347             original_7 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 7) + address];
348             candidate_0 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 0) + address];
349             candidate_1 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 1) + address];
350             candidate_2 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 2) + address];
351             candidate_3 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 3) + address];
352             candidate_4 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 4) + address];
```

```
353                 candidate_5 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 5) + address];
354                 candidate_6 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 6) + address];
355                 candidate_7 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 7) + address];
356                 address=address+1;
357             #CLK_PERIOD;//state INITIALIZE_12
358                 original_0 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 0) + address];
359                 original_1 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 1) + address];
360                 original_2 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 2) + address];
361                 original_3 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 3) + address];
362                 original_4 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 4) + address];
363                 original_5 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 5) + address];
364                 original_6 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 6) + address];
365                 original_7 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 7) + address];
366                 candidate_0 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 0) + address];
367                 candidate_1 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 1) + address];
368                 candidate_2 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 2) + address];
369                 candidate_3 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 3) + address];
370                 candidate_4 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 4) + address];
371                 candidate_5 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 5) + address];
372                 candidate_6 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 6) + address];
373                 candidate_7 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 7) + address];
374                 address=address+1;
375             #CLK_PERIOD;//state INITIALIZE_13
376                 original_0 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 0) + address];
377                 original_1 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 1) + address];
378                 original_2 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 2) + address];
379                 original_3 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 3) + address];
380                 original_4 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 4) + address];
381                 original_5 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 5) + address];
382                 original_6 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 6) + address];
383                 original_7 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 7) + address];
384                 candidate_0 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 0) + address];
```

```
385             candidate_1 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 1) + address];
386             candidate_2 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 2) + address];
387             candidate_3 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 3) + address];
388             candidate_4 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 4) + address];
389             candidate_5 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 5) + address];
390             candidate_6 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 6) + address];
391             candidate_7 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 7) + address];
392             address=address+1;
393             if(results_vector4x4_2[i-1] === satd4x4A &&
                    results_vector4x4_3[i-1] === satd4x4B ) begin//&&
                    results_vector[i-2] === satd8x8) begin
394                 // $display("Correct\n");
395             end else begin
396                 $display("ERROR:\n");
397                 $display("i: %d\tcalc1: %d\tcalc2: %d\tcalc8x8: %d\
                        t4x4_0: %d\t4x4_1: %d\t4x4_2: %d\t4x4_3: %d\t8x8
                        : %d",i, satd4x4A, satd4x4B, satd8x8,
                        results_vector4x4_0[i-1], results_vector4x4_1[i
                        -1], results_vector4x4_2[i-1],
                        results_vector4x4_3[i-1], results_vector[i-2]);
398                 $finish;
399             end
400
401         #CLK_PERIOD;//state INITIALIZE_14
402             original_0 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 0) + address];
403             original_1 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 1) + address];
404             original_2 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 2) + address];
405             original_3 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 3) + address];
406             original_4 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 4) + address];
407             original_5 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 5) + address];
408             original_6 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 6) + address];
409             original_7 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 7) + address];
410             candidate_0 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 0) + address];
411             candidate_1 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 1) + address];
412             candidate_2 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 2) + address];
413             candidate_3 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 3) + address];
414             candidate_4 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 4) + address];
415             candidate_5 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 5) + address];
416             candidate_6 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 6) + address];
```

```
417                 candidate_7 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 7) + address];
418                 address=address+1;
419             #CLK_PERIOD;//state INITIALIZE_15
420                 original_0 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 0) + address];
421                 original_1 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 1) + address];
422                 original_2 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 2) + address];
423                 original_3 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 3) + address];
424                 original_4 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 4) + address];
425                 original_5 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 5) + address];
426                 original_6 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 6) + address];
427                 original_7 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 7) + address];
428                 candidate_0 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 0) + address];
429                 candidate_1 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 1) + address];
430                 candidate_2 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 2) + address];
431                 candidate_3 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 3) + address];
432                 candidate_4 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 4) + address];
433                 candidate_5 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 5) + address];
434                 candidate_6 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 6) + address];
435                 candidate_7 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 7) + address];
436                 address=0;
437                 i=i+1;
438                 controlExecution=controlExecution+1;
439             #CLK_PERIOD;//state INITIALIZE_16
440                 original_0 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 0) + address];
441                 original_1 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 1) + address];
442                 original_2 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 2) + address];
443                 original_3 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 3) + address];
444                 original_4 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 4) + address];
445                 original_5 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 5) + address];
446                 original_6 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 6) + address];
447                 original_7 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 7) + address];
448                 candidate_0 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 0) + address];
449                 candidate_1 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 1) + address];
```

```
450             candidate_2 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 2) + address];
451             candidate_3 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 3) + address];
452             candidate_4 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 4) + address];
453             candidate_5 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 5) + address];
454             candidate_6 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 6) + address];
455             candidate_7 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 7) + address];
456             address=address+1;
457         #CLK_PERIOD;//state INITIALIZE_17
458             original_0 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 0) + address];
459             original_1 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 1) + address];
460             original_2 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 2) + address];
461             original_3 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 3) + address];
462             original_4 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 4) + address];
463             original_5 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 5) + address];
464             original_6 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 6) + address];
465             original_7 = original_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 7) + address];
466             candidate_0 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 0) + address];
467             candidate_1 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 1) + address];
468             candidate_2 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 2) + address];
469             candidate_3 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 3) + address];
470             candidate_4 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 4) + address];
471             candidate_5 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 5) + address];
472             candidate_6 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 6) + address];
473             candidate_7 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 7) + address];
474             address=address+1;
475             if(results_vector4x4_0[i-1] === satd4x4A &&
                    results_vector4x4_1[i-1] === satd4x4B ) begin//&&
                    results_vector[i-2] === satd8x8) begin
476                 /// $display("Correct\n");
477             end else begin
478                 $display("ERROR:\n");
479                 $display("i: %d\tcalc1: %d\tcalc2: %d\tcalc8x8: %d\
                        t4x4_0: %d\t4x4_1: %d\t4x4_2: %d\t4x4_3: %d\t8x8
                        : %d",i, satd4x4A, satd4x4B, satd8x8,
                        results_vector4x4_0[i-1], results_vector4x4_1[i
                        -1], results_vector4x4_2[i-1],
                        results_vector4x4_3[i-1], results_vector[i-2]);
480                 $finish;
481             end
```

```
482             #CLK_PERIOD;//state INITIALIZE_18
483                 original_0 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 0) + address];
484                 original_1 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 1) + address];
485                 original_2 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 2) + address];
486                 original_3 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 3) + address];
487                 original_4 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 4) + address];
488                 original_5 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 5) + address];
489                 original_6 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 6) + address];
490                 original_7 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 7) + address];
491                 candidate_0 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 0) + address];
492                 candidate_1 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 1) + address];
493                 candidate_2 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 2) + address];
494                 candidate_3 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 3) + address];
495                 candidate_4 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 4) + address];
496                 candidate_5 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 5) + address];
497                 candidate_6 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 6) + address];
498                 candidate_7 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 7) + address];
499                 address=address+1;
500             #CLK_PERIOD;//state INITIALIZE_19
501                 original_0 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 0) + address];
502                 original_1 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 1) + address];
503                 original_2 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 2) + address];
504                 original_3 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 3) + address];
505                 original_4 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 4) + address];
506                 original_5 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 5) + address];
507                 original_6 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 6) + address];
508                 original_7 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 7) + address];
509                 candidate_0 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 0) + address];
510                 candidate_1 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 1) + address];
511                 candidate_2 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 2) + address];
512                 candidate_3 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 3) + address];
513                 candidate_4 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 4) + address];
```

```
514             candidate_5 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 5) + address];
515             candidate_6 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 6) + address];
516             candidate_7 = candidate_vector[(i*TWO_POW_SIZE*
                    TWO_POW_SIZE) + (TWO_POW_SIZE * 7) + address];
517             address=address+1;
518         if (controlExecution >= TOTAL_EXECS) begin
519             reloadMemory=reloadMemory+1;
520             enable = 0;
521             #CLK_PERIOD;//state PRE_TERMINATE_0
522             #CLK_PERIOD;//state PRE_TERMINATE_1
523             if(results_vector4x4_2[i-1] === satd4x4A &&
                    results_vector4x4_3[i-1] === satd4x4B &&
                    results_vector[i-2] === satd8x8) begin
524                 // $display("Correct\n");
525             end else begin
526                 $display("ERROR:\n");
527                 $display("Early finished\ncalc1: %d\tcalc2: %d\
                        tcalc8x8: %d\t4x4_0: %d\t4x4_1: %d\t4x4_2: %d\
                        t4x4_3: %d\t8x8: %d",satd4x4A, satd4x4B, satd8x8
                        , results_vector4x4_0[i-1], results_vector4x4_1[
                        i-1], results_vector4x4_2[i-1],
                        results_vector4x4_3[i-1], results_vector[i-2]);
528                 $finish;
529             end
530             i=i+1;
531             controlExecution=controlExecution+1;
532         end else begin
533             #CLK_PERIOD;//state PRE_CALCULATE_0
534                 original_0 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 0) + address];
535                 original_1 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 1) + address];
536                 original_2 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 2) + address];
537                 original_3 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 3) + address];
538                 original_4 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 4) + address];
539                 original_5 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 5) + address];
540                 original_6 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 6) + address];
541                 original_7 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 7) + address];
542                 candidate_0 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 0) + address];
543                 candidate_1 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 1) + address];
544                 candidate_2 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 2) + address];
545                 candidate_3 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 3) + address];
546                 candidate_4 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 4) + address];
547                 candidate_5 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 5) + address];
548                 candidate_6 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 6) + address];
```

```
549                 candidate_7 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 7) + address];
550                 address=address+1;
551             #CLK_PERIOD;//state PRE_CALCULATE_1
552                 original_0 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 0) + address];
553                 original_1 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 1) + address];
554                 original_2 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 2) + address];
555                 original_3 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 3) + address];
556                 original_4 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 4) + address];
557                 original_5 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 5) + address];
558                 original_6 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 6) + address];
559                 original_7 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 7) + address];
560                 candidate_0 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 0) + address];
561                 candidate_1 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 1) + address];
562                 candidate_2 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 2) + address];
563                 candidate_3 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 3) + address];
564                 candidate_4 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 4) + address];
565                 candidate_5 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 5) + address];
566                 candidate_6 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 6) + address];
567                 candidate_7 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 7) + address];
568                 address=address+1;
569                 if(results_vector4x4_2[i-1] === satd4x4A &&
                        results_vector4x4_3[i-1] === satd4x4B &&
                        results_vector[i-2] === satd8x8) begin
570                     // $display("Correct\n");
571                 end else begin
572                     $display("ERROR:\n");
573                     $display("PRE_CALCULATE_1\n controlExecution: %d
                            \ti: %d\tcalc1: %d\tcalc2: %d\tcalc8x8: %d\
                            t4x4_2: %d\t4x4_3: %d\t8x8: %d",
                            controlExecution,i,satd4x4A, satd4x4B,
                            satd8x8, results_vector4x4_2[i-1],
                            results_vector4x4_3[i-1],results_vector[i
                            -2]);
574                     $finish;
575                 end
576         end
577         while(controlExecution<TOTAL_EXECS) begin
578             #CLK_PERIOD;//state CALCULATE_2
579                 original_0 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 0) + address];
580                 original_1 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 1) + address];
581                 original_2 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 2) + address];
```

```
582                 original_3 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 3) + address];
583                 original_4 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 4) + address];
584                 original_5 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 5) + address];
585                 original_6 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 6) + address];
586                 original_7 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 7) + address];
587                 candidate_0 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 0) + address];
588                 candidate_1 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 1) + address];
589                 candidate_2 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 2) + address];
590                 candidate_3 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 3) + address];
591                 candidate_4 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 4) + address];
592                 candidate_5 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 5) + address];
593                 candidate_6 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 6) + address];
594                 candidate_7 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 7) + address];
595                 address=address+1;
596             #CLK_PERIOD;//state CALCULATE_3
597                 original_0 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 0) + address];
598                 original_1 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 1) + address];
599                 original_2 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 2) + address];
600                 original_3 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 3) + address];
601                 original_4 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 4) + address];
602                 original_5 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 5) + address];
603                 original_6 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 6) + address];
604                 original_7 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 7) + address];
605                 candidate_0 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 0) + address];
606                 candidate_1 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 1) + address];
607                 candidate_2 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 2) + address];
608                 candidate_3 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 3) + address];
609                 candidate_4 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 4) + address];
610                 candidate_5 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 5) + address];
611                 candidate_6 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 6) + address];
612                 candidate_7 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 7) + address];
613                 address=0;
```

```
614                     i=i+1;
615                     controlExecution=controlExecution+1;
616                     // Used to load more data in memory. The tools
                            limits the size of memory for data in 2GB - 1.
617                     if (i===MEMSIZE) begin
618                         result8x8_m2 = results_vector[i-2];
619                         result8x8_m1 = results_vector[i-1];
620                         result0_m1 = results_vector4x4_0[i-1];
621                         result1_m1 = results_vector4x4_1[i-1];
622                         result2_m1 = results_vector4x4_2[i-1];
623                         result3_m1 = results_vector4x4_3[i-1];
624                         i=0;
625                         $toggle_stop;
626                         $display("Reset. controlExecution: %d\n",
                                controlExecution);
627                         $toggle_report("output.saif", 0.0000000001,
                                SATD_UV);
628                         $toggle_start;
629                         $readmemh("PATH_OF_VECTORS/canmain.txt",
                                candidate_vector,controlExecution*
                                TWO_POW_SIZE*TWO_POW_SIZE,TWO_POW_SIZE*
                                TWO_POW_SIZE*(MEMSIZE+controlExecution)-1);
630                         $readmemh("PATH_OF_VECTORS/orimain.txt",
                                original_vector,controlExecution*
                                TWO_POW_SIZE*TWO_POW_SIZE,TWO_POW_SIZE*
                                TWO_POW_SIZE*(MEMSIZE+controlExecution)-1);
631                         $readmemh("PATH_OF_VECTORS/results_8x8.txt",
                                results_vector,controlExecution*TWO_POW_SIZE
                                *TWO_POW_SIZE,TWO_POW_SIZE*TWO_POW_SIZE*(
                                MEMSIZE+controlExecution)-1);
632                         $readmemh("PATH_OF_VECTORS/results_4x4_0.txt",
                                results_vector4x4_0,controlExecution*
                                TWO_POW_SIZE*TWO_POW_SIZE,TWO_POW_SIZE*
                                TWO_POW_SIZE*(MEMSIZE+controlExecution)-1);
633                         $readmemh("PATH_OF_VECTORS/results_4x4_1.txt",
                                results_vector4x4_1,controlExecution*
                                TWO_POW_SIZE*TWO_POW_SIZE,TWO_POW_SIZE*
                                TWO_POW_SIZE*(MEMSIZE+controlExecution)-1);
634                         $readmemh("PATH_OF_VECTORS/results_4x4_2.txt",
                                results_vector4x4_2,controlExecution*
                                TWO_POW_SIZE*TWO_POW_SIZE,TWO_POW_SIZE*
                                TWO_POW_SIZE*(MEMSIZE+controlExecution)-1);
635                         $readmemh("PATH_OF_VECTORS/results_4x4_3.txt",
                                results_vector4x4_3,controlExecution*
                                TWO_POW_SIZE*TWO_POW_SIZE,TWO_POW_SIZE*
                                TWO_POW_SIZE*(MEMSIZE+controlExecution)-1);
636                     end
637                 #CLK_PERIOD;//state CALCULATE_4
638                     original_0 = original_vector[(i*TWO_POW_SIZE*
                            TWO_POW_SIZE) + (TWO_POW_SIZE * 0) + address];
639                     original_1 = original_vector[(i*TWO_POW_SIZE*
                            TWO_POW_SIZE) + (TWO_POW_SIZE * 1) + address];
640                     original_2 = original_vector[(i*TWO_POW_SIZE*
                            TWO_POW_SIZE) + (TWO_POW_SIZE * 2) + address];
641                     original_3 = original_vector[(i*TWO_POW_SIZE*
                            TWO_POW_SIZE) + (TWO_POW_SIZE * 3) + address];
642                     original_4 = original_vector[(i*TWO_POW_SIZE*
                            TWO_POW_SIZE) + (TWO_POW_SIZE * 4) + address];
643                     original_5 = original_vector[(i*TWO_POW_SIZE*
                            TWO_POW_SIZE) + (TWO_POW_SIZE * 5) + address];
```

```
644                     original_6 = original_vector[(i*TWO_POW_SIZE*
                            TWO_POW_SIZE) + (TWO_POW_SIZE * 6) + address];
645                     original_7 = original_vector[(i*TWO_POW_SIZE*
                            TWO_POW_SIZE) + (TWO_POW_SIZE * 7) + address];
646                     candidate_0 = candidate_vector[(i*TWO_POW_SIZE*
                            TWO_POW_SIZE) + (TWO_POW_SIZE * 0) + address];
647                     candidate_1 = candidate_vector[(i*TWO_POW_SIZE*
                            TWO_POW_SIZE) + (TWO_POW_SIZE * 1) + address];
648                     candidate_2 = candidate_vector[(i*TWO_POW_SIZE*
                            TWO_POW_SIZE) + (TWO_POW_SIZE * 2) + address];
649                     candidate_3 = candidate_vector[(i*TWO_POW_SIZE*
                            TWO_POW_SIZE) + (TWO_POW_SIZE * 3) + address];
650                     candidate_4 = candidate_vector[(i*TWO_POW_SIZE*
                            TWO_POW_SIZE) + (TWO_POW_SIZE * 4) + address];
651                     candidate_5 = candidate_vector[(i*TWO_POW_SIZE*
                            TWO_POW_SIZE) + (TWO_POW_SIZE * 5) + address];
652                     candidate_6 = candidate_vector[(i*TWO_POW_SIZE*
                            TWO_POW_SIZE) + (TWO_POW_SIZE * 6) + address];
653                     candidate_7 = candidate_vector[(i*TWO_POW_SIZE*
                            TWO_POW_SIZE) + (TWO_POW_SIZE * 7) + address];
654                     address=address+1;
655                 #CLK_PERIOD;//state CALCULATE_5
656                     original_0 = original_vector[(i*TWO_POW_SIZE*
                            TWO_POW_SIZE) + (TWO_POW_SIZE * 0) + address];
657                     original_1 = original_vector[(i*TWO_POW_SIZE*
                            TWO_POW_SIZE) + (TWO_POW_SIZE * 1) + address];
658                     original_2 = original_vector[(i*TWO_POW_SIZE*
                            TWO_POW_SIZE) + (TWO_POW_SIZE * 2) + address];
659                     original_3 = original_vector[(i*TWO_POW_SIZE*
                            TWO_POW_SIZE) + (TWO_POW_SIZE * 3) + address];
660                     original_4 = original_vector[(i*TWO_POW_SIZE*
                            TWO_POW_SIZE) + (TWO_POW_SIZE * 4) + address];
661                     original_5 = original_vector[(i*TWO_POW_SIZE*
                            TWO_POW_SIZE) + (TWO_POW_SIZE * 5) + address];
662                     original_6 = original_vector[(i*TWO_POW_SIZE*
                            TWO_POW_SIZE) + (TWO_POW_SIZE * 6) + address];
663                     original_7 = original_vector[(i*TWO_POW_SIZE*
                            TWO_POW_SIZE) + (TWO_POW_SIZE * 7) + address];
664                     candidate_0 = candidate_vector[(i*TWO_POW_SIZE*
                            TWO_POW_SIZE) + (TWO_POW_SIZE * 0) + address];
665                     candidate_1 = candidate_vector[(i*TWO_POW_SIZE*
                            TWO_POW_SIZE) + (TWO_POW_SIZE * 1) + address];
666                     candidate_2 = candidate_vector[(i*TWO_POW_SIZE*
                            TWO_POW_SIZE) + (TWO_POW_SIZE * 2) + address];
667                     candidate_3 = candidate_vector[(i*TWO_POW_SIZE*
                            TWO_POW_SIZE) + (TWO_POW_SIZE * 3) + address];
668                     candidate_4 = candidate_vector[(i*TWO_POW_SIZE*
                            TWO_POW_SIZE) + (TWO_POW_SIZE * 4) + address];
669                     candidate_5 = candidate_vector[(i*TWO_POW_SIZE*
                            TWO_POW_SIZE) + (TWO_POW_SIZE * 5) + address];
670                     candidate_6 = candidate_vector[(i*TWO_POW_SIZE*
                            TWO_POW_SIZE) + (TWO_POW_SIZE * 6) + address];
671                     candidate_7 = candidate_vector[(i*TWO_POW_SIZE*
                            TWO_POW_SIZE) + (TWO_POW_SIZE * 7) + address];
672                     address=address+1;
673                     if(results_vector4x4_0[i-1] === satd4x4A &&
                            results_vector4x4_1[i-1] === satd4x4B) begin
674                         // $display("Correct\n");
675                     end else begin
676                         if (i == 0) begin
```

```
677                         if(result0_m1 === satd4x4A && result1_m1 ===
                                satd4x4B) begin
678                             // $display("Correct\n");
679                         end else begin
680                             $display("HOUSTON WE HAVE A PROBLEM\n");
681                             $finish;
682                         end
683                     end else begin
684                         $display("ERROR:\n");
685                         $display("controlExecution: %d\ti: %d\tcalc1
                                : %d\tcalc2: %d\tcalc8x8: %d\t4x4_2: %d\
                                t4x4_3: %d\t8x8: %d",controlExecution,i,
                                satd4x4A, satd4x4B, satd8x8,
                                results_vector4x4_2[i-1],
                                results_vector4x4_3[i-1],results_vector[
                                i-2]);
686                         $finish;
687                     end
688                 end
689             #CLK_PERIOD;//state CALCULATE_6
690                 original_0 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 0) + address];
691                 original_1 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 1) + address];
692                 original_2 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 2) + address];
693                 original_3 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 3) + address];
694                 original_4 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 4) + address];
695                 original_5 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 5) + address];
696                 original_6 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 6) + address];
697                 original_7 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 7) + address];
698                 candidate_0 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 0) + address];
699                 candidate_1 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 1) + address];
700                 candidate_2 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 2) + address];
701                 candidate_3 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 3) + address];
702                 candidate_4 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 4) + address];
703                 candidate_5 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 5) + address];
704                 candidate_6 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 6) + address];
705                 candidate_7 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 7) + address];
706                 address=address+1;
707             #CLK_PERIOD;//state CALCULATE_7
708                 original_0 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 0) + address];
709                 original_1 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 1) + address];
710                 original_2 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 2) + address];
```

