
# PROJETO DE CIRCUITOS PARA COMPRESSÃO DE IMAGENS NO PLANO FOCAL DE CÂMERAS CMOS

Hugo de Lemos Haas

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia Elétrica, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia Elétrica.

Orientadores: Antonio Petraglia
José Gabriel Rodríguez Carneiro Gomes

Rio de Janeiro
Fevereiro de 2012

PROJETO DE CIRCUITOS PARA COMPRESSÃO DE IMAGENS NO PLANO FOCAL DE CÂMERAS CMOS

Hugo de Lemos Haas

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA ELÉTRICA.

Examinada por:

---

Prof. Antonio Petraglia, Ph.D.

---

Prof. José Gabriel Rodríguez Carneiro Gomes, Ph.D.

---

Prof. Antonio Carneiro de Mesquita Filho, Dr.d'État

---

Prof. Davies William de Lima Monteiro, Ph.D.

RIO DE JANEIRO, RJ – BRASIL
FEVEREIRO DE 2012

Haas, Hugo de Lemos

Projeto de Circuitos para Compressão de Imagens no Plano Focal de Câmeras CMOS/Hugo de Lemos Haas. – Rio de Janeiro: UFRJ/COPPE, 2012.

XIII, 95 p.: il.; 29, 7cm.

Orientadores: Antonio Petraglia

José Gabriel Rodríguez Carneiro Gomes

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia Elétrica, 2012.

Referências Bibliográficas: p. 70 – 72.

1. CMOS. 2. Compressão de Imagens. 3. Plano Focal. I. Petraglia, Antonio *et al.* II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia Elétrica. III. Título.

*Aos meus pais.*

# Agradecimentos

Aos meus pais e à minha família pelo apoio, suporte e educação que me permitiram concluir o mestrado com êxito.

Aos professores, orientadores e amigos Antonio Petraglia e José Gabriel Rodríguez Carneiro Gomes que me orientaram com paciência, clareza e incentivo. Além disto, agradeço pela presteza e disponibilidade de sempre.

Aos colegas e professores do Laboratório de Processamento Analógico e Digital de Sinais pelo incentivo e ajuda.

Aos meus amigos e colegas de classe pela companhia, incentivo, ajuda e momentos de descontração.

Aos meu amigos e companheiros da vida diária pela amizade, incentivo e que, de certa forma, contribuíram muito para esta conquista.

Aos professores e demais colaboradores do Programa de Engenharia Elétrica da COPPE/UFRJ pela formação de alto nível e pela dedicação aos alunos.

Aos examinadores desta dissertação pela paciência e presteza.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

# PROJETO DE CIRCUITOS PARA COMPRESSÃO DE IMAGENS NO PLANO FOCAL DE CÂMERAS CMOS

Hugo de Lemos Haas

Fevereiro/2012

Orientadores: Antonio Petraglia
José Gabriel Rodríguez Carneiro Gomes

Programa: Engenharia Elétrica

Nesta dissertação é apresentado um projeto de um circuito de compressão de imagens no plano focal de câmeras digitais. O método de compressão envolve quantizadores vetoriais e DPCM e é implementado com hardware analógico. Os transdutores e os circuitos de processamento foram projetados para serem construídos com tecnologia CMOS 0.35 $\mu m$ da AMS.

Os resultados foram obtidos em simulações elétricas que levaram em conta as variações do processo de fabricação CMOS, de acordo com os parâmetros oferecidos pelo fabricante. As simulações foram feitas para imagens com $32 \times 32$ pixels, mesma resolução do protótipo testado.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

# CIRCUIT DESIGN FOR FOCAL PLANE IMAGE COMPRESSION IN CMOS CAMERAS

Hugo de Lemos Haas

February/2012

Advisors: Antonio Petraglia
José Gabriel Rodríguez Carneiro Gomes

Department: Electrical Engineering

In this work, we present the design of a focal plane image compression circuit for digital cameras. The compression method uses vector quantization and DPCM and it is implemented with analog hardware. The transducers and the processing circuits are designed to be built with AMS CMOS 0.35 $\mu m$ technology.

The results were obtained in electrical simulations that included the CMOS fabrication process variations, in accordance with the foundry parameters. The simulations were made for images with $32 \times 32$ pixels, which is the same resolution as that of the tested prototype.

# Sumário

# Lista de Figuras

# Lista de Tabelas

# Capítulo 1

# Introdução

## 1.1 Câmeras Digitais

As câmeras digitais revolucionaram o processo de captura de imagens, contribuindo para a popularização da fotografia em ambientes domésticos para entretenimento e em ambientes industriais.

Ao invés de utilizar uma película fotossensível (filme) para o registro das imagens, que requer, posteriormente à aquisição das imagens, um processo químico de revelação e ampliação das cópias, a câmera digital capta as imagens através de um sensor eletrônico e as armazena em cartões de memória ou outro dispositivo de armazenamento eletrônico ou magnético. Estas imagens podem ser visualizadas no monitor da própria câmera ou em algum monitor distante, pois os dados podem ser transmitidos logo após a captura. Estes dados não se deterioram com o tempo, o que aconteceria com um filme fotográfico ou uma ampliação revelada.

Uma vantagem importante das imagens digitalizadas é a facilidade do processamento das mesmas, pois os sinais provenientes do sensor, por serem elétricos, podem ser facilmente processados por meio de circuitos eletrônicos.

### 1.1.1 Características

Uma das características mais importantes de câmeras digitais é a resolução do sensor da câmera, medida pela quantidade de pixels. Esta característica determina o número de elementos que podem ser capturados em cada foto. Atualmente, as câmeras modernas são capazes de capturar até dezenas de milhões de pixels, ou mais comumente, dezenas de megapixels (MP).

Para as câmeras portáteis, uma característica que deve ser levada em conta é o consumo de energia para cada imagem capturada, pois a bateria da câmera tem carga limitada. Outra característica importante, principalmente para aplicações industriais e para utilização no modo vídeo (captura sequencial dinâmica), é a ve-

locidade de captura das imagens, que, em alguns estudos [1], já chega a algums milhares de imagens por segundo.

### 1.1.2 CCD

O CCD (*Charge Coupled Device*) é um sensor para captação de imagens formado por uma matriz de capacitores acoplados. Sob o controle de um circuito externo, cada capacitor transfere sua carga elétrica para um outro capacitor vizinho, até chegar ao circuito que digitaliza esta carga. Os capacitores se carregam quando expostos à luz e sua carga depende do nível e do tempo de exposição. Seu tamanho pode ser bastante reduzido, mas todos os circuitos necessários à quantização do sinal capturado são externos ao elemento sensor. Em geral são utilizados outros chips para este fim.

A tecnologia CCD tem sido usada para captura de imagens desde os anos 60, podendo ser considerada uma tecnologia madura. Esta tecnologia ainda é utilizada largamente tanto para câmeras digitais comerciais quanto para aplicações industriais de alto desempenho.

### 1.1.3 CMOS

O sensor CMOS (*Complementary Metal Oxide Semiconductor*) utiliza elemento fotosensíveis para capturar imagens. Cada pixel da imagem é composto por vários transistores do tipo MOSFET e, em geral, o transdutor de potência luminosa para eletricidade é um fotodiodo. Os circuitos de leitura de cada pixel e sua quantização podem ser realizados no mesmo chip que captura a imagem. Uma das principais vantagens de um sensor CMOS em relação a um sensor CCD é o seu consumo de energia, que pode ser menor.

Atualmente há camêras comerciais que utilizam os dois tipos de sensores. Mais informações e discussões sobre diferenças entre CCD e CMOS podem ser encontradas em [2] e [3].

## 1.2 Compressão de Imagens

As imagens capturadas pelos sensores, após serem digitalizadas, utilizam muitos bits (normalmente oito bits ou mais para cada pixel da imagem) para sua representação. A consequência de não tratar deste aspecto é o aumento do espaço ocupado em uma memória ou aumento da banda necessária para transmissão da imagem. Para reduzir este problema é feita a compressão dos dados.

A compressão pode se basear em inúmeros algoritmos e esquemas com o objetivo de reduzir o número de bits necessários para representar cada imagem capturada

pela câmera. Após a compressão, a imagem pode ser reconstruída a partir dos dados comprimidos e de um algoritmo ou esquema de reconstrução. A compressão pode acarretar em perda de qualidade ou não dependendo do método utilizado.

## 1.3 SoC

Atualmente há uma tendência de que um único circuito integrado agregue várias funções. Esta técnica é conhecida como SoC (*System on a Chip*). A vantagem mais óbvia de se fazer um sistema completo em um chip é o tamanho ocupado pelo circuito e a ausência de conexões entre diferentes chips. Estes aspectos podem diminuir o custo e aumentar a confiabilidade.

Em geral, um SoC é um circuito digital, no qual um processador é agregado com seus periféricos (memórias e interfaces, por exemplo) no entanto utilizaremos esta técnica para projetar os sensores, o processamento e a digitalização em um único circuito. Um escolha importante que devemos fazer quando os sensores, circuitos analógicos e digitais estão no mesmo chip se refere a quais tecnologia podem ser utilizadas. No caso deste trabalho, foi escolhida a tecnologia 0,35 $\mu$m da AMS (austriamicrosystems), que comporta circuitos mistos (analógicos e digitais) e possui etapas no processo de fabricação que permitem a realização de sensores óticos.

### 1.3.1 Compressão de Imagens no Plano Focal

Com a maior utilização de sensores CMOS nas câmeras digitais, tornou-se possível a inclusão de outras funcionalidades no circuito integrado do sensor CMOS. Atualmente diferentes tarefas podem ser executadas sobre a imagem capturada. Este processo é denominado Processamento de Imagens no Plano Focal. Recentemente vários trabalhos vêm sendo desenvolvidos nesta área de processamento de imagens no plano focal de câmeras ([4]–[12]). Este enfoque se torna interessante quando altíssimas velocidades e baixo consumo são necessários.

Este trabalho se concentra na implementação de um sistema de compressão de imagens no plano focal, que pode ser considerado um caso particular de processamento de sinais, utilizando quantizadores vetoriais (Seção 2.4) e DPCM (*differential pulse-code modulation*, Seção 2.2). Os estudos teóricos para um sistema deste tipo iniciaram-se em [13] e neste trabalho são projetados os circuitos necessários para realizar a compressão no plano focal.

Uma tentativa anterior de projetar circuitos capazer de realizar parte do processamento necessário pode ser encontrada em [14]. No entanto a sensibilidade dos circuitos propostos em [14] aos erros introduzidos na fabricação do circuito integrado tornou necessário o reprojeto destes circuitos.