```
711                 original_3 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 3) + address];
712                 original_4 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 4) + address];
713                 original_5 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 5) + address];
714                 original_6 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 6) + address];
715                 original_7 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 7) + address];
716                 candidate_0 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 0) + address];
717                 candidate_1 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 1) + address];
718                 candidate_2 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 2) + address];
719                 candidate_3 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 3) + address];
720                 candidate_4 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 4) + address];
721                 candidate_5 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 5) + address];
722                 candidate_6 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 6) + address];
723                 candidate_7 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 7) + address];
724                 address=address+1;
725             if (controlExecution >= TOTAL_EXECS) begin
726                 enable = 0;
727                 #CLK_PERIOD;//state TERMINATE_0
728                 #CLK_PERIOD;//state TERMINATE_1
729                 if(satd8x8 === results_vector[i-2] &&
                        results_vector4x4_2[i-1] === satd4x4A &&
                        results_vector4x4_3[i-1] === satd4x4B) begin
730                     // $display("Correct\n");
731                 end else begin
732                     $display("ERROR:\n");
733                     $display("calc1: %d\tcalc2: %d\tcalc8x8: %d\
                            t4x4_0: %d\t4x4_1: %d\t4x4_2: %d\t4x4_3: %d\
                            t8x8: %d",satd4x4A, satd4x4B, satd8x8,
                            results_vector4x4_0[i-1],
                            results_vector4x4_1[i-1],
                            results_vector4x4_2[i-1],
                            results_vector4x4_3[i-1], results_vector[i
                            -2]);
734                     $finish;
735                 end
736             end else begin
737             #CLK_PERIOD;//state CALCULATE_0
738                 original_0 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 0) + address];
739                 original_1 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 1) + address];
740                 original_2 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 2) + address];
741                 original_3 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 3) + address];
742                 original_4 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 4) + address];
743                 original_5 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 5) + address];
```

```
744                 original_6 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 6) + address];
745                 original_7 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 7) + address];
746                 candidate_0 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 0) + address];
747                 candidate_1 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 1) + address];
748                 candidate_2 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 2) + address];
749                 candidate_3 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 3) + address];
750                 candidate_4 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 4) + address];
751                 candidate_5 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 5) + address];
752                 candidate_6 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 6) + address];
753                 candidate_7 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 7) + address];
754                 address=address+1;
755             #CLK_PERIOD;//state CALCULATE_1
756                 original_0 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 0) + address];
757                 original_1 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 1) + address];
758                 original_2 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 2) + address];
759                 original_3 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 3) + address];
760                 original_4 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 4) + address];
761                 original_5 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 5) + address];
762                 original_6 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 6) + address];
763                 original_7 = original_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 7) + address];
764                 candidate_0 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 0) + address];
765                 candidate_1 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 1) + address];
766                 candidate_2 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 2) + address];
767                 candidate_3 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 3) + address];
768                 candidate_4 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 4) + address];
769                 candidate_5 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 5) + address];
770                 candidate_6 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 6) + address];
771                 candidate_7 = candidate_vector[(i*TWO_POW_SIZE*
                        TWO_POW_SIZE) + (TWO_POW_SIZE * 7) + address];
772                 address=address+1;
773                 if(results_vector4x4_2[i-1] === satd4x4A &&
                        results_vector4x4_3[i-1] === satd4x4B &&
                        results_vector[i-2] === satd8x8) begin
774                     // $display("Correct\n");
775                 end else begin
776                     if (i == 0) begin
```

```
777                         if(result2_m1 === satd4x4A && result3_m1 ===
                                satd4x4B && result8x8_m2 === satd8x8)
                                begin
778                             // $display("Correct\n");
779                         end else begin
780                             $display("HOUSTON WE HAVE A PROBLEM\n");
781                             $finish;
782                         end
783                     end else begin
784                         if (i == 1) begin
785                             if (results_vector4x4_2[i-1] ===
                                    satd4x4A && results_vector4x4_3[i-1]
                                     === satd4x4B && result8x8_m1 ===
                                    satd8x8) begin
786                                 // $display("Correct\n");
787                             end else begin
788                                 $display("HOUSTON WE HAVE A PROBLEM\
                                        n");
789                                 $finish;
790                             end
791                         end else begin
792                             $display("ERROR:\n");
793                             $display("calc1: %d\tcalc2: %d\tcalc8x8:
                                     %d\t4x4_0: %d\t4x4_1: %d\t4x4_2: %d
                                    \t4x4_3: %d\t8x8: %d",satd4x4A,
                                    satd4x4B, satd8x8,
                                    results_vector4x4_0[i-1],
                                    results_vector4x4_1[i-1],
                                    results_vector4x4_2[i-1],
                                    results_vector4x4_3[i-1],
                                    results_vector[i-2]);
794                             $finish;
795                         end
796                     end
797                 end
798             end
799         end
800         #CLK_PERIOD; //TERMINATE_2
801         #CLK_PERIOD; //TERMINATE_3
802         #CLK_PERIOD; //TERMINATE_4
803         #CLK_PERIOD; //TERMINATE_5
804         #CLK_PERIOD; //TERMINATE_6
805         #CLK_PERIOD; //TERMINATE_7
806         #CLK_PERIOD; //LAST 0
807         #CLK_PERIOD; //LAST 1
808         if(satd8x8 === results_vector[i-2]) begin
809             // $display("Correct\n");
810         end else begin
811             #CLK_PERIOD; //IDLE
812             #CLK_PERIOD; //IDLE
813             #CLK_PERIOD; //IDLE
814             $toggle_stop;
815             $display("Write saif for execution: %d",
                    controlExecution);
816             $toggle_report("output.saif", 0.0000000001, SATD_UV);
817             $finish;
818         end
819     end
820 endmodule
```

A.3 ARQUITETURA

Listing A.7: Código fonte do absoluto para quatro entradas

```
//
    ----------------------------------------------------------------------------

// Design Name : abs_layer_4inputs
// File Name   : abs_layer_4inputs.v
// Function    : Performs the combinational absolute value
    extraction over an array of 4 signed values
// Coder       : Ismael Seidel
//
    ----------------------------------------------------------------------------

module abs_layer_4inputs (
    in_0,
    in_1,
    in_2,
    in_3,
    out_0,
    out_1,
    out_2,
    out_3
);

parameter DATA_WIDTH = 8;


//------------------------------- Input Ports
    ----------------------------------
input signed [DATA_WIDTH-1:0] in_0, in_1, in_2, in_3;

//------------------------------- Output Ports
    ----------------------------------
output [DATA_WIDTH-2:0] out_0, out_1, out_2, out_3;

//---------------------------------- Code Starts Here
    -----------------------------
absolute #(DATA_WIDTH) abs_0(in_0, out_0);
absolute #(DATA_WIDTH) abs_1(in_1, out_1);
absolute #(DATA_WIDTH) abs_2(in_2, out_2);
absolute #(DATA_WIDTH) abs_3(in_3, out_3);

endmodule
```

Listing A.8: Código fonte do bloco operativo para a SATD de blocos 8×8

```
//
    ----------------------------------------------------------------------------

// Design Name : satd_8x8_8pairs_operative
// File Name   : satd_8x8_8pairs_operative.v
// Function    : SATD of 8x8 blocks with 8 pixel pairs as input
// Coder       : Ismael Seidel and Marcio Monteiro
//
    ----------------------------------------------------------------------------
```

```
7  module satd_8x8_8pairs_operative(
8      clock,
9      reset,
10     enable_inputs,
11     reset_transpose_buffer,
12     enable_transpose_buffer,
13     change_transpose_buffer_direction,
14     reset_psatd,
15     enable_psatd,
16     reset_satd,
17     enable_satd,
18     out_HT_0,
19     out_HT_1,
20     out_HT_2,
21     out_HT_3,
22     out_HT_4,
23     out_HT_5,
24     out_HT_6,
25     out_HT_7,
26     satd
27 );
28
29 parameter DATA_WIDTH = 8;
30
31 //------------------------------- Input Ports
       ---------------------------------
32 input clock, reset, enable_inputs, reset_transpose_buffer,
       enable_transpose_buffer, change_transpose_buffer_direction,
       reset_psatd, enable_psatd, reset_satd, enable_satd;
33 input [DATA_WIDTH+4:0] out_HT_0, out_HT_1, out_HT_2, out_HT_3,
       out_HT_4, out_HT_5, out_HT_6, out_HT_7;
34
35 //------------------------------- Output Ports
       ---------------------------------
36 output reg [DATA_WIDTH+11:0] satd;
37
38 //------------------------------- Internal Wires
       ----------------------------
39 wire transpose_buffer_direction;
40 wire signed [DATA_WIDTH+5:0] out_of_1st_1d_transform_0,
       out_of_1st_1d_transform_1, out_of_1st_1d_transform_2,
       out_of_1st_1d_transform_3, out_of_1st_1d_transform_4,
       out_of_1st_1d_transform_5, out_of_1st_1d_transform_6,
       out_of_1st_1d_transform_7;
41 wire signed [DATA_WIDTH+5:0] out_of_transpose_buffer_0,
       out_of_transpose_buffer_1, out_of_transpose_buffer_2,
       out_of_transpose_buffer_3, out_of_transpose_buffer_4,
       out_of_transpose_buffer_5, out_of_transpose_buffer_6,
       out_of_transpose_buffer_7;
42 wire signed [DATA_WIDTH+6:0] out_of_2nd_1d_transform_0,
       out_of_2nd_1d_transform_1, out_of_2nd_1d_transform_2,
       out_of_2nd_1d_transform_3, out_of_2nd_1d_transform_4,
       out_of_2nd_1d_transform_5, out_of_2nd_1d_transform_6,
       out_of_2nd_1d_transform_7;
43 wire [DATA_WIDTH+5:0] absolute_transformed_difference_0,
       absolute_transformed_difference_1,
       absolute_transformed_difference_2,
       absolute_transformed_difference_3,
       absolute_transformed_difference_4,
       absolute_transformed_difference_5,
       absolute_transformed_difference_6,
```

```
    absolute_transformed_difference_7;
wire [DATA_WIDTH+8:0] sum;
wire [DATA_WIDTH+12:0] psatd;

//--------------------------------- Code Starts Here
    ------------------------------
tff_async_reset transpose_buffer_direction_holder(
    change_transpose_buffer_direction, clock, reset,
    transpose_buffer_direction);

transform_1d_8inputs_mod #(DATA_WIDTH+5) first_1d_transform(out_HT_0
    , out_HT_1, out_HT_2, out_HT_3, out_HT_4, out_HT_5, out_HT_6,
    out_HT_7, out_of_1st_1d_transform_0, out_of_1st_1d_transform_1,
    out_of_1st_1d_transform_2, out_of_1st_1d_transform_3,
    out_of_1st_1d_transform_4, out_of_1st_1d_transform_5,
    out_of_1st_1d_transform_6, out_of_1st_1d_transform_7);

transpose_buffer_8x8 #(DATA_WIDTH+6) transpose_buffer(clock,
    reset_transpose_buffer, enable_transpose_buffer,
    transpose_buffer_direction, out_of_1st_1d_transform_0,
    out_of_1st_1d_transform_1, out_of_1st_1d_transform_2,
    out_of_1st_1d_transform_3, out_of_1st_1d_transform_4,
    out_of_1st_1d_transform_5, out_of_1st_1d_transform_6,
    out_of_1st_1d_transform_7, out_of_transpose_buffer_0,
    out_of_transpose_buffer_1, out_of_transpose_buffer_2,
    out_of_transpose_buffer_3, out_of_transpose_buffer_4,
    out_of_transpose_buffer_5, out_of_transpose_buffer_6,
    out_of_transpose_buffer_7);

transform_1d_8inputs_mod_sec #(DATA_WIDTH+6) second_1d_transform(
    out_of_transpose_buffer_0, out_of_transpose_buffer_1,
    out_of_transpose_buffer_2, out_of_transpose_buffer_3,
    out_of_transpose_buffer_4, out_of_transpose_buffer_5,
    out_of_transpose_buffer_6, out_of_transpose_buffer_7,
    out_of_2nd_1d_transform_0, out_of_2nd_1d_transform_1,
    out_of_2nd_1d_transform_2, out_of_2nd_1d_transform_3,
    out_of_2nd_1d_transform_4, out_of_2nd_1d_transform_5,
    out_of_2nd_1d_transform_6, out_of_2nd_1d_transform_7);

abs_layer_8inputs #(DATA_WIDTH+7) absolute1(
    out_of_2nd_1d_transform_0, out_of_2nd_1d_transform_1,
    out_of_2nd_1d_transform_2, out_of_2nd_1d_transform_3,
    out_of_2nd_1d_transform_4, out_of_2nd_1d_transform_5,
    out_of_2nd_1d_transform_6, out_of_2nd_1d_transform_7,
    absolute_transformed_difference_0,
    absolute_transformed_difference_1,
    absolute_transformed_difference_2,
    absolute_transformed_difference_3,
    absolute_transformed_difference_4,
    absolute_transformed_difference_5,
    absolute_transformed_difference_6,
    absolute_transformed_difference_7);

sum_tree_8inputs #(DATA_WIDTH+6) sum_tree(
    absolute_transformed_difference_0,
    absolute_transformed_difference_1,
    absolute_transformed_difference_2,
    absolute_transformed_difference_3,
    absolute_transformed_difference_4,
    absolute_transformed_difference_5,
    absolute_transformed_difference_6,
```

```
    absolute_transformed_difference_7, sum);

accumulator #(DATA_WIDTH+9, DATA_WIDTH+13) acc(clock, reset_psatd,
    enable_psatd, sum, psatd);

always @(posedge clock) begin
    if (reset_satd) begin
        satd <= 0;
    end
    else if (enable_satd) begin
        satd <= (psatd+sum)>>2;
    end
end

endmodule
```

Listing A.9: Código fonte do absoluto para oito entradas

```
//
    ------------------------------------------------------------------------

// Design Name : abs_layer_8inputs
// File Name   : abs_layer_8inputs.v
// Function    : Performs the combinational absolute value
    extraction over an array of 8 signed values
// Coder       : Ismael Seidel
//
    ------------------------------------------------------------------------

module abs_layer_8inputs (
    in_0,
    in_1,
    in_2,
    in_3,
    in_4,
    in_5,
    in_6,
    in_7,
    out_0,
    out_1,
    out_2,
    out_3,
    out_4,
    out_5,
    out_6,
    out_7
);

parameter DATA_WIDTH = 8;

//------------------------------ Input Ports
    --------------------------------
input signed [DATA_WIDTH-1:0] in_0, in_1, in_2, in_3, in_4, in_5,
    in_6, in_7;

//------------------------------ Output Ports
    -------------------------------
output [DATA_WIDTH-2:0] out_0, out_1, out_2, out_3, out_4, out_5,
    out_6, out_7;

```

```
34 //------------------------------- Code Starts Here
       ----------------------------
35 absolute #(DATA_WIDTH) abs_0(in_0, out_0);
36 absolute #(DATA_WIDTH) abs_1(in_1, out_1);
37 absolute #(DATA_WIDTH) abs_2(in_2, out_2);
38 absolute #(DATA_WIDTH) abs_3(in_3, out_3);
39 absolute #(DATA_WIDTH) abs_4(in_4, out_4);
40 absolute #(DATA_WIDTH) abs_5(in_5, out_5);
41 absolute #(DATA_WIDTH) abs_6(in_6, out_6);
42 absolute #(DATA_WIDTH) abs_7(in_7, out_7);
43
44 endmodule
```

Listing A.10: Código fonte do *datapath* da arquitetura

```
1  //
       ------------------------------------------------------------------------
2  // Design Name : SATD
3  // File Name   : SATD.v
4  // Function    : SATD of 4x4 blocks with 4 pixel pairs as input
5  // Coder       : Ismael Seidel and Marcio Monteiro
6  //
       ------------------------------------------------------------------------
7
8  module SATD(
9      clock,
10     reset,
11     enable,
12     original_0,
13     original_1,
14     original_2,
15     original_3,
16     original_4,
17     original_5,
18     original_6,
19     original_7,
20     candidate_0,
21     candidate_1,
22     candidate_2,
23     candidate_3,
24     candidate_4,
25     candidate_5,
26     candidate_6,
27     candidate_7,
28     doneA,
29     satd4x4A,
30     doneB,
31     satd4x4B,
32     done,
33     satd8x8
34 );
35
36 parameter DATA_WIDTH = 8;
37
38 //------------------------------- Input Ports
       ----------------------------------
39 input clock, reset, enable;
40 input [DATA_WIDTH-1:0] original_0, original_1, original_2,
       original_3, original_4, original_5, original_6, original_7,
```

```
     candidate_0, candidate_1, candidate_2, candidate_3, candidate_4,
      candidate_5, candidate_6, candidate_7;

//------------------------------- Output Ports
     ------------------------------------
output [DATA_WIDTH+7:0] satd4x4A, satd4x4B;
output [DATA_WIDTH+11:0] satd8x8;
//output [DATA_WIDTH+4:0] out_HT_0, out_HT_1, out_HT_2, out_HT_3,
     out_HT_4, out_HT_5, out_HT_6, out_HT_7;
output done, doneA, doneB;

//------------------------------- Internal Wires
     ----------------------------
wire enable_inputs4x4, reset_transpose_buffer4x4,
     enable_transpose_buffer4x4, change_transpose_buffer_direction4x4
     , reset_psatd4x4, enable_psatd4x4, reset_satd4x4, enable_satd4x4
     , enable_inputs8x8, reset_transpose_buffer8x8,
     enable_transpose_buffer8x8, change_transpose_buffer_direction8x8
     , reset_psatd8x8, enable_psatd8x8, reset_satd8x8, enable_satd8x8
     ;

//------------------------------- Code Starts Here
     ---------------------------
mainControl controlePrincipal( clock, reset, enable,
     enable_inputs4x4, reset_transpose_buffer4x4,
     enable_transpose_buffer4x4, change_transpose_buffer_direction4x4
     , reset_psatd4x4, enable_psatd4x4, reset_satd4x4, enable_satd4x4
     , doneA, doneB, enable_inputs8x8, reset_transpose_buffer8x8,
     enable_transpose_buffer8x8, change_transpose_buffer_direction8x8
     , reset_psatd8x8, enable_psatd8x8, reset_satd8x8, enable_satd8x8
     , done);

mainOperative operativoPrincipal( clock, reset, enable,
     enable_inputs4x4, reset_transpose_buffer4x4,
     enable_transpose_buffer4x4, change_transpose_buffer_direction4x4
     , reset_psatd4x4, enable_psatd4x4, reset_satd4x4, enable_satd4x4
     , enable_inputs8x8, reset_transpose_buffer8x8,
     enable_transpose_buffer8x8, change_transpose_buffer_direction8x8
     , reset_psatd8x8, enable_psatd8x8, reset_satd8x8, enable_satd8x8
     , original_0, original_1, original_2, original_3, original_4,
     original_5, original_6, original_7, candidate_0, candidate_1,
     candidate_2, candidate_3, candidate_4, candidate_5, candidate_6,
      candidate_7, doneA, satd4x4A, doneB, satd4x4B, done, satd8x8);

endmodule
```

Listing A.11: Código fonte do absoluto

```
/*
 *  \file absolute.v
 *  \brief Implementation of an parameterized absolute.
 *  \author Ismael Seidel
 *  \Date:   2015-08-13 16:55:45
 *  \Last Modified by:   ismaelseidel
 *  \Last Modified time: 2015-08-05 17:00:35
 *
 *  This absolute implementation is parameterized so as to the
      number of bits in its inputs.
 *  Thus, the value DATA_WIDTH is the number of bits of in_0 and
      also the number of bits of in_1.
```

```
 *  Notice that the output is one bit larger, so as to accomodate
      the carry out of this absolute.
 *
 *  Developed at Embedded Computing Lab (ECL), 2015.
 *
 */

//Absolute module definition.
module absolute (
    in,
    out
);

//parameter indicating the number of bits in the input. The output
    is one bit shorter.
parameter DATA_WIDTH = 8;

//declaration of number of bits for input in.
input signed[DATA_WIDTH-1:0] in;

//declaration of output.
output [DATA_WIDTH-2:0] out;

//the output is assigned to be the inverse of in for the case when
    in is negative. Otherwise, the output is the input without is
    first bit (MSB).
assign out = (in[DATA_WIDTH-1]) ? -in[DATA_WIDTH-2:0] : in[
    DATA_WIDTH-2:0];

endmodule
```

Listing A.12: Código fonte da árvore de somadores para quatro entradas

```
//
    ----------------------------------------------------------------------------

// Design Name : sum_tree_4inputs
// File Name   : sum_tree_4inputs.v
// Function    : Sums up 4 inputs in a combinational fashion
// Coder       : Ismael Seidel
//
    ----------------------------------------------------------------------------

module sum_tree_4inputs (
    in_0,
    in_1,
    in_2,
    in_3,
    out
);

parameter DATA_WIDTH = 8;

//------------------------------- Input Ports
    ----------------------------------
input [DATA_WIDTH-1:0] in_0, in_1, in_2, in_3;

//------------------------------- Output Ports
    ----------------------------------
output [DATA_WIDTH+1:0] out;

```

```
23 //------------------------------- Internal Wires
       ------------------------------
24 wire [DATA_WIDTH:0] sum_0, sum_1;
25 wire [DATA_WIDTH+1:0] sum_2;
26
27 //-------------------------------- Code Starts Here
       ------------------------------
28 adder #(DATA_WIDTH) adder_0(in_0, in_1, sum_0);
29 adder #(DATA_WIDTH) adder_1(in_2, in_3, sum_1);
30
31 adder #(DATA_WIDTH+1) adder_2(sum_0, sum_1, sum_2);
32
33 assign out = sum_2;
34
35 endmodule
```

Listing A.13: Código fonte do acumulador

```
1  //
       ------------------------------------------------------------------------------
2  // Design Name : accumulator
3  // File Name   : accumulator.v
4  // Function    : Accumulator to sum up values in a sequential
       fashion
5  // Coder       : Ismael Seidel
6  //
       ------------------------------------------------------------------------------
7  module accumulator (
8      clock,
9      reset,
10     enable,
11     in,
12     out
13 );
14
15 parameter IN_DATA_WIDTH = 8;
16 parameter MAX_DATA_WIDTH = 12;
17
18 //------------------------------- Input Ports
       ----------------------------------
19 input clock, reset, enable;
20 input [IN_DATA_WIDTH-1:0] in;
21
22
23 //------------------------------- Output Ports
       ----------------------------------
24 output [MAX_DATA_WIDTH-1:0] out;
25
26 //------------------------------- Internal Regiters
       ------------------------------
27 reg [MAX_DATA_WIDTH-1:0] out;
28
29 //-------------------------------- Code Starts Here
       ------------------------------
30 always @(posedge clock) begin
31     if (reset) begin
32         out <= 0;
33     end
34     else if (enable) begin
```

```
35          out <=out+in;
36      end
37  end
38
39  endmodule
```

Listing A.14: Código fonte da árvore de somadores para oito entradas

```
1   //
        ----------------------------------------------------------------------------
2   // Design Name : sum_tree_8inputs
3   // File Name   : sum_tree_8inputs.v
4   // Function    : Sums up 8 inputs in a combinational fashion
5   // Coder       : Ismael Seidel
6   //
        ----------------------------------------------------------------------------
7   module sum_tree_8inputs (
8       in_0,
9       in_1,
10      in_2,
11      in_3,
12      in_4,
13      in_5,
14      in_6,
15      in_7,
16      out
17  );
18
19  parameter DATA_WIDTH = 8;
20
21  //------------------------------- Input Ports
        ----------------------------------
22  input [DATA_WIDTH-1:0] in_0, in_1, in_2, in_3, in_4, in_5, in_6,
        in_7;
23
24  //------------------------------- Output Ports
        ----------------------------------
25  output [DATA_WIDTH+2:0] out;
26
27  //------------------------------- Internal Wires
        -----------------------------
28  wire [DATA_WIDTH:0] sum_0, sum_1, sum_2, sum_3;
29  wire [DATA_WIDTH+1:0] sum_4, sum_5;
30  wire [DATA_WIDTH+2:0] sum_6;
31
32  //------------------------------- Code Starts Here
        ----------------------------
33  adder #(DATA_WIDTH) adder_0(in_0, in_1, sum_0);
34  adder #(DATA_WIDTH) adder_1(in_2, in_3, sum_1);
35  adder #(DATA_WIDTH) adder_2(in_4, in_5, sum_2);
36  adder #(DATA_WIDTH) adder_3(in_6, in_7, sum_3);
37
38  adder #(DATA_WIDTH+1) adder_4(sum_0, sum_1, sum_4);
39  adder #(DATA_WIDTH+1) adder_5(sum_2, sum_3, sum_5);
40
41  adder #(DATA_WIDTH+2) adder_6(sum_4, sum_5, sum_6);
42
43  assign out = sum_6;
44
```

```
45 endmodule
```

Listing A.15: Código fonte do somador

```
1  /*
2   *  \file adder.v
3   *  \brief Implementation of an parameterized adder.
4   *  \author Ismael Seidel
5   *  \Date:   2015-07-13 15:11:45
6   *  \Last Modified by:   ismaelseidel
7   *  \Last Modified time: 2015-07-14 11:35:54
8   *
9   *  This adder implementation is parameterized so as to the number
        of bits in its inputs.
10  *  Thus, the value DATA_WIDTH is the number of bits of in_0 and
        also the number of bits of in_1.
11  *  Notice that the output is one bit larger, so as to accomodate
        the carry out of this adder.
12  *
13  *  Developed at Embedded Computing Lab (ECL), 2015.
14  *
15  */
16
17 //Adder module definition.
18 module adder (
19     in_0,
20     in_1,
21     out
22 );
23
24 //parameter indicating the number of bits in the inputs. The output
       is one bit larger.
25 parameter DATA_WIDTH = 8;
26
27 //declaration of number of bits for inputs in_0 and in_1.
28 input [DATA_WIDTH-1:0] in_0, in_1;
29
30 //declaration of output.
31 output [DATA_WIDTH:0] out;
32
33 //the output is assigned to be the sum of in_0 and in_1. As the
       output is one bit longer than the inputs, it should accomodate
       the carry out.
34 assign out = in_0+in_1;
35
36 endmodule
```

Listing A.16: Código fonte da *butterfly de duas entradas*

```
1  //
       ----------------------------------------------------------------------
2  // Design Name : butterfly2
3  // File Name   : butterfly2.v
4  // Function    : Implements a butterfly for 2 signed inputs
5  // Coder       : Ismael Seidel
6  //
       ----------------------------------------------------------------------
```

```
7  module butterfly2 (
8      in_0,
9      in_1,
10     out_0,
11     out_1
12 );
13
14 parameter DATA_WIDTH = 8;
15
16 input signed [DATA_WIDTH-1:0] in_0, in_1;
17 output signed [DATA_WIDTH:0] out_0, out_1;
18
19 assign out_0 = in_0 + in_1;
20 assign out_1 = in_0 - in_1;
21
22 endmodule
```

Listing A.17: Código fonte do *latch*

```
1  //-----------------------------------------------------
2  // Design Name : tff_async_reset
3  // File Name   : tff_async_reset.v
4  // Function    : T flip-flop async reset
5  // Coder       : Deepak Kumar Tala
6  //-----------------------------------------------------
7  module tff_async_reset (
8  data  , // Data Input
9  clk   , // Clock Input
10 reset , // Reset input
11 q       // Q output
12 );
13 //-----------Input Ports---------------
14 input data, clk, reset;
15 //-----------Output Ports---------------
16 output q;
17 //------------Internal Variables--------
18 reg q;
19 //-------------Code Starts Here---------
20 always @ ( posedge clk or posedge reset)
21 if (reset) begin
22   q <= 1'b0;
23 end else if (data) begin
24   q <= !q;
25 end
26
27 endmodule //End Of Module tff_async_reset
```

Listing A.18: Código fonte da diferença para quatro entradas

```
1  //
       ------------------------------------------------------------------------
2  // Design Name : difference_layer_4pairs
3  // File Name   : difference_layer_4pairs.v
4  // Function    : Differences over an array of 4 unsigned input pairs
5  // Coder       : Ismael Seidel
6  //
       ------------------------------------------------------------------------
```

```
7  module difference_layer_4pairs (
8      in_A_0,
9      in_A_1,
10     in_A_2,
11     in_A_3,
12     in_B_0,
13     in_B_1,
14     in_B_2,
15     in_B_3,
16     out_0,
17     out_1,
18     out_2,
19     out_3
20 );parameter DATA_WIDTH = 8;
21
22
23 //------------------------------- Input Ports
       ----------------------------------
24 input [DATA_WIDTH-1:0] in_A_0, in_A_1, in_A_2, in_A_3, in_B_0,
       in_B_1, in_B_2, in_B_3;
25
26 //------------------------------- Output Ports
       ---------------------------------
27 output [DATA_WIDTH:0] out_0, out_1, out_2, out_3;
28
29 //--------------------------------- Code Starts Here
       ----------------------------
30 difference_of_unsigned_inputs #(DATA_WIDTH) difference_0(in_A_0,
       in_B_0, out_0);
31 difference_of_unsigned_inputs #(DATA_WIDTH) difference_1(in_A_1,
       in_B_1, out_1);
32 difference_of_unsigned_inputs #(DATA_WIDTH) difference_2(in_A_2,
       in_B_2, out_2);
33 difference_of_unsigned_inputs #(DATA_WIDTH) difference_3(in_A_3,
       in_B_3, out_3);
34
35 endmodule
```

Listing A.19: Código fonte dasegunda transformada da $SATD_{4\times4}$

```
1  module transform_1d_4inputs_second (
2      in_0,
3      in_1,
4      in_2,
5      in_3,
6      out_0,
7      out_1,
8      out_2,
9      out_3
10 );
11
12 parameter DATA_WIDTH = 8;
13
14 input signed [DATA_WIDTH-1:0] in_0, in_1, in_2, in_3;
15 output signed [DATA_WIDTH+1:0] out_0, out_1, out_2, out_3;
16 wire [DATA_WIDTH:0] wire_0_0, wire_0_1, wire_0_2, wire_0_3;
17 //first layer (0)
18 butterfly2 #(DATA_WIDTH) butterfly2_0_0(in_0, in_1, wire_0_0,
       wire_0_1);
19 butterfly2 #(DATA_WIDTH) butterfly2_0_2(in_2, in_3, wire_0_2,
       wire_0_3);
```

```
20 //last layer (1)
21 butterfly2 #(DATA_WIDTH+1) butterfly2_1_0(wire_0_0, wire_0_2, out_0,
       out_1);
22 butterfly2 #(DATA_WIDTH+1) butterfly2_1_1(wire_0_1, wire_0_3, out_2,
       out_3);
23
24 endmodule
```

Listing A.20: Código fonte da diferença de dois valores sem sinal

```
1  /*
2   *  \file difference_of_unsigned_inputs.v
3   *  \brief Implementation of an parameterized
        difference_of_unsigned_inputs.
4   *  \author Ismael Seidel
5   *  \Date:   2015-07-13 15:11:45
6   *  \Last Modified by:   ismaelseidel
7   *  \Last Modified time: 2015-07-14 11:35:54
8   *
9   *  This difference_of_unsigned_inputs implementation is
        parameterized so as to the number of bits in its inputs.
10  *  Thus, the value DATA_WIDTH is the number of bits of in_0 and
        also the number of bits of in_1.
11  *  Notice that the output is one bit larger, so as to accomodate
        the carry out of this difference_of_unsigned_inputs.
12  *
13  *  Developed at Embedded Computing Lab (ECL), 2015.
14  *
15  */
16
17 //difference_of_unsigned_inputs module definition.
18 module difference_of_unsigned_inputs (
19     in_0,
20     in_1,
21     out
22 );
23
24 //parameter indicating the number of bits in the inputs. The output
       is one bit larger.
25 parameter DATA_WIDTH = 8;
26
27 //declaration of number of bits for inputs in_0 and in_1.
28 input [DATA_WIDTH-1:0] in_0, in_1;
29
30 //declaration of output.
31 output signed [DATA_WIDTH:0] out;
32
33 //the output is assigned to be the difference between in_0 and in_1.
       As the output is one bit longer than the inputs, it should
      accomodate the carry out (signal).
34 assign out = in_0-in_1;
35
36 endmodule
```

Listing A.21: Código fonte da primeira transformada da $SATD_{4\times4}$

```
1 module transform_1d_4inputs (
2     in_0,
3     in_1,
```

```
    in_2,
    in_3,
    out_0,
    out_1,
    out_2,
    out_3
);

parameter DATA_WIDTH = 8;

input signed [DATA_WIDTH-1:0] in_0, in_1, in_2, in_3;
output signed [DATA_WIDTH+1:0] out_0, out_1, out_2, out_3;
wire [DATA_WIDTH:0] wire_0_0, wire_0_1, wire_0_2, wire_0_3;
//first layer (0)
butterfly2 #(DATA_WIDTH) butterfly2_0_0(in_0, in_1, wire_0_0,
    wire_0_1);
butterfly2 #(DATA_WIDTH) butterfly2_0_2(in_2, in_3, wire_0_2,
    wire_0_3);
//last layer (1)
butterfly2 #(DATA_WIDTH+1) butterfly2_1_0(wire_0_0, wire_0_2, out_0,
     out_1);
butterfly2 #(DATA_WIDTH+1) butterfly2_1_1(wire_0_1, wire_0_3, out_2,
     out_3);

endmodule
```

Listing A.22: Código fonte do *buffer* 4×4

```
//
    ----------------------------------------------------------------------

// Design Name : input_buffer_4pairs
// File Name   : input_buffer_4pairs.v
// Function    : A simple module to hold the 4 inputs
// Coder       : Ismael Seidel
//
    ----------------------------------------------------------------------

module input_buffer_4pairs (
    clock,
    reset,
    enable,
    in_0,
    in_1,
    in_2,
    in_3,
    in_4,
    in_5,
    in_6,
    in_7,
    out_0,
    out_1,
    out_2,
    out_3,
    out_4,
    out_5,
    out_6,
    out_7
);

parameter DATA_WIDTH = 8;
```

```
30
31 //------------------------------- Input Ports
       ----------------------------------
32 input clock, reset, enable;
33 input [DATA_WIDTH-1:0] in_0, in_1, in_2, in_3, in_4, in_5, in_6,
       in_7;
34
35 //------------------------------- Output Ports
       ----------------------------------
36 output reg [DATA_WIDTH-1:0] out_0, out_1, out_2, out_3, out_4, out_5
       , out_6, out_7;
37
38 //---------------------------------- Code Starts Here
       ------------------------------
39 always @(posedge clock or posedge reset) begin
40     if (reset) begin
41         out_0 <= 0;
42         out_1 <= 0;
43         out_2 <= 0;
44         out_3 <= 0;
45         out_4 <= 0;
46         out_5 <= 0;
47         out_6 <= 0;
48         out_7 <= 0;
49     end
50     else if (enable) begin
51         out_0 <= in_0;
52         out_1 <= in_1;
53         out_2 <= in_2;
54         out_3 <= in_3;
55         out_4 <= in_4;
56         out_5 <= in_5;
57         out_6 <= in_6;
58         out_7 <= in_7;
59     end
60 end
61
62 endmodule
```

Listing A.23: Código fonte do segundo ajuste

```
1  module transform_1d_8inputs_mod_sec (
2      in_0,
3      in_1,
4      in_2,
5      in_3,
6      in_4,
7      in_5,
8      in_6,
9      in_7,
10     out_0,
11     out_1,
12     out_2,
13     out_3,
14     out_4,
15     out_5,
16     out_6,
17     out_7
18 );
19
20 parameter DATA_WIDTH = 8;
```

```
21
22 input signed [DATA_WIDTH-1:0] in_0, in_1, in_2, in_3, in_4, in_5,
       in_6, in_7;
23 output signed [DATA_WIDTH:0] out_0, out_1, out_2, out_3, out_4,
       out_5, out_6, out_7;
24
25 butterfly2 #(DATA_WIDTH) butterfly2_0_0(in_0, in_7, out_0, out_1);
26 butterfly2 #(DATA_WIDTH) butterfly2_0_2(in_1, in_6, out_2, out_3);
27 butterfly2 #(DATA_WIDTH) butterfly2_0_4(in_2, in_5, out_4, out_5);
28 butterfly2 #(DATA_WIDTH) butterfly2_0_6(in_3, in_4, out_6, out_7);
29
30 endmodule
```

Listing A.24: Código fonte do controle da arquitetura

```
1  //
       ----------------------------------------------------------------------------
2  // Design Name : mainControl
3  // File Name   : mainControl.v
4  // Function    : SATD of 4x4 blocks with 4 pixel pairs as input
5  // Coder       : Ismael Seidel and Marcio Monteiro
6  //
       ----------------------------------------------------------------------------
7  module mainControl(
8      clock,
9      reset,
10     enable,
11     enable_inputs4x4, // controls the inputs of 4x4
12     reset_transpose_buffer4x4,
13     enable_transpose_buffer4x4,
14     change_transpose_buffer_direction4x4,
15     reset_psatd4x4,
16     enable_psatd4x4,
17     reset_satd4x4,
18     enable_satd4x4,
19     done4x4,
20     done4x4B,
21     enable_inputs8x8,
22     reset_transpose_buffer8x8,
23     enable_transpose_buffer8x8,
24     change_transpose_buffer_direction8x8,
25     reset_psatd8x8,
26     enable_psatd8x8,
27     reset_satd8x8,
28     enable_satd8x8,
29     done8x8
30 );
31
32 parameter DATA_WIDTH = 8;
33
34 //------------------------------ Input Ports
       ----------------------------------
35 input clock, reset, enable;
36
37 //------------------------------ Output Ports
       ----------------------------------
38 output reg enable_inputs4x4, reset_transpose_buffer4x4,
       enable_transpose_buffer4x4, change_transpose_buffer_direction4x4
       , reset_psatd4x4, enable_psatd4x4, reset_satd4x4, enable_satd4x4
```

```
    , done4x4, done4x4B, enable_inputs8x8, reset_transpose_buffer8x8
    , enable_transpose_buffer8x8,
    change_transpose_buffer_direction8x8, reset_psatd8x8,
    enable_psatd8x8, reset_satd8x8, enable_satd8x8, done8x8;

reg [5:0] state;
parameter
IDLE = 0,
INITIALIZE_0 = 1,
INITIALIZE_1 = 2,
INITIALIZE_2 = 3,
INITIALIZE_3 = 4,
INITIALIZE_4 = 5,
INITIALIZE_5 = 6,
INITIALIZE_6 = 7,
INITIALIZE_7 = 8,

INITIALIZE_8 = 9,
INITIALIZE_9 = 10,
INITIALIZE_10 = 11,
INITIALIZE_11 = 12,

INITIALIZE_12 = 13,
INITIALIZE_13 = 14,
INITIALIZE_14 = 15,
INITIALIZE_15 = 16,
INITIALIZE_16 = 17,
INITIALIZE_17 = 18,
INITIALIZE_18 = 19,
INITIALIZE_19 = 20,

PRE_CALCULATE_0 = 21,
PRE_CALCULATE_1 = 22,
CALCULATE_0 = 23,
CALCULATE_1 = 24,
CALCULATE_2 = 25,
CALCULATE_3 = 26,
CALCULATE_4 = 27,
CALCULATE_5 = 28,
CALCULATE_6 = 29,
CALCULATE_7 = 30,
TERMINATE_0 = 31,
TERMINATE_1 = 32,
TERMINATE_2 = 33,
TERMINATE_3 = 34,
TERMINATE_4 = 35,
TERMINATE_5 = 36,
TERMINATE_6 = 37,
TERMINATE_7 = 38,
LAST_0 = 39,
LAST_1 = 40,
PRE_TERMINATE_0 = 41,
PRE_TERMINATE_1 = 42;

always @ (state) begin
    case (state)
        IDLE:begin
            enable_inputs4x4 = 1'b0;
            enable_transpose_buffer4x4 = 1'b0;
            change_transpose_buffer_direction4x4 = 1'b0;
            reset_psatd4x4 = 1'b1;
```

```
96              enable_psatd4x4 = 1'b0;
97              reset_satd4x4 = 1'b1;
98              enable_satd4x4 = 1'b0;
99              reset_transpose_buffer4x4 = 1'b1;
100             done4x4 = 1'b0;
101             done4x4B = 1'b0;
102             enable_inputs8x8 = 1'b0;
103             enable_transpose_buffer8x8 = 1'b0;
104             change_transpose_buffer_direction8x8 = 1'b0;
105             reset_psatd8x8 = 1'b1;
106             enable_psatd8x8 = 1'b0;
107             reset_satd8x8 = 1'b1;
108             enable_satd8x8 = 1'b0;
109             reset_transpose_buffer8x8 = 1'b1;
110             done8x8 = 1'b0;
111         end
112         INITIALIZE_0:begin
113             enable_inputs4x4 = 1'b1;
114             enable_transpose_buffer4x4 = 1'b1;
115             change_transpose_buffer_direction4x4 = 1'b0;
116             reset_psatd4x4 = 1'b0;
117             enable_psatd4x4 = 1'b0;
118             reset_satd4x4 = 1'b0;
119             enable_satd4x4 = 1'b0;
120             reset_transpose_buffer4x4 = 1'b0;
121             done4x4 = 1'b0;
122             done4x4B = 1'b0;
123             enable_inputs8x8 = 1'b0;
124             enable_transpose_buffer8x8 = 1'b0;
125             change_transpose_buffer_direction8x8 = 1'b0;
126             reset_psatd8x8 = 1'b0;
127             enable_psatd8x8 = 1'b0;
128             reset_satd8x8 = 1'b0;
129             enable_satd8x8 = 1'b0;
130             reset_transpose_buffer8x8 = 1'b0;
131             done8x8 = 1'b0;
132         end
133         INITIALIZE_1:begin
134             enable_inputs4x4 = 1'b1;
135             enable_transpose_buffer4x4 = 1'b1;
136             change_transpose_buffer_direction4x4 = 1'b0;
137             reset_psatd4x4 = 1'b0;
138             enable_psatd4x4 = 1'b0;
139             reset_satd4x4 = 1'b0;
140             enable_satd4x4 = 1'b0;
141             reset_transpose_buffer4x4 = 1'b0;
142             done4x4 = 1'b0;
143             done4x4B = 1'b0;
144             enable_inputs8x8 = 1'b0;
145             enable_transpose_buffer8x8 = 1'b0;
146             change_transpose_buffer_direction8x8 = 1'b0;
147             reset_psatd8x8 = 1'b0;
148             enable_psatd8x8 = 1'b0;
149             reset_satd8x8 = 1'b0;
150             enable_satd8x8 = 1'b0;
151             reset_transpose_buffer8x8 = 1'b0;
152             done8x8 = 1'b0;
153         end
154         INITIALIZE_2:begin
155             enable_inputs4x4 = 1'b1;
156             enable_transpose_buffer4x4 = 1'b1;
```