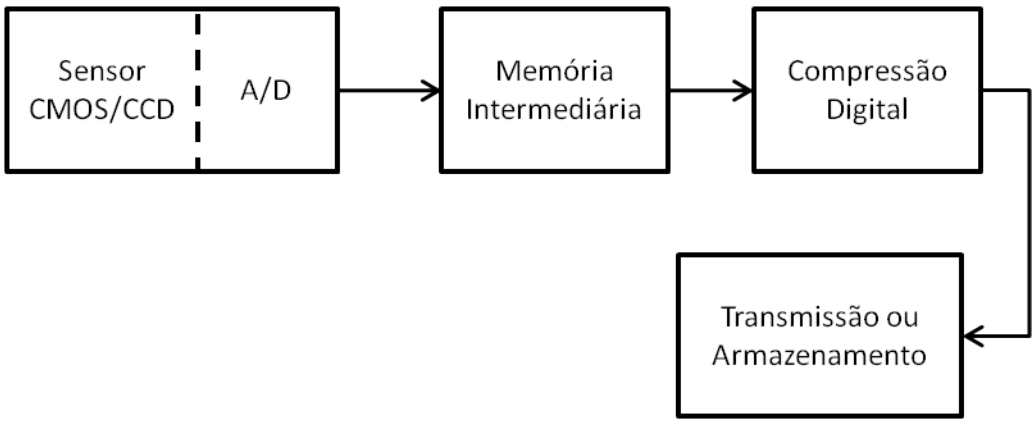



Figura 1.1: Esquema convencional de funcionamento de câmeras digitais.

## 1.4 Proposta

Em câmeras digitais convencionais baseadas em CCD ou CMOS, os dados provenientes dos sensores são imediatamente convertidos para o domínio digital por circuitos conversores analógico-digitais (internos ou não ao sensor). Estes dados são então armazenados em uma memória intermediária para posterior compressão, realizada com um processador digital e transmissão ou armazenamento dos dados comprimidos, conforme mostrado na Figura 1.1.

A compressão de imagens no plano focal usando hardware analógico, anterior à conversão analógico-digital de cada pixel, provê economias de hardware significativas ao custo de alguma degradação da imagem comprimida, como pode ser visto na Figura 1.2. A compressão de imagens no plano focal requer esquemas de baixa complexidade que sejam simples o suficiente para poderem ser implementados na matriz de pixels sem diminuir proibitivamente a resolução do sensor.

O número de transistores em cada bloco foi minimizado para que o espaço ocupado seja o menor possível mantendo a qualidade da imagem aceitável para a aplicação em questão. Como o circuito funciona paralelamente, isto é, cada bloco processa seus pixels individualmente, a velocidade de aquisição da imagem pode ser significativamente maior em comparação com o método convencional.

As operações realizadas pelo circuito são implementadas em modo de corrente, com espelhos de corrente PMOS e NMOS (Seção 4.1), de modo que todo o circuito funcione usando correntes como variáveis de entrada e saída, com exceção das saídas externas do circuito, que são digitais a fim de facilitar a leitura e posterior transmissão dos dados. Os circuitos utilizados também são suficientemente robustos para suportar efeitos parasitas e as variações dos parâmetros dos transistores causadas pelo processo de fabricação. A sensibilidade a tais imperfeições é medida através de simulações de Monte Carlo capazes de indicar estatisticamente os efeitos das imperfeições.

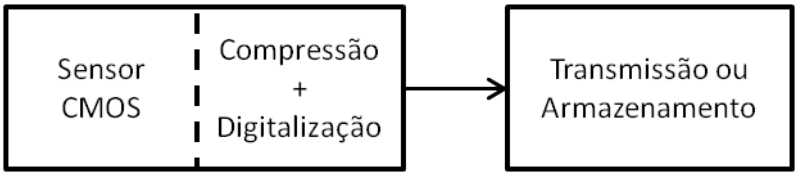



Figura 1.2: Esquema proposto.

## 1.5 Contribuições e Organização do Texto

A principal contribuição deste trabalho é o projeto de um circuito completo capaz de capturar e comprimir imagens em um mesmo circuito integrado. A organização de cada capítulo é resumida a seguir:

O **Capítulo 2** apresenta os fundamentos teóricos do método de compressão de imagens usado neste trabalho.

O **Capítulo 3** descreve, através de diagramas de blocos, como os circuitos farão a compressão da imagem.

O **Capítulo 4** ilustra o funcionamento dos circuitos utilizados neste trabalho através de esquemáticos e simulações.

O **Capítulo 5** apresenta os resultados obtidos através de simulações do circuito e a partir de processamento por *software*, comparando-os. Também são apresentados os resultados obtidos com o chip fabricado.

O **Capítulo 6** conclui o trabalho e apresenta sugestões para melhoria do circuito e continuidade do projeto.

O **Apêndice A** mostra os resultados das simulações de Monte Carlo e das medidas do protótipo fabricado.

O **Apêndice B** fornece os códigos utilizados para realizar as simulações do circuito, o método de compressão e a reconstrução das imagens.

# Capítulo 2

# Método de Compressão

Neste capítulo serão apresentados os fundamentos teóricos do método de compressão de imagens aplicado neste trabalho.

## Definições Iniciais

Consideraremos apenas imagens monocromáticas onde cada pixel é representado por um sinal cuja faixa de valores varia de 0 a 1, onde 0 corresponde ao valor mais escuro (preto) e 1 ao valor mais claro (branco).

As imagens ainda são divididas em blocos de 4×4 pixels. Conforme sugerido em [15], cada bloco é representado por um vetor $\mathbf{y}(n)$, com 16 componentes ($y_1(n)$ a $y_{16}(n)$) obtido a partir de uma varredura de cada bloco como na Figura 2.1. O índice $n$ está na faixa de 1 até $N$, onde $N$ é o número total de blocos na imagem.

O resultado da compressão é o mapeamento de $\mathbf{y}(n)$ (16 pixels analógicos) em uma palavra binária de 15 bits, $\mathbf{b}(n)$. Com isso alcançamos a taxa de $15/16 = 0.9375$ bits/pixel mesmo sem utilizar qualquer método adicional de compressão, como o método de Huffman [16] por exemplo, capaz de reduzir ainda mais a taxa de compressão sem adicionar novas distorções por ser um método de compressão sem erros.

Os 15 bits de $b(n)$ estão divididos da seguinte forma:

1. Quatro bits são obtidos por DPCM da média dos pixels dos blocos (Seção 2.2) ($\mathbf{b_1}(n)$);
2. Quatro bits dos sinais das componentes principais (Seção 2.3) ($\mathbf{b_2}(n)$);
3. Sete bits obtidos pelo quantizador vetorial (Seção 2.4) ($\mathbf{b_3}(n)$).

## 2.1 Quantização Escalar

A quantização escalar consiste em discretizar sinais escalares (ou seja, de 1 dimensão) analógicos como mostrado na Figura 2.2.

Os sinais discretizados, ou índices, possuem um valor finito de representações, desta forma torna-se possível a transmissão e o armazenamento dos dados no domínio digital, usualmente na forma binária.

A quantização escalar pode ser realizada de muitas formas. Neste trabalho usaremos os conversores analógico-digitais do tipo *flash* [17]. Estes funcionam comparando o sinal analógico de entrada com limiares determinados, formando um código do tipo "termômetro", de $N_t$ bits. Como este código utiliza mais bits que o necessário, o dado de saída ainda pode ser codificado, através de circuitos lógicos formados por portas lógicas XOR ou XNOR, para que seja possível representá-lo com apenas $N_b$ bits, como mostrado na Figura 2.3.

O níveis (limiares) de quantização podem ser uniformes (igualmente espaçados entre si) ou não uniformes (com espaçamento entre limiares variável), dependendo da aplicação. Normalmente a quantização não uniforme é utilizada quando se tem alguma informação acerca da estatística ou comportamento do sinal analógico. Esta também obtém, em média, menor erro de quantização do que a quantização uniforme

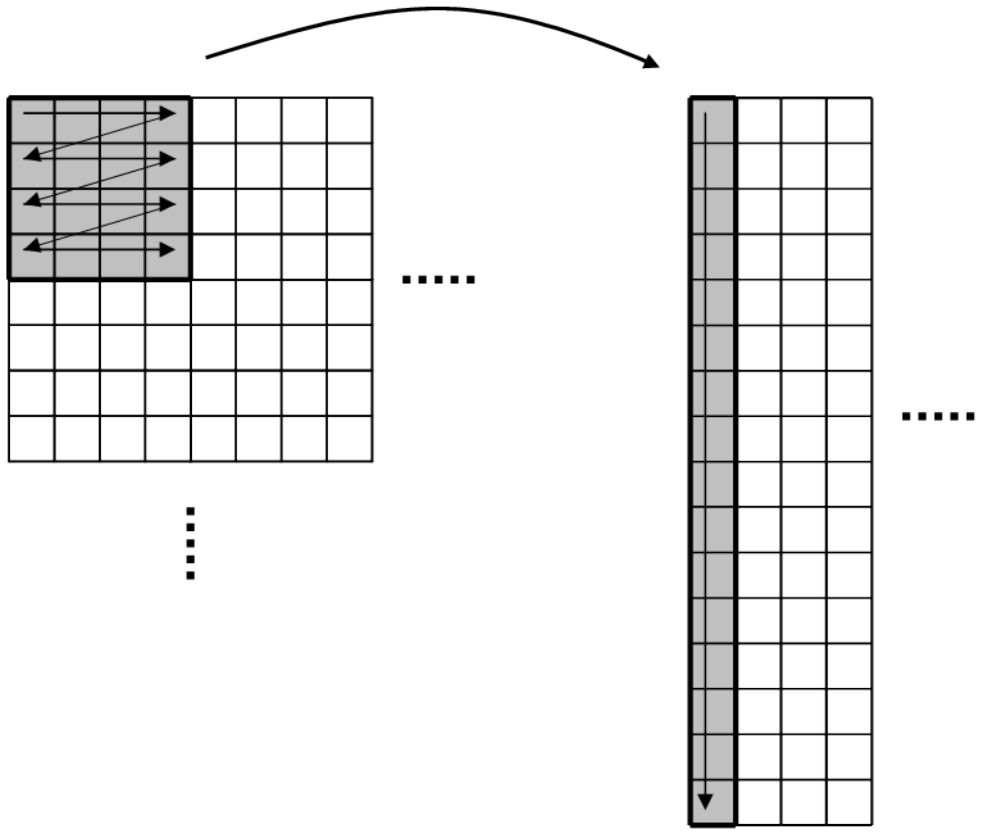
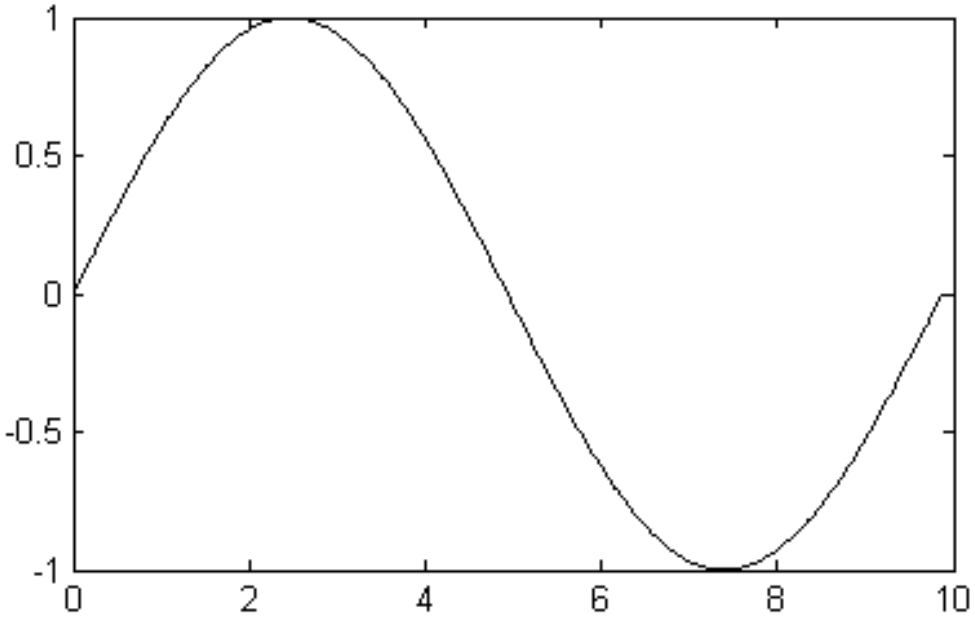
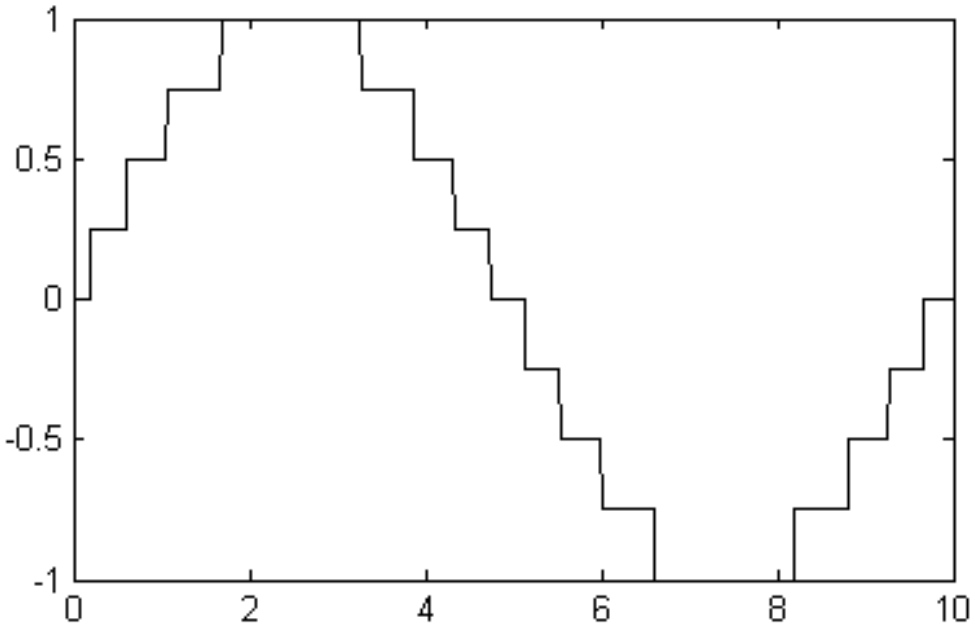


Figura 2.1: Varredura do bloco.




(a)




(b)

Figura 2.2: Quantização escalar: (a) sinal original; (b) sinal quantizado reconstruído.

para o mesmo número de bits, pois os limiares são otimizados para tal, embora seu projeto e implementação sejam mais complexos.

A escolha por conversores do tipo *flash* se deu por sua elevada velocidade de operação e por permitir que a quantização seja não uniforme, mais adequada à quantização dos dados. Isto ocorre pois é implementado com comparadores em paralelo.

A quantidade de limiares ($N_t$) depende do número de bits ($N_b$) com os quais o sinal será representado. Estes parâmetros seguem a seguinte relação:

$$N_t = 2^{N_b} - 1$$

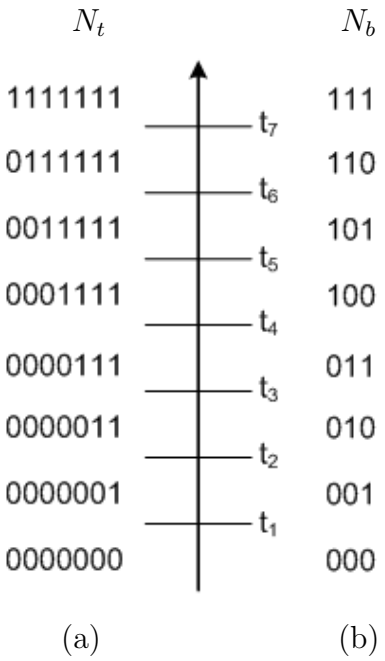



Figura 2.3: Códigos: (a) termômetro; (b) codificada.

A reconstrução do sinal é feita buscando o valor quantizado a partir do índice da tabela de reconstrução, também chamada de dicionário. O dicionário contém os valores que melhor representam as variáveis analógicas em cada intervalo entre os limiares ($\mathbf{t}$) de forma a minimizar o erro de quantização. O erro de quantização é definido como a média das diferenças entre o sinal reconstruído e o sinal original elevadas ao quadrado.

## 2.2 DPCM

A técnica de compressão DPCM (*Differential Pulse-Code Modulation*) baseia-se em quantizar a diferença entre amostras adjacentes ao invés de quantizar cada amostra

individualmente. O DPCM se torna vantajoso quando há uma correlação entre dados adjacentes de um determinado sinal, como acontece em grande parte dos sinais, como por exemplo, linhas horizontais de imagens naturais.

Para exemplificar, consideraremos a linha 256 da imagem "lena.mat" de tamanho 512×512 pixels, conforme indicado na Figura 2.4. A Figura 2.5 mostra a distribuição das amostras de luminância (e seu histograma) e a Figura 2.6 mostra as diferenças entre amostras vizinhas horizontalmente (e seu histograma). Note que a distribuição da Figura 2.6(b) é mais concentrada nas pequenas diferenças, com isso, podemos representar estas diferenças com menos bits do que as amostras sem tratamento da Figura 2.5(a). Se fizermos uma distribuição dos limiares de quantização de forma a privilegiar as menores diferenças (quantização não uniforme [17]), poderemos comprimir a informação com menor perda e taxa.

Figura 2.4: Indicação da linha 256 da figura "lena.mat".

Utilizaremos esta técnica para comprimir a média das componentes de $\mathbf{y}(n)$ ($s(n)$), considerando a correlação existente entre $s(n)$ e $s(n+1)$.

Definimos $e(n)$ como a diferença a ser quantizada:

$$e(n) = s(n) - \hat{s}(n-1)$$

onde $\hat{s}(n)$ é a reconstrução de $s(n)$:

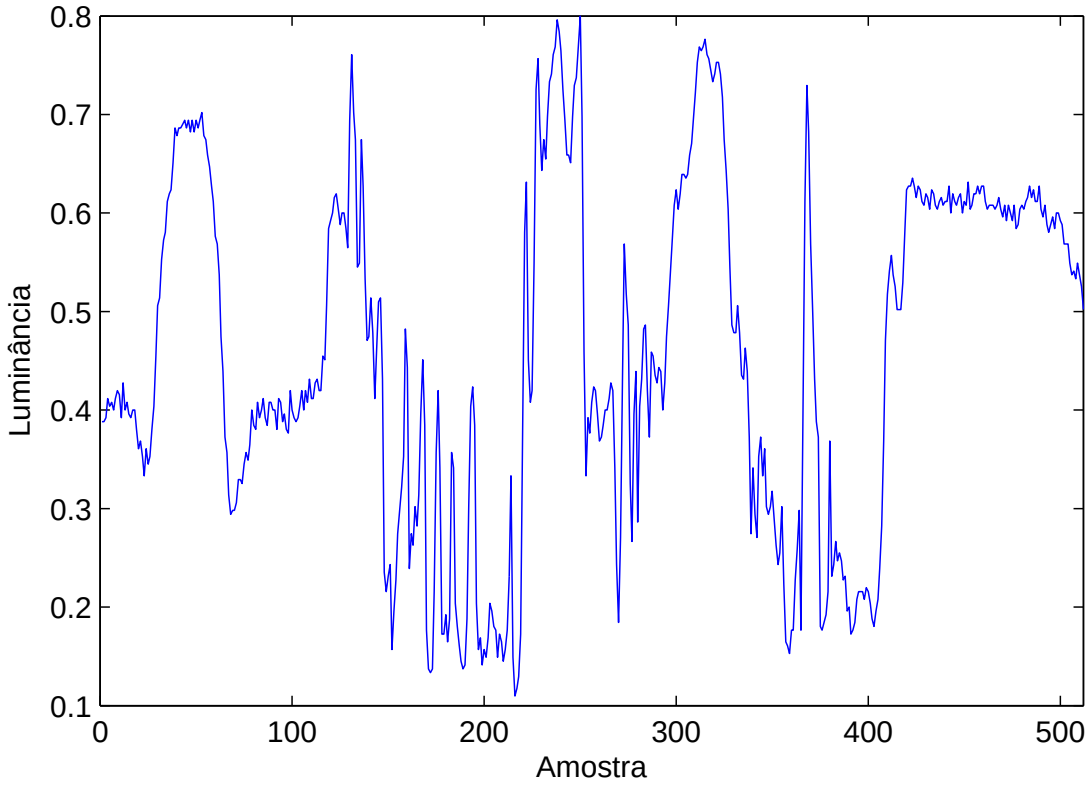
$$\hat{s}(n) = \hat{s}(n-1) + \hat{e}(n)$$

e $\hat{e}(n)$ é a reconstrução de $e(n)$.