```
157             change_transpose_buffer_direction4x4 = 1'b0;
158             reset_psatd4x4 = 1'b0;
159             enable_psatd4x4 = 1'b0;
160             reset_satd4x4 = 1'b0;
161             enable_satd4x4 = 1'b0;
162             reset_transpose_buffer4x4 = 1'b0;
163             done4x4 = 1'b0;
164             done4x4B = 1'b0;
165             enable_inputs8x8 = 1'b0;
166             enable_transpose_buffer8x8 = 1'b0;
167             change_transpose_buffer_direction8x8 = 1'b0;
168             reset_psatd8x8 = 1'b0;
169             enable_psatd8x8 = 1'b0;
170             reset_satd8x8 = 1'b0;
171             enable_satd8x8 = 1'b0;
172             reset_transpose_buffer8x8 = 1'b0;
173             done8x8 = 1'b0;
174         end
175         INITIALIZE_3:begin
176             enable_inputs4x4 = 1'b1;
177             enable_transpose_buffer4x4 = 1'b1;
178             change_transpose_buffer_direction4x4 = 1'b0;
179             reset_psatd4x4 = 1'b0;
180             enable_psatd4x4 = 1'b1;
181             reset_satd4x4 = 1'b0;
182             enable_satd4x4 = 1'b0;
183             reset_transpose_buffer4x4 = 1'b0;
184             done4x4 = 1'b0;
185             done4x4B = 1'b0;
186             enable_inputs8x8 = 1'b0;
187             enable_transpose_buffer8x8 = 1'b0;
188             change_transpose_buffer_direction8x8 = 1'b0;
189             reset_psatd8x8 = 1'b0;
190             enable_psatd8x8 = 1'b0;
191             reset_satd8x8 = 1'b0;
192             enable_satd8x8 = 1'b0;
193             reset_transpose_buffer8x8 = 1'b0;
194             done8x8 = 1'b0;
195         end
196         INITIALIZE_4:begin
197             enable_inputs4x4 = 1'b1;
198             enable_transpose_buffer4x4 = 1'b1;
199             change_transpose_buffer_direction4x4 = 1'b1;
200             reset_psatd4x4 = 1'b0;
201             enable_psatd4x4 = 1'b1;
202             reset_satd4x4 = 1'b0;
203             enable_satd4x4 = 1'b0;
204             reset_transpose_buffer4x4 = 1'b0;
205             done4x4 = 1'b0;
206             done4x4B = 1'b0;
207             enable_inputs8x8 = 1'b0;
208             enable_transpose_buffer8x8 = 1'b0;
209             change_transpose_buffer_direction8x8 = 1'b0;
210             reset_psatd8x8 = 1'b0;
211             enable_psatd8x8 = 1'b0;
212             reset_satd8x8 = 1'b0;
213             enable_satd8x8 = 1'b0;
214             reset_transpose_buffer8x8 = 1'b0;
215             done8x8 = 1'b0;
216         end
217         INITIALIZE_5:begin
```

```
218             enable_inputs4x4 = 1'b1;
219             enable_transpose_buffer4x4 = 1'b1;
220             change_transpose_buffer_direction4x4 = 1'b0;
221             reset_psatd4x4 = 1'b0;
222             enable_psatd4x4 = 1'b1;
223             reset_satd4x4 = 1'b0;
224             enable_satd4x4 = 1'b0;
225             reset_transpose_buffer4x4 = 1'b0;
226             done4x4 = 1'b0;
227             done4x4B = 1'b0;
228             enable_inputs8x8 = 1'b1;
229             enable_transpose_buffer8x8 = 1'b1;
230             change_transpose_buffer_direction8x8 = 1'b0;
231             reset_psatd8x8 = 1'b0;
232             enable_psatd8x8 = 1'b0;
233             reset_satd8x8 = 1'b0;
234             enable_satd8x8 = 1'b0;
235             reset_transpose_buffer8x8 = 1'b0;
236             done8x8 = 1'b0;
237         end
238         INITIALIZE_6:begin
239             enable_inputs4x4 = 1'b1;
240             enable_transpose_buffer4x4 = 1'b1;
241             change_transpose_buffer_direction4x4 = 1'b0;
242             reset_psatd4x4 = 1'b0;
243             enable_psatd4x4 = 1'b1;
244             reset_satd4x4 = 1'b0;
245             enable_satd4x4 = 1'b0;
246             reset_transpose_buffer4x4 = 1'b0;
247             done4x4 = 1'b0;
248             done4x4B = 1'b0;
249             enable_inputs8x8 = 1'b1;
250             enable_transpose_buffer8x8 = 1'b1;
251             change_transpose_buffer_direction8x8 = 1'b0;
252             reset_psatd8x8 = 1'b0;
253             enable_psatd8x8 = 1'b0;
254             reset_satd8x8 = 1'b0;
255             enable_satd8x8 = 1'b0;
256             reset_transpose_buffer8x8 = 1'b0;
257             done8x8 = 1'b0;
258         end
259         INITIALIZE_7:begin
260             enable_inputs4x4 = 1'b1;
261             enable_transpose_buffer4x4 = 1'b1;
262             change_transpose_buffer_direction4x4 = 1'b0;
263             reset_psatd4x4 = 1'b0;
264             enable_psatd4x4 = 1'b1;
265             reset_satd4x4 = 1'b0;
266             enable_satd4x4 = 1'b0;
267             reset_transpose_buffer4x4 = 1'b0;
268             done4x4 = 1'b0;
269             done4x4B = 1'b0;
270             enable_inputs8x8 = 1'b1;
271             enable_transpose_buffer8x8 = 1'b1;
272             change_transpose_buffer_direction8x8 = 1'b0;
273             reset_psatd8x8 = 1'b0;
274             enable_psatd8x8 = 1'b0;
275             reset_satd8x8 = 1'b0;
276             enable_satd8x8 = 1'b0;
277             reset_transpose_buffer8x8 = 1'b0;
278             done8x8 = 1'b0;
```

```
279             end
280             INITIALIZE_8:begin
281                 enable_inputs4x4 = 1'b1;
282                 enable_transpose_buffer4x4 = 1'b1;
283                 change_transpose_buffer_direction4x4 = 1'b1;
284                 reset_psatd4x4 = 1'b1;
285                 enable_psatd4x4 = 1'b0;
286                 reset_satd4x4 = 1'b0;
287                 enable_satd4x4 = 1'b1;
288                 reset_transpose_buffer4x4 = 1'b0;
289                 done4x4 = 1'b0;
290                 done4x4B = 1'b0;
291                 enable_inputs8x8 = 1'b1;
292                 enable_transpose_buffer8x8 = 1'b1;
293                 change_transpose_buffer_direction8x8 = 1'b0;
294                 reset_psatd8x8 = 1'b1;
295                 enable_psatd8x8 = 1'b0;
296                 reset_satd8x8 = 1'b0;
297                 enable_satd8x8 = 1'b0;
298                 reset_transpose_buffer8x8 = 1'b0;
299                 done8x8 = 1'b0;
300             end
301             INITIALIZE_9:begin
302                 enable_inputs4x4 = 1'b1;
303                 enable_transpose_buffer4x4 = 1'b1;
304                 change_transpose_buffer_direction4x4 = 1'b0;
305                 reset_psatd4x4 = 1'b0;
306                 enable_psatd4x4 = 1'b1;
307                 reset_satd4x4 = 1'b0;
308                 enable_satd4x4 = 1'b0;
309                 reset_transpose_buffer4x4 = 1'b0;
310                 done4x4 = 1'b1;
311                 done4x4B = 1'b1;
312                 enable_inputs8x8 = 1'b1;
313                 enable_transpose_buffer8x8 = 1'b1;
314                 change_transpose_buffer_direction8x8 = 1'b0;
315                 reset_psatd8x8 = 1'b0;
316                 enable_psatd8x8 = 1'b1;
317                 reset_satd8x8 = 1'b0;
318                 enable_satd8x8 = 1'b0;
319                 reset_transpose_buffer8x8 = 1'b0;
320                 done8x8 = 1'b0;
321             end
322             INITIALIZE_10:begin
323                 enable_inputs4x4 = 1'b1;
324                 enable_transpose_buffer4x4 = 1'b1;
325                 change_transpose_buffer_direction4x4 = 1'b0;
326                 reset_psatd4x4 = 1'b0;
327                 enable_psatd4x4 = 1'b1;
328                 reset_satd4x4 = 1'b0;
329                 enable_satd4x4 = 1'b0;
330                 reset_transpose_buffer4x4 = 1'b0;
331                 done4x4 = 1'b0;
332                 done4x4B = 1'b0;
333                 enable_inputs8x8 = 1'b1;
334                 enable_transpose_buffer8x8 = 1'b1;
335                 change_transpose_buffer_direction8x8 = 1'b0;
336                 reset_psatd8x8 = 1'b0;
337                 enable_psatd8x8 = 1'b1;
338                 reset_satd8x8 = 1'b0;
339                 enable_satd8x8 = 1'b0;
```

```
340             reset_transpose_buffer8x8 = 1'b0;
341             done8x8 = 1'b0;
342         end
343         INITIALIZE_11:begin
344             enable_inputs4x4 = 1'b1;
345             enable_transpose_buffer4x4 = 1'b1;
346             change_transpose_buffer_direction4x4 = 1'b0;
347             reset_psatd4x4 = 1'b0;
348             enable_psatd4x4 = 1'b1;
349             reset_satd4x4 = 1'b0;
350             enable_satd4x4 = 1'b0;
351             reset_transpose_buffer4x4 = 1'b0;
352             done4x4 = 1'b0;
353             done4x4B = 1'b0;
354             enable_inputs8x8 = 1'b1;
355             enable_transpose_buffer8x8 = 1'b1;
356             change_transpose_buffer_direction8x8 = 1'b0;
357             reset_psatd8x8 = 1'b0;
358             enable_psatd8x8 = 1'b1;
359             reset_satd8x8 = 1'b0;
360             enable_satd8x8 = 1'b0;
361             reset_transpose_buffer8x8 = 1'b0;
362             done8x8 = 1'b0;
363         end
364         INITIALIZE_12:begin
365             enable_inputs4x4 = 1'b1;
366             enable_transpose_buffer4x4 = 1'b1;
367             change_transpose_buffer_direction4x4 = 1'b1;
368             reset_psatd4x4 = 1'b1;
369             enable_psatd4x4 = 1'b0;
370             reset_satd4x4 = 1'b0;
371             enable_satd4x4 = 1'b1;
372             reset_transpose_buffer4x4 = 1'b0;
373             done4x4 = 1'b0;
374             done4x4B = 1'b0;
375             enable_inputs8x8 = 1'b1;
376             enable_transpose_buffer8x8 = 1'b1;
377             change_transpose_buffer_direction8x8 = 1'b1;
378             reset_psatd8x8 = 1'b0;
379             enable_psatd8x8 = 1'b1;
380             reset_satd8x8 = 1'b0;
381             enable_satd8x8 = 1'b0;
382             reset_transpose_buffer8x8 = 1'b0;
383             done8x8 = 1'b0;
384         end
385         INITIALIZE_13:begin
386             enable_inputs4x4 = 1'b1;
387             enable_transpose_buffer4x4 = 1'b1;
388             change_transpose_buffer_direction4x4 = 1'b0;
389             reset_psatd4x4 = 1'b0;
390             enable_psatd4x4 = 1'b1;
391             reset_satd4x4 = 1'b0;
392             enable_satd4x4 = 1'b0;
393             reset_transpose_buffer4x4 = 1'b0;
394             done4x4 = 1'b1;
395             done4x4B = 1'b1;
396             enable_inputs8x8 = 1'b1;
397             enable_transpose_buffer8x8 = 1'b1;
398             change_transpose_buffer_direction8x8 = 1'b0;
399             reset_psatd8x8 = 1'b0;
400             enable_psatd8x8 = 1'b1;
```

```
401                 reset_satd8x8 = 1'b0;
402                 enable_satd8x8 = 1'b0;
403                 reset_transpose_buffer8x8 = 1'b0;
404                 done8x8 = 1'b0;
405             end
406             INITIALIZE_14:begin
407                 enable_inputs4x4 = 1'b1;
408                 enable_transpose_buffer4x4 = 1'b1;
409                 change_transpose_buffer_direction4x4 = 1'b0;
410                 reset_psatd4x4 = 1'b0;
411                 enable_psatd4x4 = 1'b1;
412                 reset_satd4x4 = 1'b0;
413                 enable_satd4x4 = 1'b0;
414                 reset_transpose_buffer4x4 = 1'b0;
415                 done4x4 = 1'b0;
416                 done4x4B = 1'b0;
417                 enable_inputs8x8 = 1'b1;
418                 enable_transpose_buffer8x8 = 1'b1;
419                 change_transpose_buffer_direction8x8 = 1'b0;
420                 reset_psatd8x8 = 1'b0;
421                 enable_psatd8x8 = 1'b1;
422                 reset_satd8x8 = 1'b0;
423                 enable_satd8x8 = 1'b0;
424                 reset_transpose_buffer8x8 = 1'b0;
425                 done8x8 = 1'b0;
426             end
427             INITIALIZE_15:begin
428                 enable_inputs4x4 = 1'b1;
429                 enable_transpose_buffer4x4 = 1'b1;
430                 change_transpose_buffer_direction4x4 = 1'b0;
431                 reset_psatd4x4 = 1'b0;
432                 enable_psatd4x4 = 1'b1;
433                 reset_satd4x4 = 1'b0;
434                 enable_satd4x4 = 1'b0;
435                 reset_transpose_buffer4x4 = 1'b0;
436                 done4x4 = 1'b0;
437                 done4x4B = 1'b0;
438                 enable_inputs8x8 = 1'b1;
439                 enable_transpose_buffer8x8 = 1'b1;
440                 change_transpose_buffer_direction8x8 = 1'b0;
441                 reset_psatd8x8 = 1'b0;
442                 enable_psatd8x8 = 1'b1;
443                 reset_satd8x8 = 1'b0;
444                 enable_satd8x8 = 1'b0;
445                 reset_transpose_buffer8x8 = 1'b0;
446                 done8x8 = 1'b0;
447             end
448             INITIALIZE_16:begin
449                 enable_inputs4x4 = 1'b1;
450                 enable_transpose_buffer4x4 = 1'b1;
451                 change_transpose_buffer_direction4x4 = 1'b1;
452                 reset_psatd4x4 = 1'b1;
453                 enable_psatd4x4 = 1'b0;
454                 reset_satd4x4 = 1'b0;
455                 enable_satd4x4 = 1'b1;
456                 reset_transpose_buffer4x4 = 1'b0;
457                 done4x4 = 1'b0;
458                 done4x4B = 1'b0;
459                 enable_inputs8x8 = 1'b1;
460                 enable_transpose_buffer8x8 = 1'b1;
461                 change_transpose_buffer_direction8x8 = 1'b0;
```

```
462             reset_psatd8x8 = 1'b0;
463             enable_psatd8x8 = 1'b1;
464             reset_satd8x8 = 1'b0;
465             enable_satd8x8 = 1'b0;
466             reset_transpose_buffer8x8 = 1'b0;
467             done8x8 = 1'b0;
468         end
469         INITIALIZE_17:begin
470             enable_inputs4x4 = 1'b1;
471             enable_transpose_buffer4x4 = 1'b1;
472             change_transpose_buffer_direction4x4 = 1'b0;
473             reset_psatd4x4 = 1'b0;
474             enable_psatd4x4 = 1'b1;
475             reset_satd4x4 = 1'b0;
476             enable_satd4x4 = 1'b0;
477             reset_transpose_buffer4x4 = 1'b0;
478             done4x4 = 1'b1;
479             done4x4B = 1'b1;
480             enable_inputs8x8 = 1'b1;
481             enable_transpose_buffer8x8 = 1'b1;
482             change_transpose_buffer_direction8x8 = 1'b0;
483             reset_psatd8x8 = 1'b0;
484             enable_psatd8x8 = 1'b1;
485             reset_satd8x8 = 1'b0;
486             enable_satd8x8 = 1'b0;
487             reset_transpose_buffer8x8 = 1'b0;
488             done8x8 = 1'b0;
489         end
490         INITIALIZE_18:begin
491             enable_inputs4x4 = 1'b1;
492             enable_transpose_buffer4x4 = 1'b1;
493             change_transpose_buffer_direction4x4 = 1'b0;
494             reset_psatd4x4 = 1'b0;
495             enable_psatd4x4 = 1'b1;
496             reset_satd4x4 = 1'b0;
497             enable_satd4x4 = 1'b0;
498             reset_transpose_buffer4x4 = 1'b0;
499             done4x4 = 1'b0;
500             done4x4B = 1'b0;
501             enable_inputs8x8 = 1'b1;
502             enable_transpose_buffer8x8 = 1'b1;
503             change_transpose_buffer_direction8x8 = 1'b0;
504             reset_psatd8x8 = 1'b0;
505             enable_psatd8x8 = 1'b1;
506             reset_satd8x8 = 1'b0;
507             enable_satd8x8 = 1'b0;
508             reset_transpose_buffer8x8 = 1'b0;
509             done8x8 = 1'b0;
510         end
511         INITIALIZE_19:begin
512             enable_inputs4x4 = 1'b1;
513             enable_transpose_buffer4x4 = 1'b1;
514             change_transpose_buffer_direction4x4 = 1'b0;
515             reset_psatd4x4 = 1'b0;
516             enable_psatd4x4 = 1'b1;
517             reset_satd4x4 = 1'b0;
518             enable_satd4x4 = 1'b0;
519             reset_transpose_buffer4x4 = 1'b0;
520             done4x4 = 1'b0;
521             done4x4B = 1'b0;
522             enable_inputs8x8 = 1'b1;
```

```
523                 enable_transpose_buffer8x8 = 1'b1;
524                 change_transpose_buffer_direction8x8 = 1'b0;
525                 reset_psatd8x8 = 1'b0;
526                 enable_psatd8x8 = 1'b1;
527                 reset_satd8x8 = 1'b0;
528                 enable_satd8x8 = 1'b0;
529                 reset_transpose_buffer8x8 = 1'b0;
530                 done8x8 = 1'b0;
531             end
532             PRE_CALCULATE_0:begin
533                 enable_inputs4x4 = 1'b1;
534                 enable_transpose_buffer4x4 = 1'b1;
535                 change_transpose_buffer_direction4x4 = 1'b1;
536                 reset_psatd4x4 = 1'b1;
537                 enable_psatd4x4 = 1'b0;
538                 reset_satd4x4 = 1'b0;
539                 enable_satd4x4 = 1'b1;
540                 reset_transpose_buffer4x4 = 1'b0;
541                 done4x4 = 1'b0;
542                 done4x4B = 1'b0;
543                 enable_inputs8x8 = 1'b1;
544                 enable_transpose_buffer8x8 = 1'b1;
545                 change_transpose_buffer_direction8x8 = 1'b1;
546                 reset_psatd8x8 = 1'b1;
547                 enable_psatd8x8 = 1'b0;
548                 reset_satd8x8 = 1'b0;
549                 enable_satd8x8 = 1'b1;
550                 reset_transpose_buffer8x8 = 1'b0;
551                 done8x8 = 1'b0;
552             end
553             PRE_CALCULATE_1:begin
554                 enable_inputs4x4 = 1'b1;
555                 enable_transpose_buffer4x4 = 1'b1;
556                 change_transpose_buffer_direction4x4 = 1'b0;
557                 reset_psatd4x4 = 1'b0;
558                 enable_psatd4x4 = 1'b1;
559                 reset_satd4x4 = 1'b0;
560                 enable_satd4x4 = 1'b0;
561                 reset_transpose_buffer4x4 = 1'b0;
562                 done4x4 = 1'b1;
563                 done4x4B = 1'b1;
564                 enable_inputs8x8 = 1'b1;
565                 enable_transpose_buffer8x8 = 1'b1;
566                 change_transpose_buffer_direction8x8 = 1'b0;
567                 reset_psatd8x8 = 1'b0;
568                 enable_psatd8x8 = 1'b1;
569                 reset_satd8x8 = 1'b0;
570                 enable_satd8x8 = 1'b0;
571                 reset_transpose_buffer8x8 = 1'b0;
572                 done8x8 = 1'b1;
573             end
574             CALCULATE_0:begin
575                 enable_inputs4x4 = 1'b1;
576                 enable_transpose_buffer4x4 = 1'b1;
577                 change_transpose_buffer_direction4x4 = 1'b1;
578                 reset_psatd4x4 = 1'b1;
579                 enable_psatd4x4 = 1'b0;
580                 reset_satd4x4 = 1'b0;
581                 enable_satd4x4 = 1'b1;
582                 reset_transpose_buffer4x4 = 1'b0;
583                 done4x4 = 1'b0;
```