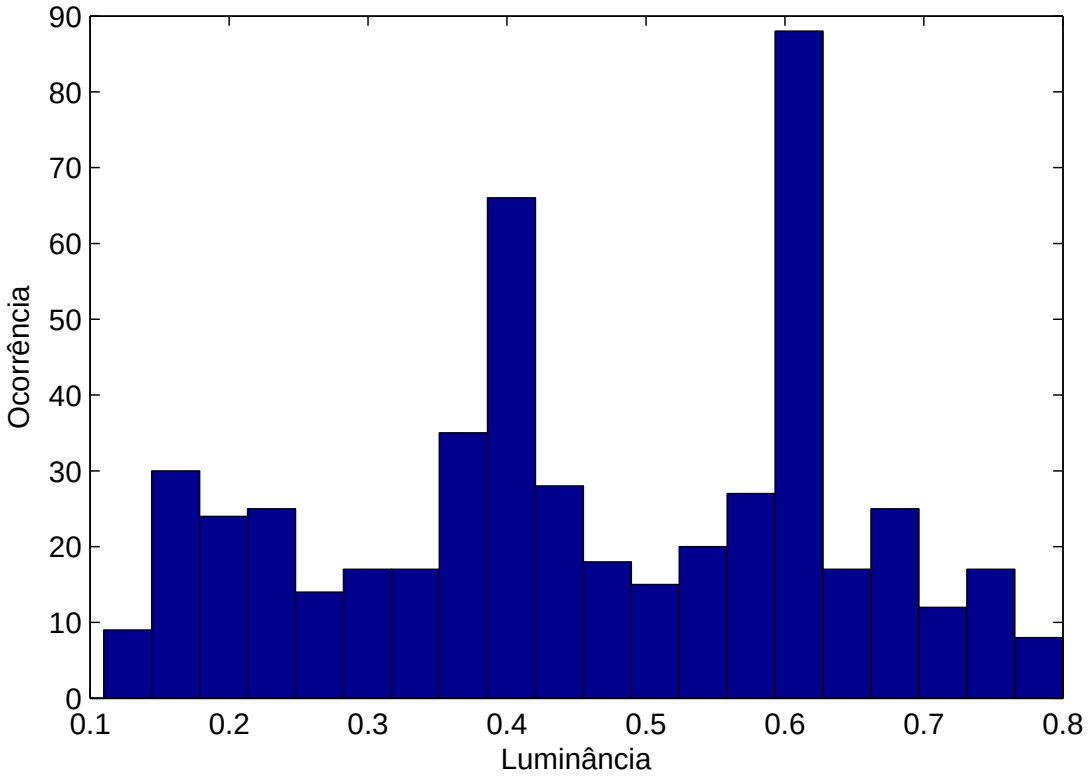
A quantização é feita conforme a Seção 2.1, resultando no vetor binário $\mathbf{b_1}(n)$ de 4 componentes, com a diferença que o primeiro bit é referente ao sinal de $e(n)$. Os limiares de decisão do quantizador utilizado no DPCM são representados pelo vetor coluna:

$$t_0 = [0,0125 \quad 0,0375 \quad 0,0750 \quad 0,1250 \quad 0,1875 \quad 0,2750 \quad 0,4000]^T$$

E os níveis de reconstrução são representados pelo vetor coluna:

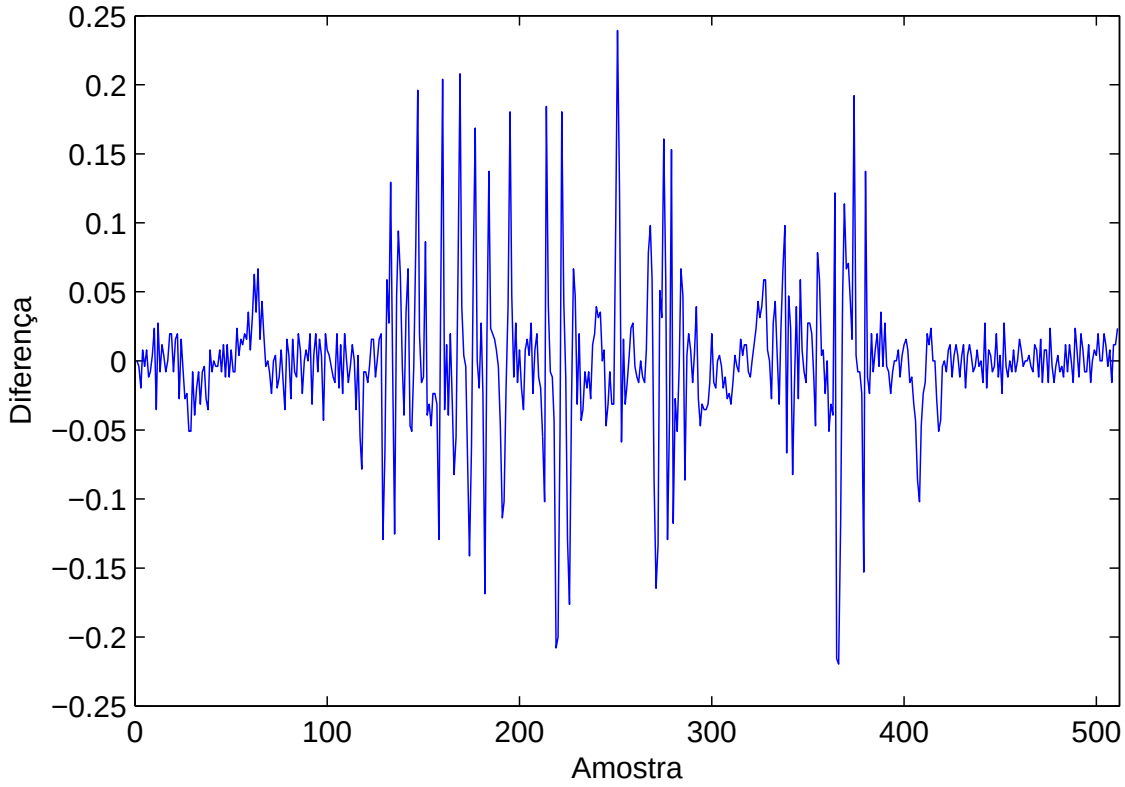



(a)

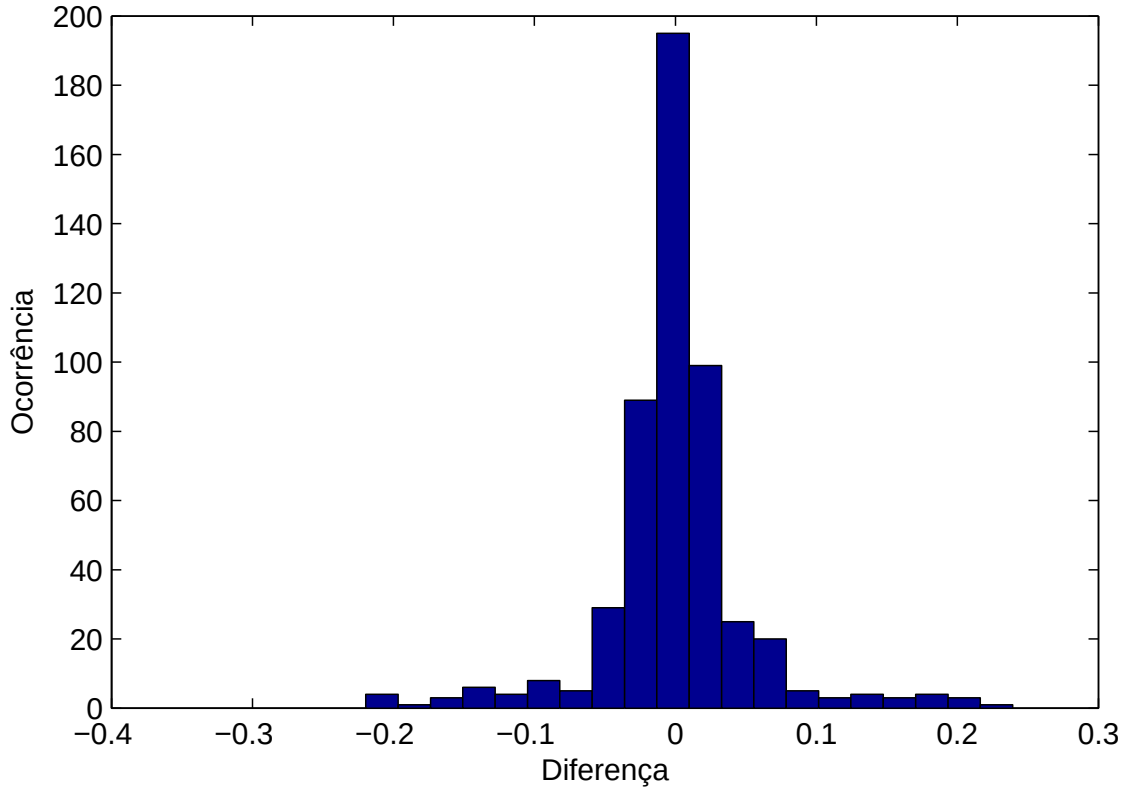



(b)

Figura 2.5: (a) amostras de luminância na linha 256 da imagem "lena.mat"; (b) histograma das amostras.




(a)




(b)

Figura 2.6: (a) diferenças entre amostras vizinhas; (b) histograma da diferença entre amostras vizinhas.

$$Y_0 = [0,00625 \quad 0,0250 \quad 0,05625 \quad 0,1000 \quad 0,1500 \quad 0,2250 \quad 0,3250 \quad 0,4687]^T$$

Estes valores foram encontrados de forma a minimizar o erro de quantização considerando a compressão DPCM em um banco com 21 imagens diversas e o arredondamento que permite sua implementação em hardware analógico.

A Figura 2.7 mostra as diferenças ($e(n)$) e os limiares de decisão e a Figura 2.8 mostra uma parte da imagem "lena.mat" (32×32 pixels) reconstruída após compressão por DPCM com os parâmetros desta seção.

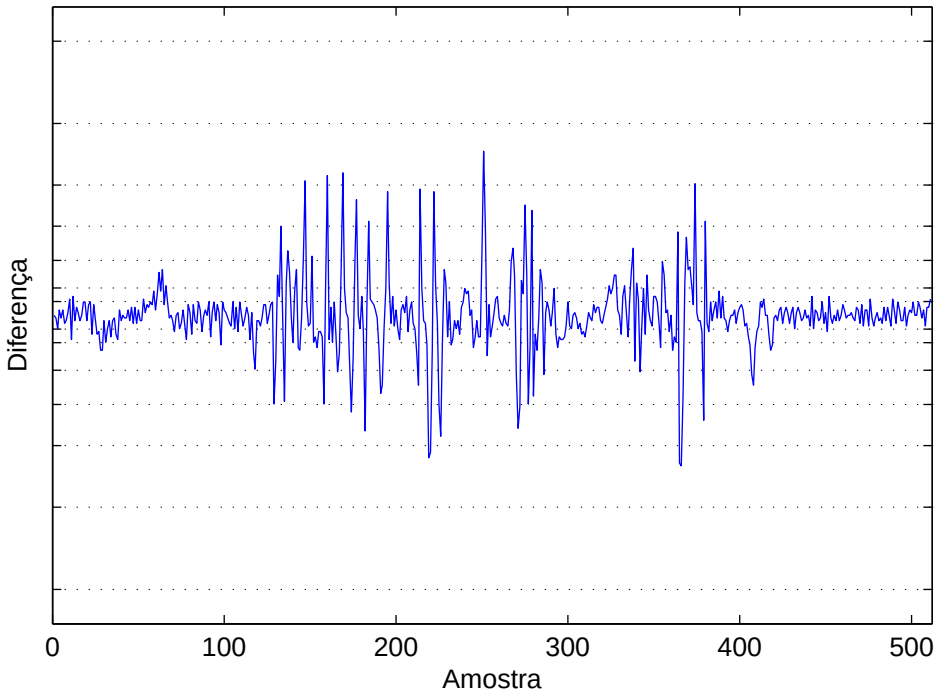
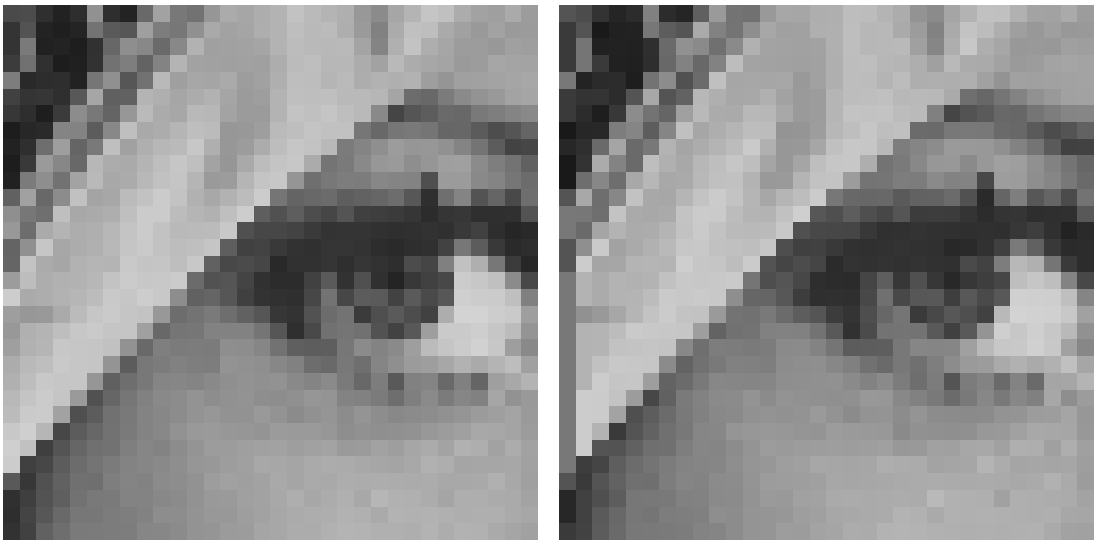