```
584                 done4x4B = 1'b0;
585                 enable_inputs8x8 = 1'b1;
586                 enable_transpose_buffer8x8 = 1'b1;
587                 change_transpose_buffer_direction8x8 = 1'b1;
588                 reset_psatd8x8 = 1'b1;
589                 enable_psatd8x8 = 1'b0;
590                 reset_satd8x8 = 1'b0;
591                 enable_satd8x8 = 1'b1;
592                 reset_transpose_buffer8x8 = 1'b0;
593                 done8x8 = 1'b0;
594             end
595             CALCULATE_1:begin
596                 enable_inputs4x4 = 1'b1;
597                 enable_transpose_buffer4x4 = 1'b1;
598                 change_transpose_buffer_direction4x4 = 1'b0;
599                 reset_psatd4x4 = 1'b0;
600                 enable_psatd4x4 = 1'b1;
601                 reset_satd4x4 = 1'b0;
602                 enable_satd4x4 = 1'b0;
603                 reset_transpose_buffer4x4 = 1'b0;
604                 done4x4 = 1'b1;
605                 done4x4B = 1'b1;
606                 enable_inputs8x8 = 1'b1;
607                 enable_transpose_buffer8x8 = 1'b1;
608                 change_transpose_buffer_direction8x8 = 1'b0;
609                 reset_psatd8x8 = 1'b0;
610                 enable_psatd8x8 = 1'b1;
611                 reset_satd8x8 = 1'b0;
612                 enable_satd8x8 = 1'b0;
613                 reset_transpose_buffer8x8 = 1'b0;
614                 done8x8 = 1'b1;
615             end
616             CALCULATE_2:begin
617                 enable_inputs4x4 = 1'b1;
618                 enable_transpose_buffer4x4 = 1'b1;
619                 change_transpose_buffer_direction4x4 = 1'b0;
620                 reset_psatd4x4 = 1'b0;
621                 enable_psatd4x4 = 1'b1;
622                 reset_satd4x4 = 1'b0;
623                 enable_satd4x4 = 1'b0;
624                 reset_transpose_buffer4x4 = 1'b0;
625                 done4x4 = 1'b0;
626                 done4x4B = 1'b0;
627                 enable_inputs8x8 = 1'b1;
628                 enable_transpose_buffer8x8 = 1'b1;
629                 change_transpose_buffer_direction8x8 = 1'b0;
630                 reset_psatd8x8 = 1'b0;
631                 enable_psatd8x8 = 1'b1;
632                 reset_satd8x8 = 1'b0;
633                 enable_satd8x8 = 1'b0;
634                 reset_transpose_buffer8x8 = 1'b0;
635                 done8x8 = 1'b0;
636             end
637             CALCULATE_3:begin
638                 enable_inputs4x4 = 1'b1;
639                 enable_transpose_buffer4x4 = 1'b1;
640                 change_transpose_buffer_direction4x4 = 1'b0;
641                 reset_psatd4x4 = 1'b0;
642                 enable_psatd4x4 = 1'b1;
643                 reset_satd4x4 = 1'b0;
644                 enable_satd4x4 = 1'b0;
```

```
645                 reset_transpose_buffer4x4 = 1'b0;
646                 done4x4 = 1'b0;
647                 done4x4B = 1'b0;
648                 enable_inputs8x8 = 1'b1;
649                 enable_transpose_buffer8x8 = 1'b1;
650                 change_transpose_buffer_direction8x8 = 1'b0;
651                 reset_psatd8x8 = 1'b0;
652                 enable_psatd8x8 = 1'b1;
653                 reset_satd8x8 = 1'b0;
654                 enable_satd8x8 = 1'b0;
655                 reset_transpose_buffer8x8 = 1'b0;
656                 done8x8 = 1'b0;
657             end
658             CALCULATE_4:begin
659                 enable_inputs4x4 = 1'b1;
660                 enable_transpose_buffer4x4 = 1'b1;
661                 change_transpose_buffer_direction4x4 = 1'b1;
662                 reset_psatd4x4 = 1'b1;
663                 enable_psatd4x4 = 1'b0;
664                 reset_satd4x4 = 1'b0;
665                 enable_satd4x4 = 1'b1;
666                 reset_transpose_buffer4x4 = 1'b0;
667                 done4x4 = 1'b0;
668                 done4x4B = 1'b0;
669                 enable_inputs8x8 = 1'b1;
670                 enable_transpose_buffer8x8 = 1'b1;
671                 change_transpose_buffer_direction8x8 = 1'b0;
672                 reset_psatd8x8 = 1'b0;
673                 enable_psatd8x8 = 1'b1;
674                 reset_satd8x8 = 1'b0;
675                 enable_satd8x8 = 1'b0;
676                 reset_transpose_buffer8x8 = 1'b0;
677                 done8x8 = 1'b0;
678             end
679             CALCULATE_5:begin
680                 enable_inputs4x4 = 1'b1;
681                 enable_transpose_buffer4x4 = 1'b1;
682                 change_transpose_buffer_direction4x4 = 1'b0;
683                 reset_psatd4x4 = 1'b0;
684                 enable_psatd4x4 = 1'b1;
685                 reset_satd4x4 = 1'b0;
686                 enable_satd4x4 = 1'b0;
687                 reset_transpose_buffer4x4 = 1'b0;
688                 done4x4 = 1'b1;
689                 done4x4B = 1'b1;
690                 enable_inputs8x8 = 1'b1;
691                 enable_transpose_buffer8x8 = 1'b1;
692                 change_transpose_buffer_direction8x8 = 1'b0;
693                 reset_psatd8x8 = 1'b0;
694                 enable_psatd8x8 = 1'b1;
695                 reset_satd8x8 = 1'b0;
696                 enable_satd8x8 = 1'b0;
697                 reset_transpose_buffer8x8 = 1'b0;
698                 done8x8 = 1'b0;
699             end
700             CALCULATE_6:begin
701                 enable_inputs4x4 = 1'b1;
702                 enable_transpose_buffer4x4 = 1'b1;
703                 change_transpose_buffer_direction4x4 = 1'b0;
704                 reset_psatd4x4 = 1'b0;
705                 enable_psatd4x4 = 1'b1;
```

```
706             reset_satd4x4 = 1'b0;
707             enable_satd4x4 = 1'b0;
708             reset_transpose_buffer4x4 = 1'b0;
709             done4x4 = 1'b0;
710             done4x4B = 1'b0;
711             enable_inputs8x8 = 1'b1;
712             enable_transpose_buffer8x8 = 1'b1;
713             change_transpose_buffer_direction8x8 = 1'b0;
714             reset_psatd8x8 = 1'b0;
715             enable_psatd8x8 = 1'b1;
716             reset_satd8x8 = 1'b0;
717             enable_satd8x8 = 1'b0;
718             reset_transpose_buffer8x8 = 1'b0;
719             done8x8 = 1'b0;
720         end
721         CALCULATE_7:begin
722             enable_inputs4x4 = 1'b1;
723             enable_transpose_buffer4x4 = 1'b1;
724             change_transpose_buffer_direction4x4 = 1'b0;
725             reset_psatd4x4 = 1'b0;
726             enable_psatd4x4 = 1'b1;
727             reset_satd4x4 = 1'b0;
728             enable_satd4x4 = 1'b0;
729             reset_transpose_buffer4x4 = 1'b0;
730             done4x4 = 1'b0;
731             done4x4B = 1'b0;
732             enable_inputs8x8 = 1'b1;
733             enable_transpose_buffer8x8 = 1'b1;
734             change_transpose_buffer_direction8x8 = 1'b0;
735             reset_psatd8x8 = 1'b0;
736             enable_psatd8x8 = 1'b1;
737             reset_satd8x8 = 1'b0;
738             enable_satd8x8 = 1'b0;
739             reset_transpose_buffer8x8 = 1'b0;
740             done8x8 = 1'b0;
741         end
742         TERMINATE_0:begin
743             enable_inputs4x4 = 1'b0;
744             enable_transpose_buffer4x4 = 1'b1;
745             change_transpose_buffer_direction4x4 = 1'b0;
746             reset_psatd4x4 = 1'b0;
747             enable_psatd4x4 = 1'b1;
748             reset_satd4x4 = 1'b0;
749             enable_satd4x4 = 1'b0;
750             reset_transpose_buffer4x4 = 1'b0;
751             done4x4 = 1'b0;
752             done4x4B = 1'b0;
753             enable_inputs8x8 = 1'b1;
754             enable_transpose_buffer8x8 = 1'b0;
755             change_transpose_buffer_direction8x8 = 1'b0;
756             reset_psatd8x8 = 1'b0;
757             enable_psatd8x8 = 1'b1;
758             reset_satd8x8 = 1'b0;
759             enable_satd8x8 = 1'b0;
760             reset_transpose_buffer8x8 = 1'b0;
761             done8x8 = 1'b1;
762         end
763         TERMINATE_1:begin
764             enable_inputs4x4 = 1'b0;
765             enable_transpose_buffer4x4 = 1'b0;
766             change_transpose_buffer_direction4x4 = 1'b0;
```

```
767                 reset_psatd4x4 = 1'b0;
768                 enable_psatd4x4 = 1'b0;
769                 reset_satd4x4 = 1'b0;
770                 enable_satd4x4 = 1'b1;
771                 reset_transpose_buffer4x4 = 1'b1;
772                 done4x4 = 1'b0;
773                 done4x4B = 1'b0;
774                 enable_inputs8x8 = 1'b0;
775                 enable_transpose_buffer8x8 = 1'b1;
776                 change_transpose_buffer_direction8x8 = 1'b0;
777                 reset_psatd8x8 = 1'b1;
778                 enable_psatd8x8 = 1'b0;
779                 reset_satd8x8 = 1'b0;
780                 enable_satd8x8 = 1'b1;
781                 reset_transpose_buffer8x8 = 1'b0;
782                 done8x8 = 1'b0;
783             end
784             TERMINATE_2:begin
785                 enable_inputs4x4 = 1'b0;
786                 enable_transpose_buffer4x4 = 1'b0;
787                 change_transpose_buffer_direction4x4 = 1'b0;
788                 reset_psatd4x4 = 1'b0;
789                 enable_psatd4x4 = 1'b0;
790                 reset_satd4x4 = 1'b0;
791                 enable_satd4x4 = 1'b0;
792                 reset_transpose_buffer4x4 = 1'b0;
793                 done4x4 = 1'b0;
794                 done4x4B = 1'b0;
795                 enable_inputs8x8 = 1'b0;
796                 enable_transpose_buffer8x8 = 1'b1;
797                 change_transpose_buffer_direction8x8 = 1'b0;
798                 reset_psatd8x8 = 1'b0;
799                 enable_psatd8x8 = 1'b1;
800                 reset_satd8x8 = 1'b0;
801                 enable_satd8x8 = 1'b0;
802                 reset_transpose_buffer8x8 = 1'b0;
803                 done8x8 = 1'b0;
804             end
805             TERMINATE_3:begin
806                 enable_inputs4x4 = 1'b0;
807                 enable_transpose_buffer4x4 = 1'b0;
808                 change_transpose_buffer_direction4x4 = 1'b0;
809                 reset_psatd4x4 = 1'b0;
810                 enable_psatd4x4 = 1'b0;
811                 reset_satd4x4 = 1'b0;
812                 enable_satd4x4 = 1'b0;
813                 reset_transpose_buffer4x4 = 1'b0;
814                 done4x4 = 1'b1;
815                 done4x4B = 1'b1;
816                 enable_inputs8x8 = 1'b0;
817                 enable_transpose_buffer8x8 = 1'b1;
818                 change_transpose_buffer_direction8x8 = 1'b0;
819                 reset_psatd8x8 = 1'b0;
820                 enable_psatd8x8 = 1'b0;
821                 reset_satd8x8 = 1'b1;
822                 enable_satd8x8 = 1'b0;
823                 reset_transpose_buffer8x8 = 1'b0;
824                 done8x8 = 1'b0;
825             end
826             TERMINATE_4:begin
827                 enable_inputs4x4 = 1'b0;
```

```
828             enable_transpose_buffer4x4 = 1'b0;
829             change_transpose_buffer_direction4x4 = 1'b0;
830             reset_psatd4x4 = 1'b0;
831             enable_psatd4x4 = 1'b0;
832             reset_satd4x4 = 1'b0;
833             enable_satd4x4 = 1'b0;
834             reset_transpose_buffer4x4 = 1'b0;
835             done4x4 = 1'b0;
836             done4x4B = 1'b0;
837             enable_inputs8x8 = 1'b0;
838             enable_transpose_buffer8x8 = 1'b1;
839             change_transpose_buffer_direction8x8 = 1'b0;
840             reset_psatd8x8 = 1'b0;
841             enable_psatd8x8 = 1'b1;
842             reset_satd8x8 = 1'b0;
843             enable_satd8x8 = 1'b0;
844             reset_transpose_buffer8x8 = 1'b0;
845             done8x8 = 1'b0;
846         end
847         TERMINATE_5:begin
848             enable_inputs4x4 = 1'b0;
849             enable_transpose_buffer4x4 = 1'b0;
850             change_transpose_buffer_direction4x4 = 1'b0;
851             reset_psatd4x4 = 1'b0;
852             enable_psatd4x4 = 1'b0;
853             reset_satd4x4 = 1'b0;
854             enable_satd4x4 = 1'b0;
855             reset_transpose_buffer4x4 = 1'b0;
856             done4x4 = 1'b0;
857             done4x4B = 1'b0;
858             enable_inputs8x8 = 1'b0;
859             enable_transpose_buffer8x8 = 1'b1;
860             change_transpose_buffer_direction8x8 = 1'b0;
861             reset_psatd8x8 = 1'b0;
862             enable_psatd8x8 = 1'b1;
863             reset_satd8x8 = 1'b0;
864             enable_satd8x8 = 1'b0;
865             reset_transpose_buffer8x8 = 1'b0;
866             done8x8 = 1'b0;
867         end
868         TERMINATE_6:begin
869             enable_inputs4x4 = 1'b0;
870             enable_transpose_buffer4x4 = 1'b0;
871             change_transpose_buffer_direction4x4 = 1'b0;
872             reset_psatd4x4 = 1'b0;
873             enable_psatd4x4 = 1'b0;
874             reset_satd4x4 = 1'b0;
875             enable_satd4x4 = 1'b0;
876             reset_transpose_buffer4x4 = 1'b0;
877             done4x4 = 1'b0;
878             done4x4B = 1'b0;
879             enable_inputs8x8 = 1'b0;
880             enable_transpose_buffer8x8 = 1'b1;
881             change_transpose_buffer_direction8x8 = 1'b0;
882             reset_psatd8x8 = 1'b0;
883             enable_psatd8x8 = 1'b1;
884             reset_satd8x8 = 1'b0;
885             enable_satd8x8 = 1'b0;
886             reset_transpose_buffer8x8 = 1'b0;
887             done8x8 = 1'b0;
888         end
```

```
889             TERMINATE_7:begin
890                 enable_inputs4x4 = 1'b0;
891                 enable_transpose_buffer4x4 = 1'b0;
892                 change_transpose_buffer_direction4x4 = 1'b0;
893                 reset_psatd4x4 = 1'b0;
894                 enable_psatd4x4 = 1'b0;
895                 reset_satd4x4 = 1'b0;
896                 enable_satd4x4 = 1'b0;
897                 reset_transpose_buffer4x4 = 1'b0;
898                 done4x4 = 1'b0;
899                 done4x4B = 1'b0;
900                 enable_inputs8x8 = 1'b0;
901                 enable_transpose_buffer8x8 = 1'b1;
902                 change_transpose_buffer_direction8x8 = 1'b1;
903                 reset_psatd8x8 = 1'b1;
904                 enable_psatd8x8 = 1'b0;
905                 reset_satd8x8 = 1'b0;
906                 enable_satd8x8 = 1'b0;
907                 reset_transpose_buffer8x8 = 1'b0;
908                 done8x8 = 1'b0;
909             end
910             LAST_0:begin
911                 enable_inputs4x4 = 1'b0;
912                 enable_transpose_buffer4x4 = 1'b0;
913                 change_transpose_buffer_direction4x4 = 1'b0;
914                 reset_psatd4x4 = 1'b0;
915                 enable_psatd4x4 = 1'b0;
916                 reset_satd4x4 = 1'b0;
917                 enable_satd4x4 = 1'b0;
918                 reset_transpose_buffer4x4 = 1'b0;
919                 done4x4 = 1'b0;
920                 done4x4B = 1'b0;
921                 enable_inputs8x8 = 1'b0;
922                 enable_transpose_buffer8x8 = 1'b1;
923                 change_transpose_buffer_direction8x8 = 1'b0;
924                 reset_psatd8x8 = 1'b0;
925                 enable_psatd8x8 = 1'b1;
926                 reset_satd8x8 = 1'b0;
927                 enable_satd8x8 = 1'b0;
928                 reset_transpose_buffer8x8 = 1'b0;
929                 done8x8 = 1'b1;
930             end
931             LAST_1:begin
932                 enable_inputs4x4 = 1'b0;
933                 enable_transpose_buffer4x4 = 1'b0;
934                 change_transpose_buffer_direction4x4 = 1'b0;
935                 reset_psatd4x4 = 1'b0;
936                 enable_psatd4x4 = 1'b0;
937                 reset_satd4x4 = 1'b0;
938                 enable_satd4x4 = 1'b0;
939                 reset_transpose_buffer4x4 = 1'b0;
940                 done4x4 = 1'b0;
941                 done4x4B = 1'b0;
942                 enable_inputs8x8 = 1'b0;
943                 enable_transpose_buffer8x8 = 1'b0;
944                 change_transpose_buffer_direction8x8 = 1'b0;
945                 reset_psatd8x8 = 1'b0;
946                 enable_psatd8x8 = 1'b0;
947                 reset_satd8x8 = 1'b0;
948                 enable_satd8x8 = 1'b1;
949                 reset_transpose_buffer8x8 = 1'b0;
```

```
950                 done8x8 = 1'b0;
951             end
952             PRE_TERMINATE_0:begin
953                 enable_inputs4x4 = 1'b0;
954                 enable_transpose_buffer4x4 = 1'b0;
955                 change_transpose_buffer_direction4x4 = 1'b0;
956                 reset_psatd4x4 = 1'b0;
957                 enable_psatd4x4 = 1'b0;
958                 reset_satd4x4 = 1'b0;
959                 enable_satd4x4 = 1'b0;
960                 reset_transpose_buffer4x4 = 1'b0;
961                 done4x4 = 1'b0;
962                 done4x4B = 1'b0;
963                 enable_inputs8x8 = 1'b0;
964                 enable_transpose_buffer8x8 = 1'b0;
965                 change_transpose_buffer_direction8x8 = 1'b0;
966                 reset_psatd8x8 = 1'b0;
967                 enable_psatd8x8 = 1'b0;
968                 reset_satd8x8 = 1'b0;
969                 enable_satd8x8 = 1'b0;
970                 reset_transpose_buffer8x8 = 1'b0;
971                 done8x8 = 1'b0;
972             end
973             PRE_TERMINATE_1:begin
974                 enable_inputs4x4 = 1'b0;
975                 enable_transpose_buffer4x4 = 1'b0;
976                 change_transpose_buffer_direction4x4 = 1'b0;
977                 reset_psatd4x4 = 1'b0;
978                 enable_psatd4x4 = 1'b0;
979                 reset_satd4x4 = 1'b0;
980                 enable_satd4x4 = 1'b0;
981                 reset_transpose_buffer4x4 = 1'b0;
982                 done4x4 = 1'b0;
983                 done4x4B = 1'b0;
984                 enable_inputs8x8 = 1'b0;
985                 enable_transpose_buffer8x8 = 1'b0;
986                 change_transpose_buffer_direction8x8 = 1'b0;
987                 reset_psatd8x8 = 1'b0;
988                 enable_psatd8x8 = 1'b0;
989                 reset_satd8x8 = 1'b0;
990                 enable_satd8x8 = 1'b0;
991                 reset_transpose_buffer8x8 = 1'b0;
992                 done8x8 = 1'b0;
993             end
994             default:begin
995                 enable_inputs4x4 = 1'b0;
996                 enable_transpose_buffer4x4 = 1'b0;
997                 change_transpose_buffer_direction4x4 = 1'b0;
998                 reset_psatd4x4 = 1'b0;
999                 enable_psatd4x4 = 1'b0;
1000                reset_satd4x4 = 1'b0;
1001                enable_satd4x4 = 1'b0;
1002                reset_transpose_buffer4x4 = 1'b0;
1003                done4x4 = 1'b0;
1004                done4x4B = 1'b0;
1005                enable_inputs8x8 = 1'b0;
1006                enable_transpose_buffer8x8 = 1'b0;
1007                change_transpose_buffer_direction8x8 = 1'b0;
1008                reset_psatd8x8 = 1'b0;
1009                enable_psatd8x8 = 1'b0;
1010                reset_satd8x8 = 1'b0;
```

```
1011                enable_satd8x8 = 1'b0;
1012                reset_transpose_buffer8x8 = 1'b0;
1013                done8x8 = 1'b0;
1014            end
1015        endcase
1016    end
1017
1018    always @ (posedge clock or posedge reset) begin
1019        if (reset)
1020            state <= IDLE;
1021        else
1022            case (state)
1023                IDLE: if(enable)
1024                    state<= INITIALIZE_0;
1025                INITIALIZE_0: state<=INITIALIZE_1;
1026                INITIALIZE_1: state<=INITIALIZE_2;
1027                INITIALIZE_2: state<=INITIALIZE_3;
1028                INITIALIZE_3: state<=INITIALIZE_4;
1029                INITIALIZE_4: state<=INITIALIZE_5;
1030                INITIALIZE_5: state<=INITIALIZE_6;
1031                INITIALIZE_6: state<=INITIALIZE_7;
1032                INITIALIZE_7: state<=INITIALIZE_8;
1033                INITIALIZE_8: state<=INITIALIZE_9;
1034                INITIALIZE_9: state<=INITIALIZE_10;
1035                INITIALIZE_10: state<=INITIALIZE_11;
1036
1037                INITIALIZE_11: state<=INITIALIZE_12;
1038                INITIALIZE_12: state<=INITIALIZE_13;
1039                INITIALIZE_13: state<=INITIALIZE_14;
1040                INITIALIZE_14: state<=INITIALIZE_15;
1041                INITIALIZE_15: state<=INITIALIZE_16;
1042                INITIALIZE_16: state<=INITIALIZE_17;
1043                INITIALIZE_17: state<=INITIALIZE_18;
1044                INITIALIZE_18: state<=INITIALIZE_19;
1045
1046                INITIALIZE_19: if(enable)
1047                    state<=PRE_CALCULATE_0;
1048                else
1049                    state<=PRE_TERMINATE_0;
1050                CALCULATE_0: state<=CALCULATE_1;
1051                CALCULATE_1: state<=CALCULATE_2;
1052                CALCULATE_2: state<=CALCULATE_3;
1053                CALCULATE_3: state<=CALCULATE_4;
1054                CALCULATE_4: state<=CALCULATE_5;
1055                CALCULATE_5: state<=CALCULATE_6;
1056                CALCULATE_6: state<=CALCULATE_7;
1057                CALCULATE_7: if(enable)
1058                    state<=CALCULATE_0;
1059                else
1060                    state<=TERMINATE_0;
1061                TERMINATE_0: state<= TERMINATE_1;
1062                TERMINATE_1: state<= TERMINATE_2;
1063                TERMINATE_2: state<= TERMINATE_3;
1064                TERMINATE_3: state<= TERMINATE_4;
1065                TERMINATE_4: state<= TERMINATE_5;
1066                TERMINATE_5: state<= TERMINATE_6;
1067                TERMINATE_6: state<= TERMINATE_7;
1068                TERMINATE_7: state<=LAST_0;
1069                PRE_CALCULATE_0: state<= PRE_CALCULATE_1;
1070                PRE_CALCULATE_1: state<= CALCULATE_2;
1071                LAST_0: state<= LAST_1;
```