Figura 2.7: Níveis de quantização (linhas horizontais) e diferenças entre amostras vizinhas.

É possível notar que a primeira coluna da imagem comprimida está escurecida. Isto ocorre pois assuminos o valor 0 para $\hat{s}(0)$ que é o ponto de partida para o DPCM.

### 2.2.1 Correção do DPCM

Em uma implementação experimental, o DPCM acumulará erros ao longo dos blocos. O decodificador não consegue compensar estes erros. Para corrigir a média do erro de reconstrução, foi proposto um método através do qual é possível estimar o erro de reconstrução acumulado do DPCM no codificador.



(a) (b)

Figura 2.8: Exemplo de imagem comprimida por DPCM (a) imagem original; (b) imagem comprimida.

O método consiste em adicionar alguns circuitos de DPCM no final das linhas horizontais de blocos. Estes circuitos de DPCM têm suas entradas iguais a 0.

Como a entrada destes blocos é conhecida, podemos descontar a diferença entre a estimativa do decodificador e o valor reconstruído no codificador ($\epsilon$). Para tal, podemos descontar o erro médio ao longo da linha através da seguinte equação:

$$\hat{s}_c(n) = \hat{s}(n) + \frac{\epsilon.n}{N}$$

onde $\hat{s}_c(n)$ é a média corrigida do bloco $n$.

A Figura 2.9 mostra simulações de compressão DPCM com erros aleatórios, que representam erros de implementação, de variância igual a 0,01, injetados na reconstrução de uma parte da imagem "lena.mat" (32×32 pixels) e a correção utilizando três pixels extras (com valor 0) para a correção do DPCM ao final da linha. O resultado de 20 rodadas da simulação está na Tabela 2.1. O parâmetro para comparação usado foi o erro médio quadrático (MSE), dado pela média do quadrado de todas as diferenças entre os pixels originais e os reconstruídos da imagem.

Tabela 2.1: Resultados para a correção de DPCM.

| Imagem | MSE médio ($\times 10^3$) | desvio padrão ($\times 10^3$) |
|---|---|---|
| (a) Original | 0 | - |
| (b) DPCM sem erro | 0,913 | - |
| (c) DPCM sem correção | 2,483 | 0,327 |
| (d) DPCM com correção | 1,487 | 0,093 |

Como pode-se observar, a correção fez com que o MSE médio da imagem recons-

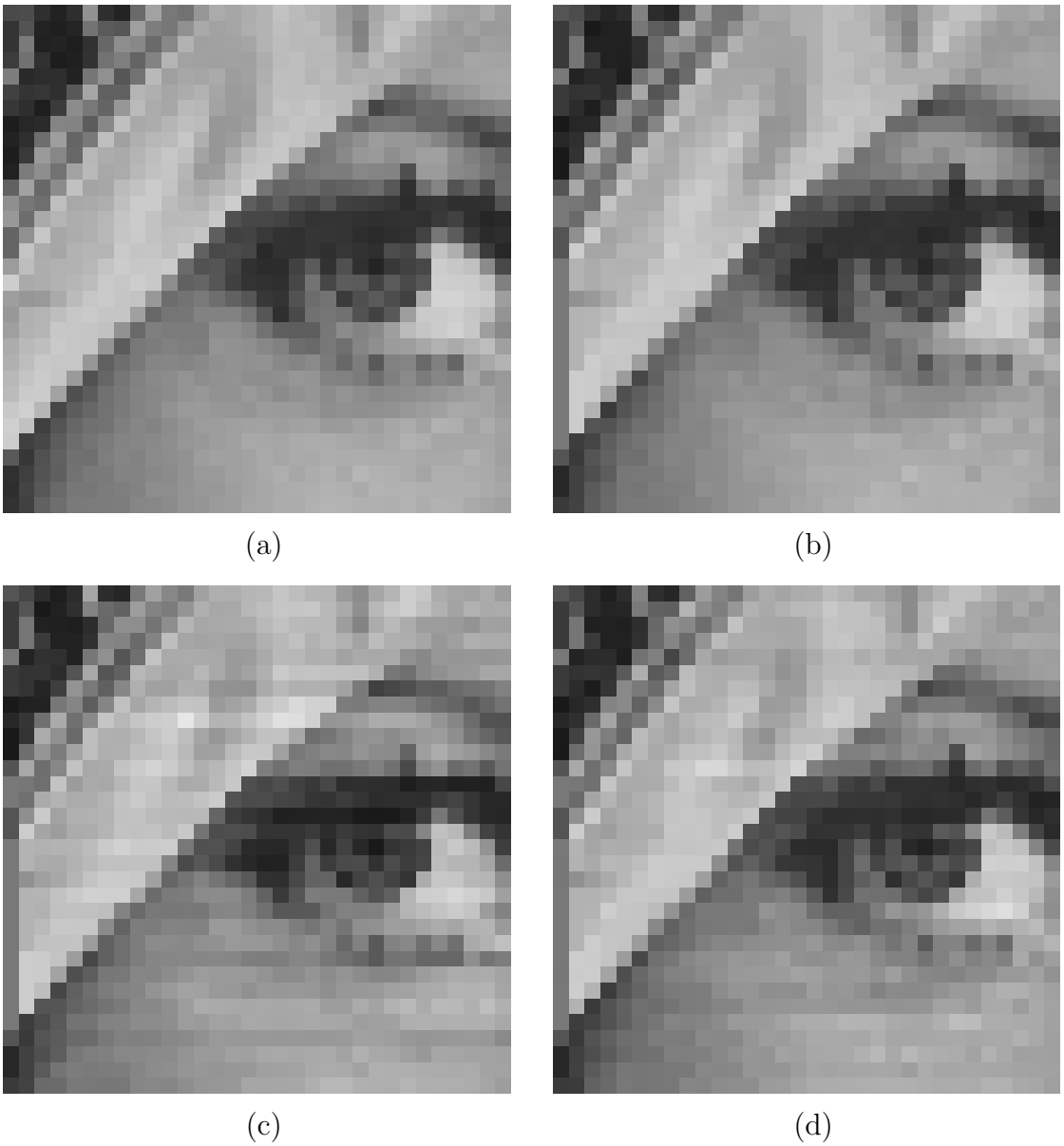


Figura 2.9: Exemplos de imagem comprimida por DPCM com erro de reconstrução e correção: (a) imagem original; (b) imagem comprimida sem erro; (c) imagem comprimida com erro e sem correção; (d) imagem comprimida com erro e com correção.

truída diminuísse em 40%, aproximando-se do MSE obtido quando nenhum erro é injetado. A correção também contribuiu para que a imagem fosse mais uniforme entre várias rodadas, como pode ser notado a partir da redução desvio padrão do MSE obtido na simulação (reduzido em 71%).

## 2.3 Análise de Componentes Principais (PCA)

Assumindo a divisão da imagem em blocos com 4×4 pixels, as componentes principais ($\mathbf{p}(n)$) do n-ésimo bloco formam um vetor que representa o vetor de luminâncias

daquele bloco ($\mathbf{y}(n)$) sob uma transformação linear ($\mathbf{H}$), de forma que cada componente do vetor contenha informação com menos redundância que o sinal original. Desta forma, com apenas algumas componentes principais, é possível uma representação razoável do sinal original, ou seja, diminuímos as dimensões da variável a ser comprimida com perda mínima.

Neste trabalho usaremos quatro componentes principais e a matriz $\mathbf{H}$ de transformação com elementos inteiros baseando-nos em estudos que mais tarde vieram a ser incorporados na codificação H.264 [18]. O uso de elementos inteiros na matriz $\mathbf{H}$ é importante para a implementação do produto interno em hardware analógico, pois serão realizados com espelhos de corrente (Seção 4.1). O número de componentes de $\mathbf{y}(n)$ é reduzido da seguinte forma para $\mathbf{p}(n)$:

$$\mathbf{p}(n) = \mathbf{DHy}(n)$$

onde:

$$\mathbf{H} = \begin{bmatrix} 2 & 1 & -1 & -2 & 2 & 1 & -1 & -2 & 2 & 1 & -1 & -2 & 2 & 1 & -1 & -2 \\ 2 & 2 & 2 & 2 & 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 & -2 & -2 & -2 & -2 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

e $\mathbf{D}$ é usado para aproximar as faixas de valores de $\mathbf{p}(n)$, pois $p_1(n)$ e $p_2(n)$ possuem, em geral, valores maiores que $p_3(n)$ e $p_4(n)$. Desta forma, o projeto e a implementação do quantizador vetorial são simplificados.

$$\mathbf{D} = \begin{bmatrix} 0.5 & 0 & 0 & 0 \\ 0 & 0.5 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

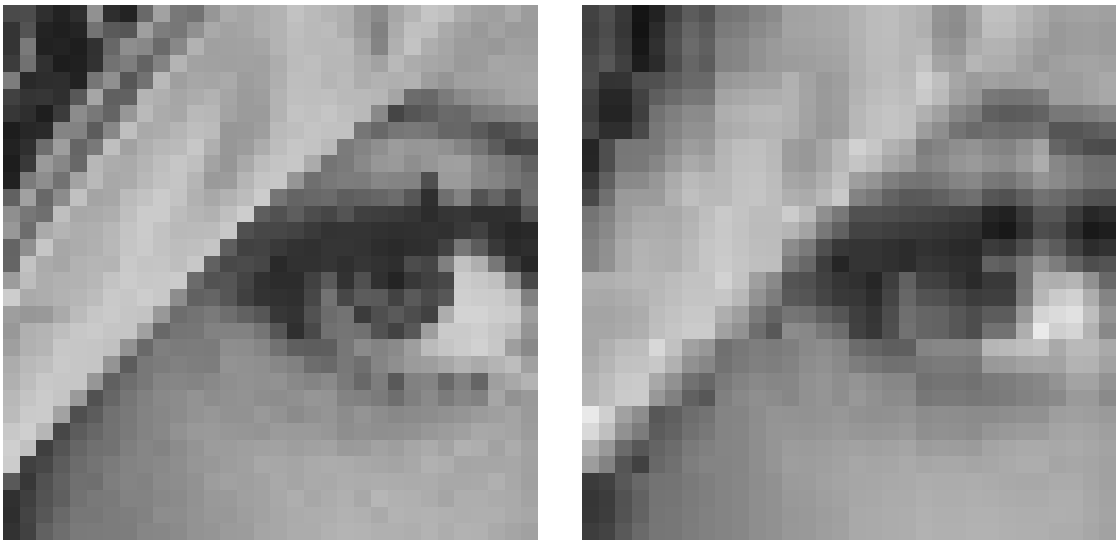
Os elementos da matriz $\mathbf{H}$ são calculados de forma a maximizar a variância das componentes principais, concentrando assim a informação nestas componentes. Como a soma dos elementos ao longo de uma linha é zero, o valor médio de $\mathbf{y}(n)$ é descartado.

A reconstrução do sinal original é realizada aplicando a transformação inversa da transformação original ($\mathbf{H}^T$) e somando a média reconstruída pelo sistema de DPCM:

$$\hat{\mathbf{y}}(n) = \mathbf{H^T}\ \mathbf{p}(n) + \hat{s}(n)\ \ [1\ ...\ 1]_{16}^T$$

onde $\hat{\mathbf{y}}(n)$ é o sinal reconstruído e $\hat{s}(n)$ é a média reconstruída do bloco.

Como algumas componentes são descartadas, a reconstrução não seria perfeita mesmo que não houvesse quantização: o erro é dado pela diferença de luminância de cada pixel reconstruído em relação ao seu valor original. A Figura 2.10 mostra uma parte da imagem "lena.mat"original e comprimida apenas pelo método PCA. As médias dos blocos não sofreram compressão.



(a) (b)

Figura 2.10: Exemplo de imagem comprimida por PCA (a) imagem original; (b) imagem comprimida.

Os sinais das componentes de $\mathbf{p}(n)$ possuem entropia unitária e portanto não podem ser comprimidos, por isso são transmitidos diretamente através de $\mathbf{b}_2(n)$ composto por 4 bits, um para cada componente. O valor da $n$-ésima componente de $\mathbf{b}_2$ é 0 quando a componente possui sinal negativo e 1 quando a componente possui sinal positivo.

## 2.4 Quantização Vetorial

A quantização vetorial (VQ) consiste em classificar uma variável de $M$ dimensões, representando-a por uma variável escalar discreta, ou índice ($i(n)$). Para exemplificar, podemos considerar um sinal de 2 dimensões (ou seja, contido em um plano). Para encontrar o índice que representará certo ponto neste plano, primeiro é necessário determinar a região em que o ponto em questão se encontra. Cada região determinada por seus limites espaciais é representada por um índice e possui um centróide $c_k$, como na Figura 2.11.

O vetor que utilizaremos como entrada para o quantizador vetorial, $\mathbf{x}(n)$, é definido como o módulo de $\mathbf{p}(n)$:

$$\mathbf{x}(n) = |\mathbf{p}(n)|$$

Deste modo, podemos considerar o VQ formado de uma função codificadora $\alpha$, tal que:

$$\alpha : \mathbb{R}^M \rightarrow 1, 2, ..., K$$

$$i(n) = \alpha(\mathbf{x}(n))$$

e de uma função decodificadora $\beta$, tal que:

$$\beta : 1, 2, ..., K \rightarrow c_1, c_1, ..., c_K$$

$$\hat{\mathbf{x}}(n) = \beta(i(n))$$

onde $c_k$ é cada elemento do dicionário de reconstrução.

A reconstrução do vetor quantizado é realizada de uma forma semelhante à reconstrução escalar. Para cada índice $i(n)$ há um vetor de reconstrução (centróide) correspondente no dicionário.

O erro de reconstrução, neste caso, é o quadrado da distância entre o vetor

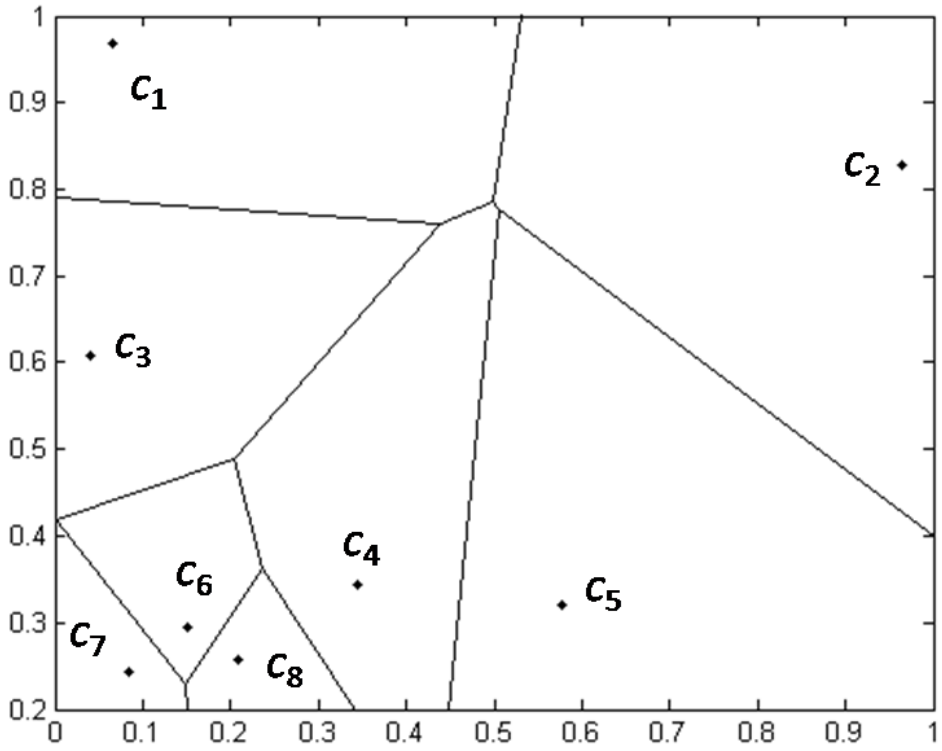



Figura 2.11: Quantização vetorial em 2D com dados aleatórios.

quantizado e o reconstruído. O parâmetro que utilizaremos para representar este erro é a distorção $D$:

$$D = \frac{1}{N} \sum_{n=1}^{N} \|x(n) - \hat{x}(n)\|$$

O comprimento da representação binária de $i(n)$ é $l(i(n))$, e a média (taxa $R$) dos comprimentos binários ($l(n)$) tem um valor esperado definido como entropia $H$:

$$H = -\sum_{k=1}^{K} P(i(n) = k) log_2 P(i(n) = k)$$

Os vetores $\mathbf{c_k}$ e comprimentos $l(k)$ do dicionário são otimizados para minimizar $D$ e $H$ a partir da expressão lagrangeana [15]:

$$J = D + \lambda H$$

### 2.4.1 VQ com Complexidade Restrita

Como a complexidade de implementação não permite uma operação de busca completa [17] pelos vetores do dicionário, a implementação da função $\alpha$ é feita através da implementação de uma rede neural artificial. As redes neurais podem ser usadas como métodos de solução de problemas em que se deseja separar dados de entrada em diferentes classes. No caso da compressão de vetores, deseja-se encontrar, para cada vetor, que índice o representa melhor (Seção 2.4).

Os parâmetros da rede são otimizados a partir de um conjunto de dados de teste, a fim de se encontrar os melhores parâmetros para uma dada função custo. Para realizar a quantização vetorial usaremos uma rede neural do tipo *Linear MLP* (*Linear Multilayer Perceptron* ou Perceptron Multicamada Linear), pois é o que possui menor complexidade de implementação e portanto é aplicável à compressão no plano focal, onde a complexidade é restrita. Testes teóricos de compressão de imagens usando esta topologia estão em [15].

Os parâmetros ótimos ($\mathbf{U}$ e $\mathbf{t}$) foram encontrados de forma a minimizar a distorção ($D$) e a entropia ($H$) (Seção 2.4) para a quantização vetorial em um conjunto de componentes principais retiradas de um grupo de imagens de teste. Detalhes desta otimização estão em [15].

Para implementar o VQ com complexidade restrita, primeiro definimos o vetor $\mathbf{f}(n)$ como:

$$\mathbf{f}(n) = \mathbf{U}\mathbf{x}(n)$$

onde $\mathbf{U} \in \mathbb{R}^{MxM}$.

Aplicamos quantização escalar sobre a m-ésima componente de $\mathbf{f}(n)$ ($f_m(n)$) usando limiares de quantização:

$$\mathbf{t}_m = [t_{m,1} \;\; t_{m,2} \; ... \; t_{m,T}]$$

Isto irá fazer com que o espaço vetorial de entrada não seja mais dividido segundo um diagrama de Voronoi [19]: as fronteiras serão definidas com retas (ao invés de segmentos de reta). A Figura 2.12 mostra um exemplo de divisão de espaço em 2D.

Os vetores $\mathbf{t}_m$ têm comprimento $T(m)$:

$$T(1) = 7$$

$$T(2) = 3$$

$$T(3) = 1$$

$$T(4) = 1$$

A saída $\mathbf{b_3}(n)$ possui 7 bits divididos da seguinte maneira: 3 para $f_1(n)$, 2 para $f_2(n)$, 1 para $f_3(n)$ e 1 para $f_4(n)$. Estes bits são gerados com 4 operações de quantização escalar, como as descritas na Seção 2.1.