```
            LAST_1: state<= IDLE;
            PRE_TERMINATE_0: state<=PRE_TERMINATE_1;
            PRE_TERMINATE_1: state<=TERMINATE_2;
        endcase
    end
endmodule
```

Listing A.25: Código fonte do primeiro ajuste

```
module transform_1d_8inputs_mod (
    in_0,
    in_1,
    in_2,
    in_3,
    in_4,
    in_5,
    in_6,
    in_7,
    out_0,
    out_1,
    out_2,
    out_3,
    out_4,
    out_5,
    out_6,
    out_7
);

parameter DATA_WIDTH = 8;

input signed [DATA_WIDTH-1:0] in_0, in_1, in_2, in_3, in_4, in_5,
    in_6, in_7;
output signed [DATA_WIDTH:0] out_0, out_1, out_2, out_3, out_4,
    out_5, out_6, out_7;

butterfly2 #(DATA_WIDTH) butterfly2_0_0(in_0, in_4, out_0, out_1);
butterfly2 #(DATA_WIDTH) butterfly2_0_2(in_1, in_5, out_2, out_3);
butterfly2 #(DATA_WIDTH) butterfly2_0_4(in_2, in_6, out_4, out_5);
butterfly2 #(DATA_WIDTH) butterfly2_0_6(in_3, in_7, out_6, out_7);

endmodule
```

Listing A.26: Código fonte do bloco operativo

```
//
    ------------------------------------------------------------------------

// Design Name : mainOperator
// File Name   : mainOperator.v
// Function    : SATD
// Coder       : Marcio Monteiro
//
    ------------------------------------------------------------------------

module mainOperative(
    clk,
    reset,
    enable,
    enable_inputs4x4, // controls the inputs of 4x4
```

```
12      reset_transpose_buffer4x4,
13      enable_transpose_buffer4x4,
14      change_transpose_buffer_direction4x4,
15      reset_psatd4x4,
16      enable_psatd4x4,
17      reset_satd4x4,
18      enable_satd4x4,
19      enable_inputs8x8,
20      reset_transpose_buffer8x8,
21      enable_transpose_buffer8x8,
22      change_transpose_buffer_direction8x8,
23      reset_psatd8x8,
24      enable_psatd8x8,
25      reset_satd8x8,
26      enable_satd8x8,
27      original_0,
28      original_1,
29      original_2,
30      original_3,
31      original_4,
32      original_5,
33      original_6,
34      original_7,
35      candidate_0,
36      candidate_1,
37      candidate_2,
38      candidate_3,
39      candidate_4,
40      candidate_5,
41      candidate_6,
42      candidate_7,
43      doneA,
44      satd4x4A,
45      doneB,
46      satd4x4B,
47      done,
48      satd8x8
49  );
50
51  parameter DATA_WIDTH = 8;
52
53  //------------------------------- Input Ports
        ---------------------------------
54  input clk, reset, enable;
55  input [DATA_WIDTH-1:0] original_0, original_1, original_2,
        original_3, original_4, original_5, original_6, original_7,
        candidate_0, candidate_1, candidate_2, candidate_3, candidate_4,
         candidate_5, candidate_6, candidate_7;
56  input enable_inputs4x4, reset_transpose_buffer4x4,
        enable_transpose_buffer4x4, change_transpose_buffer_direction4x4
        , reset_psatd4x4, enable_psatd4x4, reset_satd4x4, enable_satd4x4
        , enable_inputs8x8, reset_transpose_buffer8x8,
        enable_transpose_buffer8x8, change_transpose_buffer_direction8x8
        , reset_psatd8x8, enable_psatd8x8, reset_satd8x8, enable_satd8x8
        ;
57
58  //------------------------------- Output Ports
        ---------------------------------
59  output [DATA_WIDTH+7:0] satd4x4A, satd4x4B;
60  output [DATA_WIDTH+11:0] satd8x8;
61  output doneA, doneB, done;
```

```
62
63 //-------------------------------- Internal Wires
       ------------------------------
64 wire signed [DATA_WIDTH+4:0] out_HT_0, out_HT_1, out_HT_2, out_HT_3,
       out_HT_4, out_HT_5, out_HT_6, out_HT_7;
65
66 //-------------------------------- Code Starts Here
       ------------------------------
67 satd_4x4_4pairs_operative operativeA(clk, reset, enable_inputs4x4,
       reset_transpose_buffer4x4, enable_transpose_buffer4x4,
       change_transpose_buffer_direction4x4, reset_psatd4x4,
       enable_psatd4x4, reset_satd4x4, enable_satd4x4, original_0,
       original_1, original_2, original_3, candidate_0, candidate_1,
       candidate_2, candidate_3, satd4x4A, out_HT_0, out_HT_1, out_HT_2
       , out_HT_3);
68
69 satd_4x4_4pairs_operative operativeB(clk, reset, enable_inputs4x4,
       reset_transpose_buffer4x4, enable_transpose_buffer4x4,
       change_transpose_buffer_direction4x4, reset_psatd4x4,
       enable_psatd4x4, reset_satd4x4, enable_satd4x4, original_4,
       original_5, original_6, original_7, candidate_4, candidate_5,
       candidate_6, candidate_7, satd4x4B, out_HT_4, out_HT_5, out_HT_6
       , out_HT_7);
70
71 satd_8x8_8pairs_operative operative8x8(clk, reset, enable_inputs8x8,
       reset_transpose_buffer8x8, enable_transpose_buffer8x8,
       change_transpose_buffer_direction8x8, reset_psatd8x8,
       enable_psatd8x8, reset_satd8x8, enable_satd8x8, out_HT_0,
       out_HT_1, out_HT_2, out_HT_3, out_HT_4, out_HT_5, out_HT_6,
       out_HT_7, satd8x8);
72
73 endmodule
```

Listing A.27: Código fonte do *buffer* de transposição 4×4

```
1  module transpose_buffer_4x4 (
2      clock,
3      reset,
4      enable,
5      direction,
6      in_0,
7      in_1,
8      in_2,
9      in_3,
10     out_0,
11     out_1,
12     out_2,
13     out_3
14 );
15
16 parameter DATA_WIDTH = 8;
17
18 input clock, reset, enable, direction;
19 input signed [DATA_WIDTH-1:0] in_0, in_1, in_2, in_3;
20 output signed [DATA_WIDTH-1:0] out_0, out_1, out_2, out_3;
21 wire signed [DATA_WIDTH-1:0] out_of_0_0, out_of_0_1, out_of_0_2,
       out_of_0_3, out_of_1_0, out_of_1_1, out_of_1_2, out_of_1_3,
       out_of_2_0, out_of_2_1, out_of_2_2, out_of_2_3, out_of_3_0,
       out_of_3_1, out_of_3_2, out_of_3_3;
22
```

```
23 transpose_buffer_cell #(DATA_WIDTH) tb_cell_0_0(clock, reset, enable
       , direction, in_3, in_0, out_of_0_0);
24 transpose_buffer_cell #(DATA_WIDTH) tb_cell_0_1(clock, reset, enable
       , direction, in_2, out_of_0_0, out_of_0_1);
25 transpose_buffer_cell #(DATA_WIDTH) tb_cell_0_2(clock, reset, enable
       , direction, in_1, out_of_0_1, out_of_0_2);
26 transpose_buffer_cell #(DATA_WIDTH) tb_cell_0_3(clock, reset, enable
       , direction, in_0, out_of_0_2, out_of_0_3);
27 transpose_buffer_cell #(DATA_WIDTH) tb_cell_1_0(clock, reset, enable
       , direction, out_of_0_0, in_1, out_of_1_0);
28 transpose_buffer_cell #(DATA_WIDTH) tb_cell_1_1(clock, reset, enable
       , direction, out_of_0_1, out_of_1_0, out_of_1_1);
29 transpose_buffer_cell #(DATA_WIDTH) tb_cell_1_2(clock, reset, enable
       , direction, out_of_0_2, out_of_1_1, out_of_1_2);
30 transpose_buffer_cell #(DATA_WIDTH) tb_cell_1_3(clock, reset, enable
       , direction, out_of_0_3, out_of_1_2, out_of_1_3);
31 transpose_buffer_cell #(DATA_WIDTH) tb_cell_2_0(clock, reset, enable
       , direction, out_of_1_0, in_2, out_of_2_0);
32 transpose_buffer_cell #(DATA_WIDTH) tb_cell_2_1(clock, reset, enable
       , direction, out_of_1_1, out_of_2_0, out_of_2_1);
33 transpose_buffer_cell #(DATA_WIDTH) tb_cell_2_2(clock, reset, enable
       , direction, out_of_1_2, out_of_2_1, out_of_2_2);
34 transpose_buffer_cell #(DATA_WIDTH) tb_cell_2_3(clock, reset, enable
       , direction, out_of_1_3, out_of_2_2, out_of_2_3);
35 transpose_buffer_cell #(DATA_WIDTH) tb_cell_3_0(clock, reset, enable
       , direction, out_of_2_0, in_3, out_of_3_0);
36 transpose_buffer_cell #(DATA_WIDTH) tb_cell_3_1(clock, reset, enable
       , direction, out_of_2_1, out_of_3_0, out_of_3_1);
37 transpose_buffer_cell #(DATA_WIDTH) tb_cell_3_2(clock, reset, enable
       , direction, out_of_2_2, out_of_3_1, out_of_3_2);
38 transpose_buffer_cell #(DATA_WIDTH) tb_cell_3_3(clock, reset, enable
       , direction, out_of_2_3, out_of_3_2, out_of_3_3);
39
40 assign out_0 = (direction) ? out_of_0_3 : out_of_3_3;
41 assign out_1 = (direction) ? out_of_1_3 : out_of_3_2;
42 assign out_2 = (direction) ? out_of_2_3 : out_of_3_1;
43 assign out_3 = (direction) ? out_of_3_3 : out_of_3_0;
44
45 endmodule
```

Listing A.28: Código fonte do *buffer* de transposição 8×8

```
1  module transpose_buffer_8x8 (
2      clock,
3      reset,
4      enable,
5      direction,
6      in_0,
7      in_1,
8      in_2,
9      in_3,
10     in_4,
11     in_5,
12     in_6,
13     in_7,
14     out_0,
15     out_1,
16     out_2,
17     out_3,
18     out_4,
19     out_5,
```

```
20      out_6,
21      out_7
22  );
23
24  parameter DATA_WIDTH = 8;
25
26  input clock, reset, enable, direction;
27  input signed [DATA_WIDTH-1:0] in_0, in_1, in_2, in_3, in_4, in_5,
        in_6, in_7;
28  output signed [DATA_WIDTH-1:0] out_0, out_1, out_2, out_3, out_4,
        out_5, out_6, out_7;
29  wire signed [DATA_WIDTH-1:0] out_of_0_0, out_of_0_1, out_of_0_2,
        out_of_0_3, out_of_0_4, out_of_0_5, out_of_0_6, out_of_0_7,
        out_of_1_0, out_of_1_1, out_of_1_2, out_of_1_3, out_of_1_4,
        out_of_1_5, out_of_1_6, out_of_1_7, out_of_2_0, out_of_2_1,
        out_of_2_2, out_of_2_3, out_of_2_4, out_of_2_5, out_of_2_6,
        out_of_2_7, out_of_3_0, out_of_3_1, out_of_3_2, out_of_3_3,
        out_of_3_4, out_of_3_5, out_of_3_6, out_of_3_7, out_of_4_0,
        out_of_4_1, out_of_4_2, out_of_4_3, out_of_4_4, out_of_4_5,
        out_of_4_6, out_of_4_7, out_of_5_0, out_of_5_1, out_of_5_2,
        out_of_5_3, out_of_5_4, out_of_5_5, out_of_5_6, out_of_5_7,
        out_of_6_0, out_of_6_1, out_of_6_2, out_of_6_3, out_of_6_4,
        out_of_6_5, out_of_6_6, out_of_6_7, out_of_7_0, out_of_7_1,
        out_of_7_2, out_of_7_3, out_of_7_4, out_of_7_5, out_of_7_6,
        out_of_7_7;
30
31  transpose_buffer_cell #(DATA_WIDTH) tb_cell_0_0(clock, reset, enable
        , direction, in_7, in_0, out_of_0_0);
32  transpose_buffer_cell #(DATA_WIDTH) tb_cell_0_1(clock, reset, enable
        , direction, in_6, out_of_0_0, out_of_0_1);
33  transpose_buffer_cell #(DATA_WIDTH) tb_cell_0_2(clock, reset, enable
        , direction, in_5, out_of_0_1, out_of_0_2);
34  transpose_buffer_cell #(DATA_WIDTH) tb_cell_0_3(clock, reset, enable
        , direction, in_4, out_of_0_2, out_of_0_3);
35  transpose_buffer_cell #(DATA_WIDTH) tb_cell_0_4(clock, reset, enable
        , direction, in_3, out_of_0_3, out_of_0_4);
36  transpose_buffer_cell #(DATA_WIDTH) tb_cell_0_5(clock, reset, enable
        , direction, in_2, out_of_0_4, out_of_0_5);
37  transpose_buffer_cell #(DATA_WIDTH) tb_cell_0_6(clock, reset, enable
        , direction, in_1, out_of_0_5, out_of_0_6);
38  transpose_buffer_cell #(DATA_WIDTH) tb_cell_0_7(clock, reset, enable
        , direction, in_0, out_of_0_6, out_of_0_7);
39  transpose_buffer_cell #(DATA_WIDTH) tb_cell_1_0(clock, reset, enable
        , direction, out_of_0_0, in_1, out_of_1_0);
40  transpose_buffer_cell #(DATA_WIDTH) tb_cell_1_1(clock, reset, enable
        , direction, out_of_0_1, out_of_1_0, out_of_1_1);
41  transpose_buffer_cell #(DATA_WIDTH) tb_cell_1_2(clock, reset, enable
        , direction, out_of_0_2, out_of_1_1, out_of_1_2);
42  transpose_buffer_cell #(DATA_WIDTH) tb_cell_1_3(clock, reset, enable
        , direction, out_of_0_3, out_of_1_2, out_of_1_3);
43  transpose_buffer_cell #(DATA_WIDTH) tb_cell_1_4(clock, reset, enable
        , direction, out_of_0_4, out_of_1_3, out_of_1_4);
44  transpose_buffer_cell #(DATA_WIDTH) tb_cell_1_5(clock, reset, enable
        , direction, out_of_0_5, out_of_1_4, out_of_1_5);
45  transpose_buffer_cell #(DATA_WIDTH) tb_cell_1_6(clock, reset, enable
        , direction, out_of_0_6, out_of_1_5, out_of_1_6);
46  transpose_buffer_cell #(DATA_WIDTH) tb_cell_1_7(clock, reset, enable
        , direction, out_of_0_7, out_of_1_6, out_of_1_7);
47  transpose_buffer_cell #(DATA_WIDTH) tb_cell_2_0(clock, reset, enable
        , direction, out_of_1_0, in_2, out_of_2_0);
48  transpose_buffer_cell #(DATA_WIDTH) tb_cell_2_1(clock, reset, enable
```

```
      , direction, out_of_1_1, out_of_2_0, out_of_2_1);
49 | transpose_buffer_cell #(DATA_WIDTH) tb_cell_2_2(clock, reset, enable
      , direction, out_of_1_2, out_of_2_1, out_of_2_2);
50 | transpose_buffer_cell #(DATA_WIDTH) tb_cell_2_3(clock, reset, enable
      , direction, out_of_1_3, out_of_2_2, out_of_2_3);
51 | transpose_buffer_cell #(DATA_WIDTH) tb_cell_2_4(clock, reset, enable
      , direction, out_of_1_4, out_of_2_3, out_of_2_4);
52 | transpose_buffer_cell #(DATA_WIDTH) tb_cell_2_5(clock, reset, enable
      , direction, out_of_1_5, out_of_2_4, out_of_2_5);
53 | transpose_buffer_cell #(DATA_WIDTH) tb_cell_2_6(clock, reset, enable
      , direction, out_of_1_6, out_of_2_5, out_of_2_6);
54 | transpose_buffer_cell #(DATA_WIDTH) tb_cell_2_7(clock, reset, enable
      , direction, out_of_1_7, out_of_2_6, out_of_2_7);
55 | transpose_buffer_cell #(DATA_WIDTH) tb_cell_3_0(clock, reset, enable
      , direction, out_of_2_0, in_3, out_of_3_0);
56 | transpose_buffer_cell #(DATA_WIDTH) tb_cell_3_1(clock, reset, enable
      , direction, out_of_2_1, out_of_3_0, out_of_3_1);
57 | transpose_buffer_cell #(DATA_WIDTH) tb_cell_3_2(clock, reset, enable
      , direction, out_of_2_2, out_of_3_1, out_of_3_2);
58 | transpose_buffer_cell #(DATA_WIDTH) tb_cell_3_3(clock, reset, enable
      , direction, out_of_2_3, out_of_3_2, out_of_3_3);
59 | transpose_buffer_cell #(DATA_WIDTH) tb_cell_3_4(clock, reset, enable
      , direction, out_of_2_4, out_of_3_3, out_of_3_4);
60 | transpose_buffer_cell #(DATA_WIDTH) tb_cell_3_5(clock, reset, enable
      , direction, out_of_2_5, out_of_3_4, out_of_3_5);
61 | transpose_buffer_cell #(DATA_WIDTH) tb_cell_3_6(clock, reset, enable
      , direction, out_of_2_6, out_of_3_5, out_of_3_6);
62 | transpose_buffer_cell #(DATA_WIDTH) tb_cell_3_7(clock, reset, enable
      , direction, out_of_2_7, out_of_3_6, out_of_3_7);
63 | transpose_buffer_cell #(DATA_WIDTH) tb_cell_4_0(clock, reset, enable
      , direction, out_of_3_0, in_4, out_of_4_0);
64 | transpose_buffer_cell #(DATA_WIDTH) tb_cell_4_1(clock, reset, enable
      , direction, out_of_3_1, out_of_4_0, out_of_4_1);
65 | transpose_buffer_cell #(DATA_WIDTH) tb_cell_4_2(clock, reset, enable
      , direction, out_of_3_2, out_of_4_1, out_of_4_2);
66 | transpose_buffer_cell #(DATA_WIDTH) tb_cell_4_3(clock, reset, enable
      , direction, out_of_3_3, out_of_4_2, out_of_4_3);
67 | transpose_buffer_cell #(DATA_WIDTH) tb_cell_4_4(clock, reset, enable
      , direction, out_of_3_4, out_of_4_3, out_of_4_4);
68 | transpose_buffer_cell #(DATA_WIDTH) tb_cell_4_5(clock, reset, enable
      , direction, out_of_3_5, out_of_4_4, out_of_4_5);
69 | transpose_buffer_cell #(DATA_WIDTH) tb_cell_4_6(clock, reset, enable
      , direction, out_of_3_6, out_of_4_5, out_of_4_6);
70 | transpose_buffer_cell #(DATA_WIDTH) tb_cell_4_7(clock, reset, enable
      , direction, out_of_3_7, out_of_4_6, out_of_4_7);
71 | transpose_buffer_cell #(DATA_WIDTH) tb_cell_5_0(clock, reset, enable
      , direction, out_of_4_0, in_5, out_of_5_0);
72 | transpose_buffer_cell #(DATA_WIDTH) tb_cell_5_1(clock, reset, enable
      , direction, out_of_4_1, out_of_5_0, out_of_5_1);
73 | transpose_buffer_cell #(DATA_WIDTH) tb_cell_5_2(clock, reset, enable
      , direction, out_of_4_2, out_of_5_1, out_of_5_2);
74 | transpose_buffer_cell #(DATA_WIDTH) tb_cell_5_3(clock, reset, enable
      , direction, out_of_4_3, out_of_5_2, out_of_5_3);
75 | transpose_buffer_cell #(DATA_WIDTH) tb_cell_5_4(clock, reset, enable
      , direction, out_of_4_4, out_of_5_3, out_of_5_4);
76 | transpose_buffer_cell #(DATA_WIDTH) tb_cell_5_5(clock, reset, enable
      , direction, out_of_4_5, out_of_5_4, out_of_5_5);
77 | transpose_buffer_cell #(DATA_WIDTH) tb_cell_5_6(clock, reset, enable
      , direction, out_of_4_6, out_of_5_5, out_of_5_6);
78 | transpose_buffer_cell #(DATA_WIDTH) tb_cell_5_7(clock, reset, enable
      , direction, out_of_4_7, out_of_5_6, out_of_5_7);
```

```
79  transpose_buffer_cell #(DATA_WIDTH) tb_cell_6_0(clock, reset, enable
        , direction, out_of_5_0, in_6, out_of_6_0);
80  transpose_buffer_cell #(DATA_WIDTH) tb_cell_6_1(clock, reset, enable
        , direction, out_of_5_1, out_of_6_0, out_of_6_1);
81  transpose_buffer_cell #(DATA_WIDTH) tb_cell_6_2(clock, reset, enable
        , direction, out_of_5_2, out_of_6_1, out_of_6_2);
82  transpose_buffer_cell #(DATA_WIDTH) tb_cell_6_3(clock, reset, enable
        , direction, out_of_5_3, out_of_6_2, out_of_6_3);
83  transpose_buffer_cell #(DATA_WIDTH) tb_cell_6_4(clock, reset, enable
        , direction, out_of_5_4, out_of_6_3, out_of_6_4);
84  transpose_buffer_cell #(DATA_WIDTH) tb_cell_6_5(clock, reset, enable
        , direction, out_of_5_5, out_of_6_4, out_of_6_5);
85  transpose_buffer_cell #(DATA_WIDTH) tb_cell_6_6(clock, reset, enable
        , direction, out_of_5_6, out_of_6_5, out_of_6_6);
86  transpose_buffer_cell #(DATA_WIDTH) tb_cell_6_7(clock, reset, enable
        , direction, out_of_5_7, out_of_6_6, out_of_6_7);
87  transpose_buffer_cell #(DATA_WIDTH) tb_cell_7_0(clock, reset, enable
        , direction, out_of_6_0, in_7, out_of_7_0);
88  transpose_buffer_cell #(DATA_WIDTH) tb_cell_7_1(clock, reset, enable
        , direction, out_of_6_1, out_of_7_0, out_of_7_1);
89  transpose_buffer_cell #(DATA_WIDTH) tb_cell_7_2(clock, reset, enable
        , direction, out_of_6_2, out_of_7_1, out_of_7_2);
90  transpose_buffer_cell #(DATA_WIDTH) tb_cell_7_3(clock, reset, enable
        , direction, out_of_6_3, out_of_7_2, out_of_7_3);
91  transpose_buffer_cell #(DATA_WIDTH) tb_cell_7_4(clock, reset, enable
        , direction, out_of_6_4, out_of_7_3, out_of_7_4);
92  transpose_buffer_cell #(DATA_WIDTH) tb_cell_7_5(clock, reset, enable
        , direction, out_of_6_5, out_of_7_4, out_of_7_5);
93  transpose_buffer_cell #(DATA_WIDTH) tb_cell_7_6(clock, reset, enable
        , direction, out_of_6_6, out_of_7_5, out_of_7_6);
94  transpose_buffer_cell #(DATA_WIDTH) tb_cell_7_7(clock, reset, enable
        , direction, out_of_6_7, out_of_7_6, out_of_7_7);
95
96  assign out_0 = (direction) ? out_of_0_7 : out_of_7_7;
97  assign out_1 = (direction) ? out_of_1_7 : out_of_7_6;
98  assign out_2 = (direction) ? out_of_2_7 : out_of_7_5;
99  assign out_3 = (direction) ? out_of_3_7 : out_of_7_4;
100 assign out_4 = (direction) ? out_of_4_7 : out_of_7_3;
101 assign out_5 = (direction) ? out_of_5_7 : out_of_7_2;
102 assign out_6 = (direction) ? out_of_6_7 : out_of_7_1;
103 assign out_7 = (direction) ? out_of_7_7 : out_of_7_0;
104
105 endmodule
```

Listing A.29: Código fonte do bloco operativo da SATD 4×4

```
1  //
       ----------------------------------------------------------------------
2  // Design Name : satd_4x4_4pairs_operative
3  // File Name   : satd_4x4_4pairs_operative.v
4  // Function    : SATD of 4x4 blocks with 4 pixel pairs as input
5  // Coder       : Ismael Seidel and Marcio Monteiro
6  //
       ----------------------------------------------------------------------
7  module satd_4x4_4pairs_operative(
8      clock,
9      reset,
10     enable_inputs,
11     reset_transpose_buffer,
```

```
12      enable_transpose_buffer,
13      change_transpose_buffer_direction,
14      reset_psatd,
15      enable_psatd,
16      reset_satd,
17      enable_satd,
18      original_0,
19      original_1,
20      original_2,
21      original_3,
22      candidate_0,
23      candidate_1,
24      candidate_2,
25      candidate_3,
26      satd,
27      out_HT_4x4_0,
28      out_HT_4x4_1,
29      out_HT_4x4_2,
30      out_HT_4x4_3
31  );
32
33  parameter DATA_WIDTH = 8;
34
35  //------------------------------- Input Ports
        ---------------------------------
36  input clock, reset, enable_inputs, reset_transpose_buffer,
        enable_transpose_buffer, change_transpose_buffer_direction,
        reset_psatd, enable_psatd, reset_satd, enable_satd;
37  input [DATA_WIDTH-1:0] original_0, original_1, original_2,
        original_3, candidate_0, candidate_1, candidate_2, candidate_3;
38
39  //------------------------------- Output Ports
        ---------------------------------
40  output reg [DATA_WIDTH+7:0] satd;
41  output signed [DATA_WIDTH+4:0] out_HT_4x4_0, out_HT_4x4_1,
        out_HT_4x4_2, out_HT_4x4_3;
42
43  //------------------------------- Internal Wires
        ----------------------------
44  wire transpose_buffer_direction;
45  wire [DATA_WIDTH-1:0] original_reg_0, original_reg_1, original_reg_2
        , original_reg_3, candidate_reg_0, candidate_reg_1,
        candidate_reg_2, candidate_reg_3;
46  wire signed [DATA_WIDTH:0] difference_0, difference_1, difference_2,
         difference_3;
47  wire signed [DATA_WIDTH+2:0] out_of_1st_1d_transform_0,
        out_of_1st_1d_transform_1, out_of_1st_1d_transform_2,
        out_of_1st_1d_transform_3;
48  wire signed [DATA_WIDTH+2:0] out_of_transpose_buffer_0,
        out_of_transpose_buffer_1, out_of_transpose_buffer_2,
        out_of_transpose_buffer_3;
49  wire signed [DATA_WIDTH+4:0] out_of_2nd_1d_transform_0,
        out_of_2nd_1d_transform_1, out_of_2nd_1d_transform_2,
        out_of_2nd_1d_transform_3;
50  wire [DATA_WIDTH+3:0] absolute_transformed_difference_0,
        absolute_transformed_difference_1,
        absolute_transformed_difference_2,
        absolute_transformed_difference_3;
51  wire [DATA_WIDTH+5:0] sum;
52  wire [DATA_WIDTH+7:0] psatd;
53
```

```
54 //-------------------------------- Code Starts Here
       ----------------------------
55 tff_async_reset transpose_buffer_direction_holder(
       change_transpose_buffer_direction, clock, reset,
       transpose_buffer_direction);
56
57 input_buffer_4pairs #(DATA_WIDTH) register_inputs(clock, reset,
       enable_inputs, original_0, original_1, original_2, original_3,
       candidate_0, candidate_1, candidate_2, candidate_3,
       original_reg_0, original_reg_1, original_reg_2, original_reg_3,
       candidate_reg_0, candidate_reg_1, candidate_reg_2,
       candidate_reg_3);
58
59 difference_layer_4pairs #(DATA_WIDTH) difference(original_reg_0,
       original_reg_1, original_reg_2, original_reg_3, candidate_reg_0,
        candidate_reg_1, candidate_reg_2, candidate_reg_3, difference_0
       , difference_1, difference_2, difference_3);
60
61 transform_1d_4inputs #(DATA_WIDTH+1) first_1d_transform(difference_0
       , difference_1, difference_2, difference_3,
       out_of_1st_1d_transform_0, out_of_1st_1d_transform_1,
       out_of_1st_1d_transform_2, out_of_1st_1d_transform_3);
62
63 transpose_buffer_4x4 #(DATA_WIDTH+3) transpose_buffer(clock,
       reset_transpose_buffer, enable_transpose_buffer,
       transpose_buffer_direction, out_of_1st_1d_transform_0,
       out_of_1st_1d_transform_1, out_of_1st_1d_transform_2,
       out_of_1st_1d_transform_3, out_of_transpose_buffer_0,
       out_of_transpose_buffer_1, out_of_transpose_buffer_2,
       out_of_transpose_buffer_3);
64
65 transform_1d_4inputs_second #(DATA_WIDTH+3) second_1d_transform(
       out_of_transpose_buffer_0, out_of_transpose_buffer_1,
       out_of_transpose_buffer_2, out_of_transpose_buffer_3,
       out_of_2nd_1d_transform_0, out_of_2nd_1d_transform_1,
       out_of_2nd_1d_transform_2, out_of_2nd_1d_transform_3); // pegar
       daqui os valores e jogar pro criador de tabela 8x8
66
67 abs_layer_4inputs #(DATA_WIDTH+5) absolute1(
       out_of_2nd_1d_transform_0, out_of_2nd_1d_transform_1,
       out_of_2nd_1d_transform_2, out_of_2nd_1d_transform_3,
       absolute_transformed_difference_0,
       absolute_transformed_difference_1,
       absolute_transformed_difference_2,
       absolute_transformed_difference_3);
68
69 sum_tree_4inputs #(DATA_WIDTH+4) sum_tree(
       absolute_transformed_difference_0,
       absolute_transformed_difference_1,
       absolute_transformed_difference_2,
       absolute_transformed_difference_3, sum);
70
71 accumulator #(DATA_WIDTH+6, DATA_WIDTH+8) acc(clock, reset_psatd,
       enable_psatd, sum, psatd);
72
73 // always @(*) begin
74     assign out_HT_4x4_0 = out_of_2nd_1d_transform_0;
75     assign out_HT_4x4_1 = out_of_2nd_1d_transform_1;
76     assign out_HT_4x4_2 = out_of_2nd_1d_transform_2;
77     assign out_HT_4x4_3 = out_of_2nd_1d_transform_3;
78 // end
```

```
79
80 always @(posedge clock) begin
81     if (reset_satd) begin
82         satd <= 0;
83     end
84     else if (enable_satd) begin
85         satd <= (psatd+sum)>>1;
86     end
87 end
88
89 endmodule
```

Listing A.30: Código fonte da célula do *buffer* de transposição