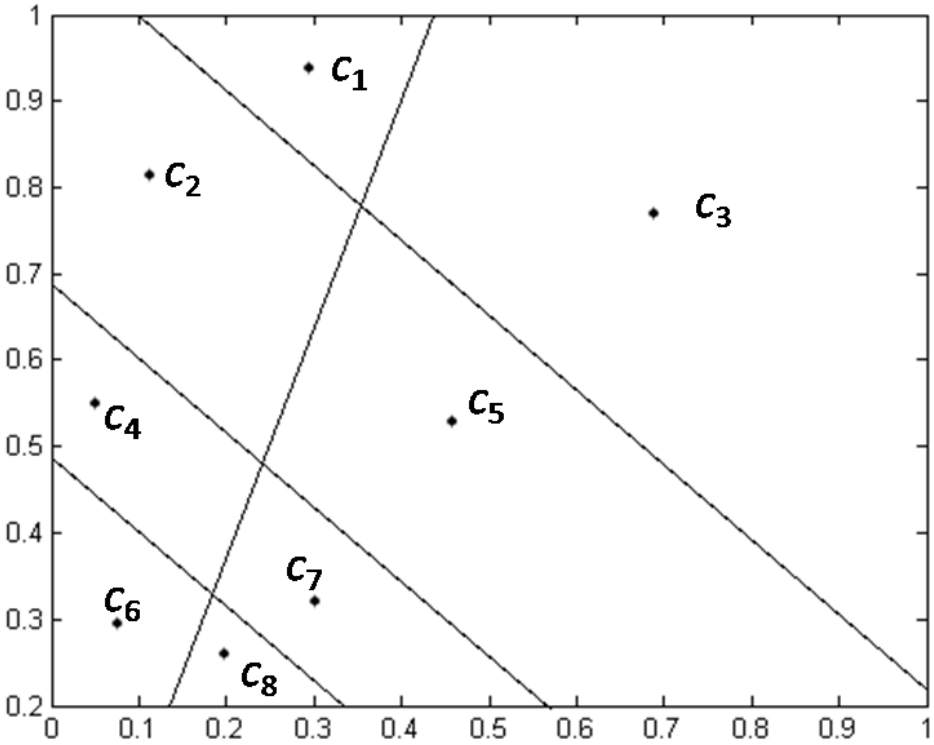



Figura 2.12: Quantização vetorial em 2D com restrição de complexidade.

Os valores de $\mathbf{U}$ são calculados pelo método de componentes principais e os valores de $\mathbf{t}$ foram ajustados por um algoritmo de otimização não-linear (Nelder-Mead [15]) de forma a minimizar $J$. Estes estudos estão mais detalhados em [15] e resultam nos valores a seguir:

$$\mathbf{U} = \begin{bmatrix} 0.54 & 0.73 & 0.23 & 0.36 \\ -0.72 & 0.49 & -0.37 & 0.34 \\ -0.13 & -0.43 & 0.42 & 0.79 \\ -0.42 & 0.23 & 0.80 & -0.36 \end{bmatrix}$$

$$\mathbf{t} = \begin{bmatrix} 0,030 \\ 0,073 \\ 0,122 \\ 0,189 \\ 0,286 \\ 0,401 \\ 0,593 \\ -0,151 \\ -0,013 \\ 0,106 \\ 0,324 \\ 0,006 \end{bmatrix}$$

No entanto valores com tal precisão não são realizáveis na prática por hardware analógico simples, então os valores são aproximados para múltiplos de 0,5 para $\mathbf{U}$ e para múltiplos de 0,05 para $\mathbf{t}$. Isto se deve à implementação destes valores por espelhos de corrente e por correntes de referência, respectivamente. Os estudos para o arredondamento destes valores, que envolveram um compromisso entre precisão e área (para valores de $\mathbf{U}$) ou consumo (para os valores de $\mathbf{t}$), encontram-se em [20]. Finalmente, os valores usados nos circuitos são os seguintes:

$$\mathbf{U} = \begin{bmatrix} 0,5 & 0,5 & 0,0 & 0,5 \\ -0,5 & 0,5 & -0,5 & 0,5 \\ 0 & -0,5 & 0,5 & 1,0 \\ -0,5 & 0,0 & 1,0 & -0,5 \end{bmatrix}$$

$$\mathbf{t} = \begin{bmatrix} 0 & 0,05 & 0,1 & 0,2 & 0,3 & 0,4 & 0,6 & -0,15 & 0 & 0,1 & 0,05 & 0 \end{bmatrix}^T.$$

# Capítulo 3

# Implementação

Neste capítulo mostraremos como os circuitos realizarão a compressão das imagens utilizando o método de compressão apresentado no Capítulo 2. Serão usados diagramas de blocos para mostrar a função de cada circuito utilizado e as interfaces entre estes circuitos. A Seção B.1 apresenta o código fonte do programa que implementa a codificação e a decodificação.

As variáveis descritas neste capítulo são representações dos sinais elétricos que estão nos circuitos. As variáveis podem estar representando correntes elétricas, tensões binárias ou tensões utilizadas por espelhos de corrente (Seção 4.1) para copiar alguma corrente que represente uma variável.

## 3.1 Visão Geral

O circuito de imageamento é dividido em blocos de $4 \times 4$ pixels conforme previsto no Capítulo 2. Estes blocos estão espacialmente dispostos no circuito integrado como mostrado na Figura 3.1.

Os blocos estão interligados horizontalmente e o fluxo de dados é da esquerda para a direita, tornando possível a realização do DPCM (Seção 2.2). Os blocos possuem saídas independentes, $\mathbf{b}(n)$. As saídas podem ser multiplexadas por linha, facilitando a disposição espacial das conexões e ainda podem ser transformadas em um sinal serial. Desta forma é necessário apenas um terminal de saída no circuito integrado.

## 3.2 Bloco 4 x 4 Pixels

A Figura 3.2 mostra o diagrama geral de um bloco $4 \times 4$ pixel.

O primeiro passo é converter a imagem incidente no sensor em um sinal tratável eletronicamente. Para tal, cada circuito de leitura (Seção 4.2) contém um fotodi-

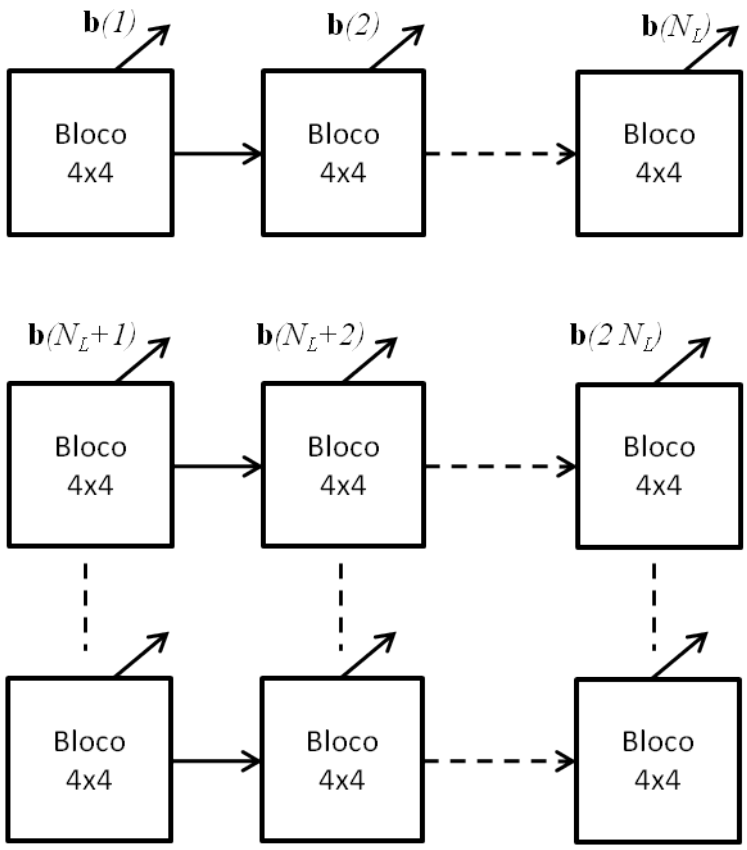



Figura 3.1: Disposição dos blocos $4 \times 4$.

odo, transdutor usado para transformar a potência luminosa incidente em corrente elétrica, e o restante do circuito necessário para a saída em modo de corrente. Neste trabalho representaremos cada pixel da imagem com uma corrente elétrica proporcional à potência luminosa incidente sobre o elemento fotosensível. A responsividade utilizada foi de $0,3$ A/W, conforme sugerido em [21] para comprimentos de onda de 550 nm (verde).

Cada bloco contém 16 circuitos de leitura e CDS (*Correlated Double Sampling*), onde os elementos fotosensíveis estarão dispostos espacialmente formando uma matriz $4\times4$. O circuito de CDS (Seção 4.3) é utilizado para minimizar efeitos indesejáveis à captura realizada pelo circuito de leitura.

Os fotodiodos devem possuir a mesma distância entre si, mesmo que pertençam a blocos diferentes, para que não seja introduzida nenhuma distorção na imagem gerada. Os sinais provenientes destes circuitos formam o vetor $\mathbf{y}(n)$, onde $n$ é o identificador do bloco.

Em seguida, o circuito que realiza o PCA (Seção 2.3) transforma o vetor $\mathbf{y}(n)$ em $\mathbf{p}(n)$. Os sinais das componentes do vetor $\mathbf{p}(n)$ formam $\mathbf{b}_2(n)$ e $\mathbf{x}(n)$ é o vetor cujas componentes são iguais aos módulos das componentes de $\mathbf{p}(n)$.

As componentes principais são utilizadas pelo VQ (Seção 2.4) para formar $\mathbf{b}_3(n)$.