```
1  module transpose_buffer_cell (
2      clock,
3      reset,
4      enable,
5      direction,
6      in_0,
7      in_1,
8      out
9  );
10
11 parameter DATA_WIDTH = 8 ;
12
13 input clock, reset, enable, direction;
14 input signed[DATA_WIDTH-1:0] in_0, in_1;
15 output signed[DATA_WIDTH-1:0] out;
16
17 reg signed[DATA_WIDTH-1:0] out;
18
19 always @ (posedge clock or posedge reset)
20 if (reset) begin
21  out <= 0;
22 end else if (enable && !direction) begin
23   out <= in_0;
24 end else if (enable && direction) begin
25   out <= in_1;
26 end
27
28 endmodule
```

APÊNDICE B – Artigo

Arquitetura energeticamente eficiente para cálculo da SATD através do reúso de dados

Marcio Monteiro, Ismael Seidel, José Luís Güntzel

[1]Departamento de Informática e Estatística –
Universidade Federal de Santa Catarina (UFSC)
Florianópolis – SC – Brazil

{marcio, ismael.seidel}@inf.ufsc.br, j.guntzel@ufsc.br

Resumo. *O contínuo aumento das resoluções usadas em vídeos digitais tornam necessária a adoção de novas técnicas de codificação de vídeo. A Estimação de Movimento (ME) é a etapa mais intensiva em termos de tempo e consumo energético por realizar um elevado número de cálculos de similaridade entre blocos, como por exemplo a SATD. Assim, este trabalho propõe uma arquitetura de SATD com reúso de cálculos, tendo como objetivo diminuir o consumo energético. Após a descrição da arquitetura, a mesma foi sintetizada e simulada com uma ferramenta de uso industrial. Foram utilizados cinco conjuntos de dados para simulação, um gerado a partir de dados aleatórios e quatro a partir de sequências de vídeos. Ao analisar os resultados obtidos, houve uma redução na área de até 80% em relação às arquiteturas do estado da arte. O consumo energético da arquitetura projetada foi até 55% menor do que aqueles apresentados pelas arquiteturas do estado da arte. Portanto, a arquitetura proposta se mostra vantajosa quando é necessário calcular múltiplos tamanhos de blocos.*

1. Introdução

Um vídeo é uma sequência de imagens, chamadas de quadros, apresentadas rapidamente no tempo. Cada quadro é uma matriz de píxeis, cujo tamanho é chamado de resolução. Em codificadores baseados em blocos os quadros são divididos matrizes menores, referenciadas por blocos. O particionamento do quadro contribui para o processamento pois facilita a busca e redução das redundâncias. Nos vídeos digitais, a redundância espacial é a repetição de informação em um mesmo quadro, enquanto que a redundância temporal é a repetição de informação entre quadros sucessivos [Ghanbari 2003].

Um codificador de vídeo pode ser dividido em diversas etapas [Sullivan et al. 2012]. Para auxiliar na redução das redundâncias, os quadros original e candidato são particionados em blocos. A etapa de predição busca entre vários blocos candidatos ($\mathbf{B}^{\text{can}}$), o mais semelhante ao bloco original ($\mathbf{B}^{\text{ori}}$) e o candidato escolhido é chamado de bloco de referência ($\mathbf{B}^{\text{ref}}$).

Um dos passos da predição é a Estimação de Movimento (ME). Tal passo tem como objetivo reduzir as redundâncias temporais. É também um dos passos mais intensivos do ponto de vista computacional [Bossen et al. 2012], sendo responsável por entre 58% e 86% [Li et al. 2016] do tempo total de codificação no padrão de Codificação de Vídeo de Alta Eficiência (HEVC). Com a adoção de resoluções cada vez maiores, como *Full HD* (1920×1080 píxeis) e *Quad HD* (3840×2160 píxeis), a ME requer cada vez mais tempo para realizar os cálculos.

Do tempo total para execução da ME a maior parte é ocupado pelo cálculo da similaridade entre blocos [Silveira et al. 2015, Soares et al. 2016]. Duas das principais métricas de similaridade utilizadas na ME são a Soma das Diferenças Absolutas (SAD) e a Soma das Diferenças Transformadas Absolutas (SATD) [Richardson 2003]. Tais métricas de similaridade são compostas por operações aritméticas com baixa complexidade (somas, subtrações e absolutos).

Algumas arquiteturas de Integração em Larga Escala (VLSI) para o cálculo da SAD exploram o reúso de cálculos para diminuir o tempo de codificação e melhorar a eficiência energética da métrica. Este reúso é feito através da soma das SADs dos blocos menores para obter a SAD do bloco maior [Kim and Park 2009]. Para a SATD, existem implementações que utilizam o mesmo bloco de *hardware* para os cálculos da SATD do bloco menor e do bloco maior [Silveira 2016]. Porém, não foram encontrados trabalhos que fizessem a aplicação do reúso de cálculos na SATD. Assim, neste trabalho é proposto um método para reúso de cálculos e uma arquitetura para cálculo das SATDs de blocos 4×4 e 8×8 com reúso de cálculos. Os tamanhos de blocos suportados foram restringidos aos supracitados para manter a compatibilidade com o Modelo de Testes do HEVC (HM). O reúso de cálculos tem por objetivo diminuir a energia necessária para codificar um vídeo.

O restante deste trabalho está organizado da seguinte forma. Na Seção 2 são apresentados os conceitos básicos de codificação de vídeo utilizados neste trabalho. Na Seção 3 é apresentada a arquitetura proposta neste trabalho. Na Seção 4 são relatados os resultados obtidos após a síntese e simulação. Finalmente, na Seção 5 são apresentadas as conclusões.

2. Conceitos Básicos

A Estimação de Movimento (ME) é a etapa responsável por escolher um $\mathbf{B}^{\text{ref}}$, dentre um conjunto de candidatos, que tenha a maior similaridade com o bloco que está sendo codificado, chamado de $\mathbf{B}^{\text{ori}}$. A ME faz a comparação de uma área do quadro candidato em que estão localizados os blocos candidatos, chamada de Área de Busca - *Search Window* (SW), com o bloco original utilizando uma métrica de similaridade. Em geral, as métricas de similaridade são computadas a partir da diferença entre o bloco original ($\mathbf{B}^{\text{ori}}$) e um candidato ($\mathbf{B}^{\text{can}}$), conforme a Equação 1. M e N são as dimensões do bloco na forma 2^k para k$\in \mathbb{N}^*$.

$$\mathbf{D}_{M \times N} = \mathbf{B}^{\text{ori}}_{M \times N} - \mathbf{B}^{\text{can}}_{M \times N} \tag{1}$$

A Soma das Diferenças Absolutas (SAD) é utilizada por sua simplicidade. Na Equação 2 é definido o cálculo da SAD, onde $d_{i,j}$ é o elemento na posição i, j de $\mathbf{D}$ (Equação 1).

$$\text{SAD}_{M \times N} = \sum_{i=1}^{M} \sum_{j=1}^{N} \mid d_{i,j} \mid \tag{2}$$

A Soma das Diferenças Transformadas Absolutas (SATD) utiliza o mesmo princípio de cálculo da SAD, mas faz uma transformação da matriz de diferenças com a Transformada de Hadamard (HT) antes de efetuar a soma absoluta. A SATD obtém melhor qualidade de codificação pois os seus valores calculados tem uma correlação maior com os resíduos transformados.

A SATD é definida como a multiplicação de uma constante de dimensionamento positiva e não nula pelo somatório dos valores absolutos de uma matriz de Diferenças Transformadas ($\mathbf{TD}$), como pode ser visto na Equação 3.

$$\mathrm{SATD}_{N\times N} = \frac{1}{log_2 N} \times \sum_{i=1}^{N}\sum_{j=1}^{N} \mid td_{i,j} \mid \tag{3}$$

onde N é da forma 2^k, com k$\in \mathbb{N}^*$ e $td_{i,j}$ é o elemento na posição i, j de $\mathbf{TD}$. A Equação 4 apresenta como calcular a $\mathbf{TD}$. Ao utilizar a multiplicação usual de matrizes são necessárias $2N^3$ multiplicações e $2N^2$ somas. Com isto nota-se que a complexidade da transformada está na multiplicação das matrizes.

$$\mathbf{TD}_{N\times N} = \mathbf{T}_{N\times N} \times \mathbf{D}_{N\times N} \times \mathbf{T}^T_{N\times N} \tag{4}$$

Uma vez que a matriz de transformação é quadrada, pode-se perceber que a SATD não pode ser aplicada diretamente em blocos não quadrados. Para aplicá-la, inicialmente o bloco não quadrado é dividido em blocos quadrados menores. A SATD é então calculada para esses blocos menores e os resultados são somados para obter a SATD do bloco não quadrado.

Ao substituir $\mathbf{T}$ pela matriz de Hadamard ($\mathbf{H}$) na Equação 4 é obtida a HT. A $\mathbf{H}$ de dimensões 2^n, com $n \in \mathbb{N}^*$, pode ser construída como segue [Agaian et al. 2011]:

$$\mathbf{H}_{2^n\times 2^n} = \begin{cases} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} & \text{, se } n = 1 \\ \begin{bmatrix} \mathbf{H}_{2^{n-1}\times 2^{n-1}} & \mathbf{H}_{2^{n-1}\times 2^{n-1}} \\ \mathbf{H}_{2^{n-1}\times 2^{n-1}} & -\mathbf{H}_{2^{n-1}\times 2^{n-1}} \end{bmatrix} & \text{, caso contrário} \end{cases} \tag{5}$$

Para diminuir o número de operações, podem ser explorados o fato da $\mathbf{H}$ ser composta apenas por valores 1 e -1 o que permite a multiplicação de matrizes usando apenas somas e subtrações. A outra é a recursividade, que permite o uso de estruturas em *butterfly* para o cálculo da transformada rápida, nesse caso, a Transformada Rápida de Hadamard (FHT). Também é possível separar as duas multiplicações de matrizes, a qual é chamada de separabilidade [Porto et al. 2005], contudo não é uma propriedade da $\mathbf{H}$.

Ao separar as duas multiplicações de matrizes da Equação 1 é possível utilizar um vetor construído a partir de uma coluna de $\mathbf{D}$ para calcular a $\mathbf{TD}$. [Anshi et al. 1993] apresentam uma formulação com um vetor genérico (Equação 6) que multiplica uma $\mathbf{H}_{4\times 4}$ (Equação 5 com $n = 2$) e o resultado obtido é apresentado na Equação 7. Nota-se que a única diferença entre as equações I e III e as equações II e IV é um sinal. Com isto, é possível subdividir as equações em operações de duas parcelas em que inicialmente são calculadas as somas/subtrações de f_0 com f_1 e f_2 com f_3, para em seguida somar/subtrair os resultados obtidos anteriormente. Pode-se notar que é possível fazer apenas uma vez os cálculos e reutilizá-los para multiplicar o vetor pela $\mathbf{H}$.

$$\mathbf{X}_{1\times 4} = \begin{matrix} f_0 & f_1 & f_2 & f_3 \end{matrix}^T \tag{6}$$

$$\begin{matrix} (f_0 + f_1) + (f_2 + f_3) & [I] \\ (f_0 - f_1) + (f_2 - f_3) & [II] \\ (f_0 + f_1) - (f_2 + f_3) & [III] \\ (f_0 - f_1) - (f_2 - f_3) & [IV] \end{matrix} \tag{7}$$

Ao utilizar *butterflies* para calcular a $\mathbf{TD}$ são necessárias $2N^2log_2N$ operações. Para matrizes de tamanho 8×8, são necessárias apenas 384 operações aritméticas para calcular a $\mathbf{TD}$. Nota-se que o uso da *butterfly* para fazer a transformação das matrizes pode reduzir o tempo necessário para calcular as mesmas.

3. Proposta

O reúso na SATD baseia-se na mesma ideia empregada para reúso na SAD, que consiste em calcular a métrica dos blocos menores e utilizar estes valores para obter o valor da métrica do bloco maior. Contudo, não é possível reaproveitar o cálculo completo da métrica, apenas as transformadas.

Na Figura 1 é apresentado o bloco operativo da arquitetura proposta, e esta pode ser dividida em três partes principais. Duas estruturas paralelas para cálculo da SATD 4×4 e uma estrutura para cálculo da SATD 8×8. A capacidade de processamento é de quatro blocos 4×4 em paralelo, sendo dois $\mathbf{B}^{\text{ori}}$ e dois $\mathbf{B}^{\text{can}}$. A entrada dos dados é feita a cada ciclo de relógio, onde são inseridos novos valores. As duas estruturas para cálculo da SATD$_4\times4$ são utilizadas para alimentar a parte responsável pelo cálculo dos blocos 8×8. Os passos realizados pelas duas estruturas são descritos a seguir.

Inicialmente, são inseridos dois vetores de tamanho 4×1, sendo um de $\mathbf{B}^{\text{ori}}$ e outro de $\mathbf{B}^{\text{can}}$, no *Registradores*, que é uma barreira de registradores para manter estáveis os valores por um ciclo de relógio. A seguir, a *Diferença* subtrai os valores do vetor de $\mathbf{B}^{\text{ori}}$ dos valores do vetor de $\mathbf{B}^{\text{can}}$ e a saída é utilizada como entrada para a primeira transformada. Nesta é feita a multiplicação da $\mathrm{H}_{4\times4}$ pelo resultado da *Diferença* através de uma *butterfly* e o resultado é armazenado no *Buffer de Transposição* 4×4. A segunda transformada multiplica os valores da saída do *buffer* pela $\mathrm{H}_{4\times4}$. Neste ponto, os valores da saída da segunda transformada são enviados para dois blocos distintos, para o *Absoluto* e para o cálculo da SATD 8×8.

O *Absoluto* obtém os valores absolutos dos resultados da segunda transformada e disponibiliza na sua saída. O *Somatório* utiliza uma árvore de somadores para obter a soma de todas as saídas do *Absoluto* e enviar para o *Acumulador*. No *Acumulador* é somada a saída do *Somatório* com o valor armazenado no registrador interno dele e o resultado é disponibilizado na saída.

Cada entrada do *Ajuste*$_0$ recebe um vetor de dimensões 4×1 das estruturas para cálculo da SATD 4×4. Esta etapa de ajuste é necessário para compor os dois vetores de entrada em um vetor de dimensões 8×1 para ser utilizado no cálculo da SATD 8×8. A construção do vetor 8×1 é feita através da multiplicação dos dois vetores 4×1 pela matriz de ajuste $\mathbf{A}$, construída conforme a Equação 8, e os valores resultantes são armazenadas no *Buffer* de Transposição (TB) 8×8.

$$\mathbf{A}_{2^n\times2^n} = \mathbf{H}_{2\times2} \otimes \mathbf{I}_{2^{n-1}\times2^{n-1}} \tag{8}$$

O bloco $Ajuste_1$ multiplica as saídas do TB pela matriz de ajuste para completar os cálculos da transformada do bloco 8×8. O *Absoluto*, o *Somatório* e o *Acumulador* são estruturas similares as utilizadas para 4×4, com a diferença que são utilizados mais operadores e o número de ciclos é maior. Ao término do cálculo do bloco, é disponibilizado o valor da SATD 8×8 como saída.

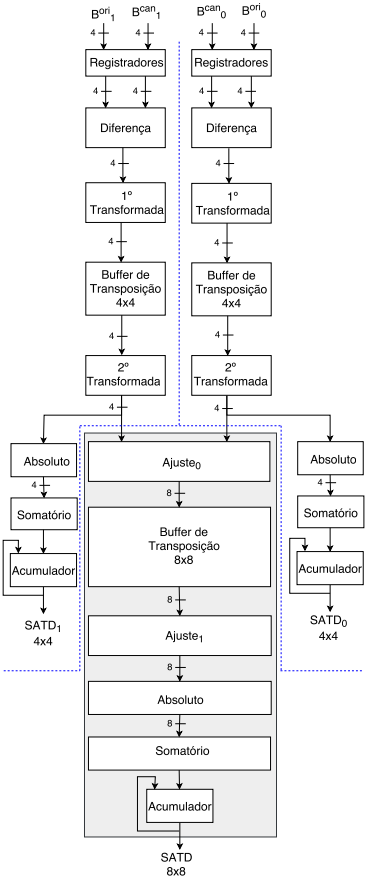


Figura 1. Esquemático da arquitetura proposta para cálculo de SATD com reúso de cálculos. As linhas tracejadas separam as duas estruturas para o cálculo das SATDs 4×4 e no retângulo sombreado está a estrutura para cálculo da 8×8. Os números assinalados na arquitetura representam a quantidade de dados transmitidos entre as partes. Dos oito pares de valores de entrada da arquitetura, $\mathrm{B}_0^{\mathbf{ori}}$ e $\mathrm{B}_0^{\mathbf{can}}$ recebem quatro pares e os demais são recebidos por $\mathrm{B}_1^{\mathbf{ori}}$ e $\mathrm{B}_1^{\mathbf{can}}$.

O comportamento do bloco operativo (Figura 1) foi projetado para se comportar conforme o comportamento da Máquina de Estados Finitos (FSM) apresentada na Figura 2. Cada estado tracejado representa um conjunto de estados que foram agrupados e que são descritos a seguir. Foram adotadas as seguintes convenções, *N* é o índice que indica o número do estado dentro de um agrupamento e as setas tracejadas representam as conexões entre um estado de origem e um de destino. O estado $\mathtt{I}_{N-1}$ representa dezenove estados($\mathtt{I}_1$ até $\mathtt{I}_{19}$) utilizados para popular a arquitetura. O estado $\mathtt{C}_{N-1}$ representa quatro estados ($\mathtt{C}_3$ à $\mathtt{C}_7$) responsáveis pelo controle da arquitetura enquanto esta estiver operando no laço. E finalmente, o estado $\mathtt{T}_{N-1}$ representa quatro estados ($\mathtt{T}_3$ até $\mathtt{T}_7$) que

descarregam o TB 8×8 e calculam as SATDs dos últimos blocos.

As duas primeiras SATDs 4×4 são liberadas no estado $\mathtt{I}_9$ e para cada quatro ciclos de relógio são liberadas mais duas. Já a primeira SATD 8×8 fica pronta somente no estado $\mathtt{PC}_1$ e a cada oito ciclos é liberado um novo valor. Os últimos valores para os blocos 4×4 são liberados no estado $\mathtt{T}_2$, já para o 8×8 é liberado no estado $\mathtt{L}_1$.

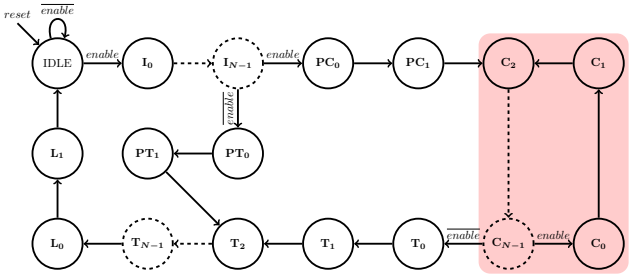


Figura 2. FSM utilizada para controlar a arquitetura proposta. Os estados tracejados representam agrupamentos de estados e as setas tracejadas representam as transições entre os estados do agrupamento. Similar ao trabalho de [Seidel et al. 2016a].

Ao analisar o número de operações necessárias para calcular a **TD** utilizando a FHT para um bloco 4×4, nota-se que estão divididas em 4 operações para *Diferença*, 8 para cada transformada, 4 para o *Absoluto* e 3 para o *Somatório*. Ao total são necessárias 108 operações por linha. Para um bloco 4×4 são necessárias mais 4 operações do *Acumulador*, o que resulta em 112 operações. Para um bloco 8×8 são necessárias 576 operações, sendo 8 para o *Acumulador* e as demais 568 para o cálculo das linhas do bloco.

Ao utilizar o reúso continuam sendo necessárias 112 operações para o cálculo da SATD 4×4. Porém, para a 8×8 são necessárias apenas 256 operações, sendo 8 para o *Acumulador* e as demais 248 operações para calcular as 8 partes da SATD 8×8. Para cada parte são feitas 16 operações para as etapas de *Ajuste* (8 para cada um), 8 para o *Absoluto* e 7 para o *Somatório*. Desta forma, ao utilizar o reúso dos cálculos das SATDs 4×4, a SATD 8×8 precisa de apenas 256 operações, economizando 320 operações em relação ao método sem reúso.

4. Resultados e discussão

Inicialmente foi desenvolvido um *software* baseado no modelo implementado por [Bräscher 2016] para validação da arquitetura proposta. Para isto, foram utilizadas as linguagens C++ para desenvolver o *software* e *Verilog* para descrever a arquitetura. Para fazer a descrição em *Verilog* e o *testbench* da arquitetura foi utilizado o gerador desenvolvido por [Seidel et al. 2016a].

Após finalizar o desenvolvimento do modelo do *software* de validação e a descrição da arquitetura, ambas foram testadas para verificar se o funcionamento estava dentro do esperado. Ao término desta avaliação, ambas foram sintetizadas e simuladas usando a biblioteca de células padrão de 45 nm da TSMC e período de relógio de 15,625 ns, a mesma configuração utilizada por [Bräscher 2016]. Para a síntese lógica foi uti-

lizada a ferramenta *Synopsys® Design Compiler®* (DC®) [Synopsys 2016a] em modo topográfico. A síntese lógica foi executada para se obter estimativas de área e potência da arquitetura.

Para todas as sínteses, os atrasos de entrada e saída foram limitados a 60% do período de relógio. A capacitância máxima das entradas foram configuradas para $10\times$ o valor da entrada de uma porta AND de 2 entradas. Já as saídas foram configuradas para $30\times$ a saída de uma porta lógica AND de 2 entradas. Também, foi utilizado o valor de 32 milhões de blocos 4×4/s como *throughput*, o dobro de [Walter and Bampi 2011].

Para gerar os dados realistas, o HM foi modificado para salvar os blocos candidatos e referências de tamanho 8×8 em arquivos. O arquivo de configuração utilizado foi o `encoder_lowdelay_P_main.cfg` com Parâmetro de Quantização - *Quantization Parameter* (QP) 32. Para cada vídeo foram sorteadas aleatoriamente 1% das janelas de buscas dos primeiros cinco quadros.

Foram feitas 4 codificações, usando o HM modificado, com as sequências de vídeos *Traffic*, *BQTerrace*, *RaceHorses* e *BlowingBubbles*. Estes vídeos são os mesmos utilizados por [Soares et al. 2016] e fazem parte das Condições Comuns de Teste (CTC)[Bossen 2012]. A partir das *netlists* obtidas na síntese lógica foram realizadas simulações com o *Synopsys® Verilog Compiler Simulator* (VCS®) [Synopsys 2016b] para validar as arquiteturas sintetizadas.

A área obtida após fazer a síntese da arquitetura proposta foi de 14981 μm^2. A Tabela 1 apresenta os resultados estimados de potência. A simulação utilizando dados aleatórios obteve os maiores resultados para as potências. A variação entre as potências estáticas ocorre pois o cálculo da potência estática dependente da atividade de chaveamento, como pode ser visto na biblioteca utilizada [tsm 2011].

Tabela 1. Resultados de potência por vídeo após a síntese e simulação da arquitetura. A primeira linha apresenta os resultados da síntese, porém sem simulação, o que assume uma atividade de chaveamento de 10% [Synopsys 2016c]. Na segunda linha é reportado os resultados da simulação com os dados aleatórios, nas quatro linhas seguintes estão os valores obtidos com as simulações com dados realistas. Nas últimas duas linhas são apresentadas a média (μ) e desvio padrão (σ) para potência e energia considerando apenas os resultados da síntese após a simulação com os dados realistas.

Dados	Potência (μW)		
	Estática	Dinâmica	Total
Sem Simulação	143,8	540,3	684,2
Dados aleatórios	157,5	1176,1	1333,6
Traffic	155,8	1027,7	1183,5
BQTerrace	154,3	1023,5	1177,8
RaceHorses	156,0	1163,4	1319,4
BlowingBubbles	156,1	1130,4	1286,5
Média (μ)	**155,6**	**1086,3**	**1241,8**
Desvio Padrão (σ)	**0,9**	**71,3**	**71,9**

Ao comparar o valor de potência total obtido para a síntese com a média dos resultados das simulações com dados realistas, nota-se que a diferença é de 44,9%. Isso dá indícios de que apenas utilizar os valores obtidos pela síntese sem simulação para obter a energia e potência não é o mais indicado. Desta forma, simular a arquitetura com dados

realistas e sintetisar considerando a atividade de chaveamento se mostra necessária para obter valores mais precisos para potência.

Para o cálculo da energia é utilizada a Equação 9, em que $\text{Energia}_{\text{SATD}}$ é a energia para cada $\text{SATD}_{8\times8}$ calculada, Potência_{tot} é a potência total, #Ciclos é o número de ciclos para calcular cada SATD e Período é o tempo de relógio entre duas subidas de *clock*.

$$\text{Energia}_{\text{SATD}} = \text{Potência}_{tot} \times \#\text{Ciclos} \times \text{Período} \tag{9}$$

Para a arquitetura proposta não é calculada a energia para as SATDs dos blocos 4×4, pois este custo já está incluso no cálculo da energia para blocos 8×8.

A Figura 3 apresenta o gráfico para energia. Nota-se que existe uma variação para as simulações com dados realistas, indicando que o conteúdo dos vídeos influencia no valor de energia.

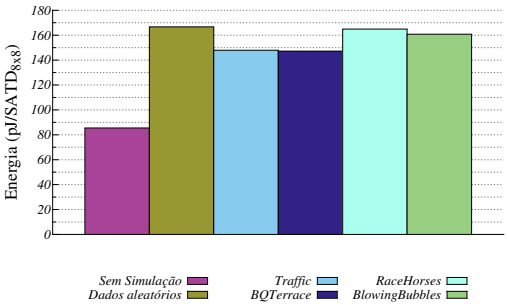


Figura 3. Resultados de energia para a síntese e as simulações.

Na Figura 4 são apresentadas as comparações de área dos trabalhos correlatos, em que o valor de base utilizado foi o obtido pela arquitetura proposta neste trabalho.

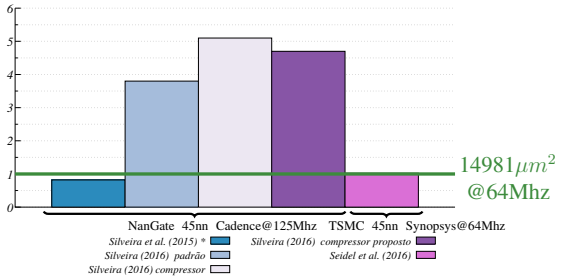


Figura 4. Comparação de área, utilizando como base o valor da arquitetura proposta neste trabalho. A arquitetura de [Silveira et al. 2015] calcula a SATD apenas para blocos 8×8.

A arquitetura proposta por [Silveira et al. 2015] ocupa uma área de 18,35% menor que a proposta neste trabalho. Porém, a arquitetura de [Silveira et al. 2015] calcula so-

mente blocos 8×8 e foi sintetizada com a biblioteca [nan 2015] de 45 nm na ferramenta Cadence RTL Compiler tool [cad 2015].

[Silveira 2016] propôs três arquiteturas para SATD, sendo uma com somadores padrão da ferramenta, uma com somadores e subtratores compressores e a última com somadores/compressores propostos no próprio trabalho. A arquitetura proposta neste trabalho ocupou 74%, 80% e 79% menos área que as arquiteturas de [Silveira 2016]. As configurações utilizadas na síntese das três arquiteturas propostas por [Silveira 2016] são as mesmas de [Silveira et al. 2015].

Ao comparar com as arquiteturas para 8×8 e 4×4 propostas por [Seidel et al. 2016a], a arquitetura deste trabalho apresentou uma redução de 2,44% na área. Já, [Seidel et al. 2016a] fizeram a síntese das arquiteturas com a ferramenta DC® e a biblioteca de células padrão de 45 nm da *Taiwan Semiconductor Manufacturing Company Limited* (TSMC) para dois cenários: Nominal (NN) e *Low-Vdd/High-Vt* (LH). Para uma comparação justa, foram utilziados os resultados NN, uma vez que foram obtidas com as mesmas configurações do presente trabalho.

No trabalho de [Seidel et al. 2016a] foram reportados 16,4 pJ/SATD para 4×4 e 60,5 pJ/SATD para 8×8, totalizando 125,9 pJ/SATD. A arquitetura proposta no presente trabalho é 32% mais energeticamente eficiente que as arquiteturas de [Seidel et al. 2016a]. Já em [Seidel et al. 2016b], a arquitetura para 4×4 obteve 8,1 pJ/SATD e para 8×8 161 pJ/SATD totalizando 193,8 pJ/SATD para calcular quatro $\text{SATDs}_{4\times4}$ e uma $\text{SATD}_{8\times8}$. Ou seja, a arquitetura do presente trabalho é 55% mais energeticamente eficiente que as de [Seidel et al. 2016b]. Ambas as comparações são relativas ao valor de energia obtido sem simulação, pois tais trabalhos não fazem simulação.

5. Conclusões

Ao examinar os resultados de potência e energia obtidos na síntese sem simulação, observa-se que estes valores são aproximadamente a metade dos obtidos após as simulações. A única exceção é a potência estática que os valores foram próximos. Desta forma pode-se notar que os resultados da síntese sem a simulação são estimativas otimistas, no contexto da codificação de vídeo.

Quando comparada com as arquiteturas do estado da arte, a arquitetura proposta neste trabalho conseguiu reduzir o consumo energético sem aumentar a área ocupada. Desta forma, utilizar a SATD com reúso de cálculos se mostra vantajosa quando existe a necessidade do cálculo de múltiplos tamanhos de blocos.

Referências

(2011). *TSMC STANDARD CELL Library TCBN45GSBWPTC*. TSMC.

(2015). *Cadence Encounter RTL Compiler v.8.10*. Cadence. Último acesso em 10/01/2017.

(2015). *NanGate 45nm Open Cell Library*. NanGate. Último acesso em 10/01/2017.

Agaian, S., Sarukhanyan, H., Egiazarian, K., and Astola, J. (2011). *Hadamard Transforms*. SPIE Press Monograph Vol. PM207. SPIE Press.

Anshi, C., Di, L., and Renzhong, Z. (1993). A research on fast Hadamard transform (FHT) digital systems. *TENCON '93*, 3:541–545 vol.3.

Bossen, F. (2012). Common test conditions and software reference configurations. Document JCTVC-K1100, Joint Collaborative Team on Video Coding (JCT-VC) of ITU-T SG16 WP3 and ISO/IEC JTC1/SC29/WG11, Shanghai.

Bossen, F., Bross, B., Suhring, K., and Flynn, D. (2012). HEVC complexity and implementation analysis. *IEEE Trans. Circuits Syst. Video Technol.*, 22(12):1685–1696.

Bräscher, A. B. (2016). Arquiteturas energeticamente eficientes para SATD com tamanho de bloco varíavel no HEVC.

Ghanbari, M. (2003). *Standard Codecs Image Compression to Advanced Video Coding*. IET Telecommunications Series. The Institution of Engineering and Technology.

Kim, J. and Park, T. (2009). A novel VLSI architecture for full-search variable block-size motion estimation. *IEEE Trans. on Consumer Electronics*, 55(2):728–733.

Li, Y., Liu, Z., Ji, X., and Wang, D. (2016). HEVC fast FME algorithm using IME RD-costs based error surface fitting scheme. In *2016 VCIP*, pages 1–4.

Porto, M., da Silva, T., Porto, R., Agostini, L., da Silva, I., and Bampi, S. (2005). Design space exploration on the H.264 4x4 Hadamard transform. In *NORCHIP Conference, 2005. 23rd*, pages 188–191, Oulu, Finland.

Richardson, I. E. G. (2003). *H. 264 and MPEG-4 video compression: video coding for next-generation multimedia*. John Wiley & Sons Inc.

Seidel, I., Bräscher, A. B., Güntzel, J. L., and Agostini, L. (2016a). Energy-efficient SATD for beyond HEVC. In *2016 ISCAS*, pages 802–805, Montreal, Canada.

Seidel, I., Güntzel, J. L., and Agostini, L. (2016b). Coarse grain partial distortion elimination for Hadamard ME in HEVC. In *2016 ICECS*, pages 704–707, Monte Carlo, Monaco.

Silveira, B. (2016). Exploração arquitetural nas métricas de similaridade para codificadores de vídeo do padrão HEVC. Dissertação de mestrado, UCPel.

Silveira, E., Diniz, C., Fonseca, M. B., and Costa, E. (2015). SATD hardware architecture based on 8x8 Hadamard transform for HEVC encoder. In *2015 ICECS*, pages 576–579, Cairo, Egypt.

Soares, L. B., Diniz, C. M., da Costa, E. A. C., and Bampi, S. (2016). A novel pruned-based algorithm for energy-efficient SATD operation in the HEVC coding. In *SBCCI '16*, pages 1–6, Belo Horizonte, Brazil.

Sullivan, G., Ohm, J., Han, W.-J., and Wiegand, T. (2012). Overview of the high efficiency video coding (HEVC) standard. *IEEE Trans. Circuits Syst. Video Technol.*, 22(12):1649–1668.

Synopsys (2016a). *Synopsys Design Compiler, Version L-2016.03-SP1*. Último acesso em 10/01/2017.

Synopsys (2016b). Synopsys vcs, version l-2016.03-sp1. Último acesso em 10/01/2017.

Synopsys (2016c). Synopsys's design compiler user guide, version l-2016.03-sp1.

Walter, F. and Bampi, S. (2011). Synthesis and comparison of low-power architectures for SAD calculation. *26th South Symposium on Microelectronics*, pages 45 – 48.