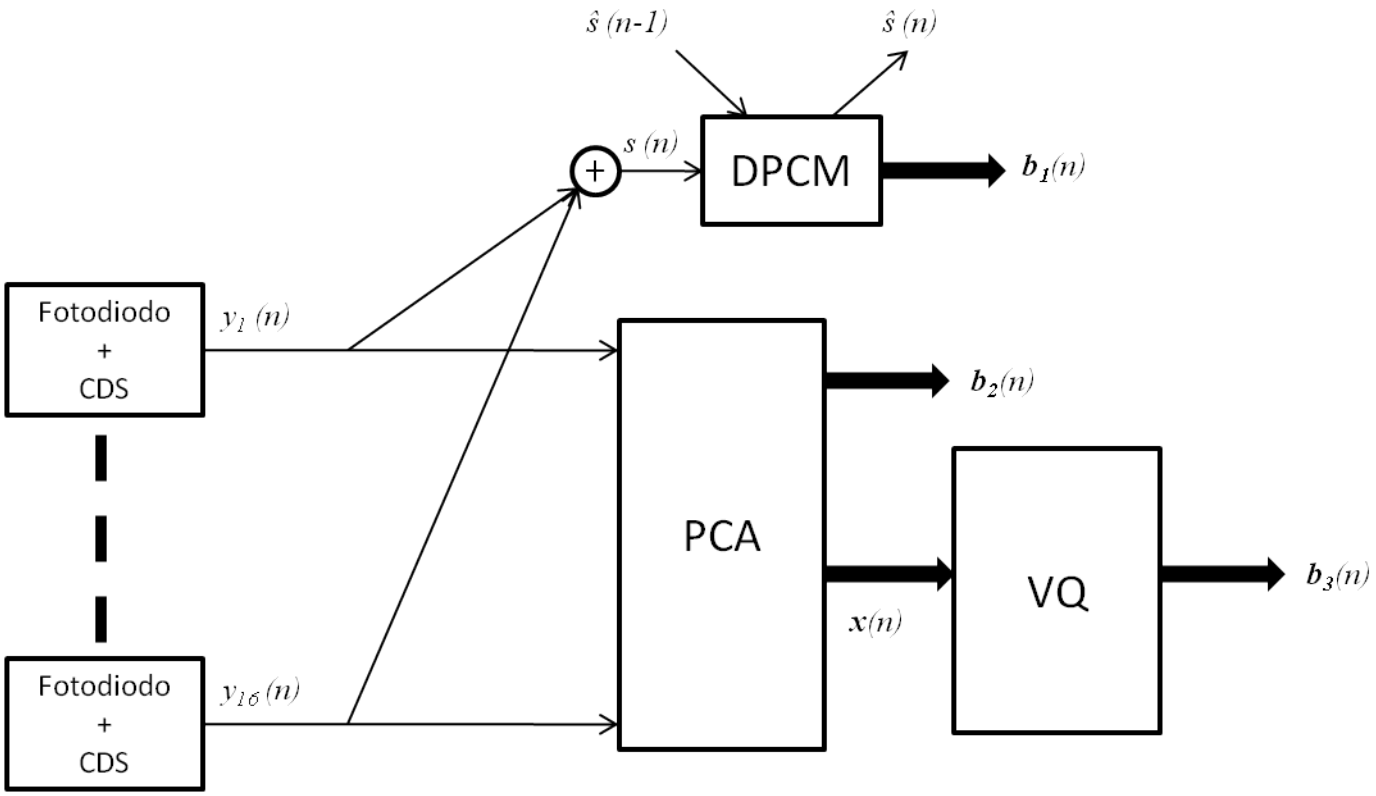



Figura 3.2: Diagrama geral do bloco 4 × 4.

Paralelamente, a média das componentes de $\mathbf{y}(n)$, $s(n)$, em conjunto com a média reconstruída do bloco anterior ($\hat{s}(n-1)$) são utilizadas pelo DPCM (Seção 2.2) para formar a saída $\mathbf{b_1}(n)$ e o sinal necessário ao DPCM do bloco seguinte ($\hat{s}(n)$), correspondente à média reconstruída do bloco.

## 3.3 Implementação do DPCM

A Figura 3.3 mostra o diagrama de blocos do sistema de compressão DPCM.

Como visto na Seção 2.2, precisamos calcular a diferença entre a média de $y(n)$ ($s(n)$) e a média reconstruída do bloco anterior, $\hat{s}(n-1)$. O resultado desta operação é $e(n)$.

Em seguida, $e(n)$ é quantizado pelo circuito descrito na Seção 4.8. A partir daí, o circuito de módulo DPCM (Seção 4.5) remove o sinal de $e(n)$ e transmite-o diretamente como $b_{1,1}(n)$.

O módulo de $e(n)$, $|e(n)|$, é então quantizado a partir do uso de um conversor do tipo *flash* usando circuitos comparadores (Seção 4.6.1). Estes circuitos são usados para comparar $|e(n)|$ com o vetor de correntes de limiar $\mathbf{t_0}$ e formar o código termômetro digital, $\mathbf{o}(n)$.

Como $\mathbf{o}(n)$ possui 7 bits e pode ser comprimido de forma simples (Seção 2.1),

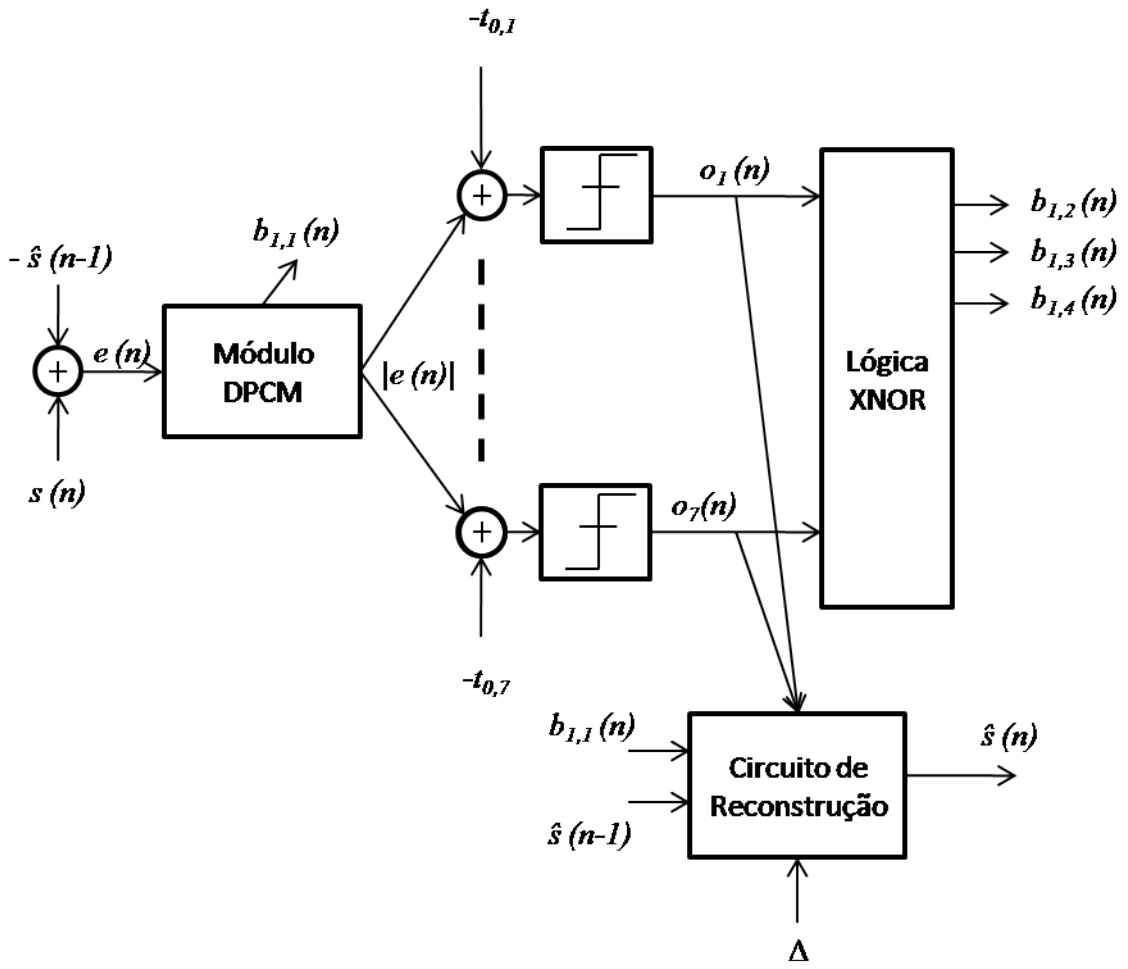



Figura 3.3: Diagrama em blocos da compressão DPCM.

usamos o decodificador lógico (Seção 4.7) para converter $\mathbf{o}(n)$ nos sinais $b_{1,2}(n)$, $b_{1,3}(n)$ e $b_{1,4}(n)$. Estes bits em conjunto com $b_{1,1}(n)$ gerado pelo circuito de módulo formam o sinal de saída $\mathbf{b_1}(n)$.

### 3.3.1 Reconstrução da Média

A partir de $\mathbf{o}(n)$, $Y_0$, $b_{1,1}(n)$ e $\hat{s}(n-1)$ é possível reconstruir $s(n)$ com o circuito de reconstrução (Seção 4.8.1). O sinal reconstruído será representado como $\hat{s}(n)$.

O sinal $\hat{s}(n)$ é usado pelo circuito do bloco seguinte, $n+1$, para reconstruir seu próprio sinal, $\hat{s}(n+1)$. A Figura 3.4 mostra como o $\hat{s}(n)$ é gerado.

Como mostrado na Figura 3.4, o circuito de reconstrução calcula $\hat{s}(n)$ da seguinte maneira:

Se $b_{1,1}(n) = 1$, ou seja, $e(n) > 0$:

$$\hat{s}(n) = \hat{s}(n-1) + |\hat{e}(n)|$$

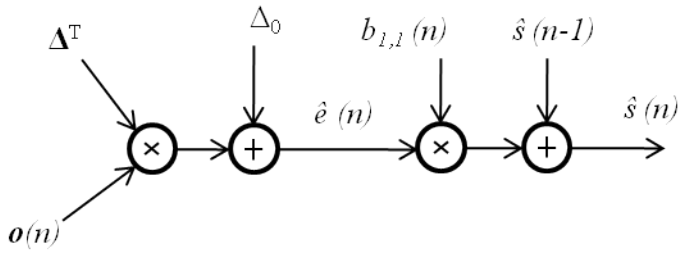



Figura 3.4: Diagrama de reconstrução do sinal $\hat{s(n)}$. O símbolo $\times$ representa um produto interno.

ou se $b_{1,1}(n) = 1$, ou seja, $e(n) < 0$:

$$\hat{s}(n) = \hat{s}(n-1) - |\hat{e}(n)|$$

onde

$$\hat{e}(n) = \mathbf{o}(n)^{\mathbf{T}}.\Delta + \Delta_0$$

onde $\Delta$ é a diferença entre os níveis de reconstrução $Y_0$:

$$\Delta = [Y_{0,2} - Y_{0,1} \quad Y_{0,3} - Y_{0,2} \quad ... \quad Y_{0,8} - Y_{0,7}]^T \text{ e } \Delta_0 = Y_{0,1}$$

Os vetores $\Delta$ e $\mathbf{t_0}$ serão gerados externamente ao bloco e são os mesmos para todo o circuito integrado. Isto permite que estes sinais sejam gerados com precisão elevada, pois basta gerá-los uma vez, copiando-os através de espelhos de corrente para cada bloco.

Espacialmente, o n-ésimo bloco recebe $\hat{s}(n-1)$ do bloco a sua esquerda e entrega $\hat{s}(n)$ ao bloco a sua direita. Em especial, consideramos para o primeiro bloco de cada coluna $\hat{s}(n-1) = 0$. O último bloco de cada coluna não possui o circuito de reconstrução pois não há nenhum bloco à sua direita.

## 3.4 Implementação do VQ

A Figura 3.5 mostra o diagrama de blocos do sistema de compressão do PCA e do VQ. O vetor de entrada $\mathbf{y}(n)$ passa pelo circuito de transformação linear (produtos internos) gerando o vetor de componentes principais $\mathbf{p}(n)$, como mostrado na Figura 3.6. O circuito que realiza esta função está descrito na Seção 4.4.

O resultado desta operação é:

$$\mathbf{p}(n) = \mathbf{DHy}(n)$$

A implementação dos multiplicadores da matriz $\mathbf{D}$, que são usados para normali-

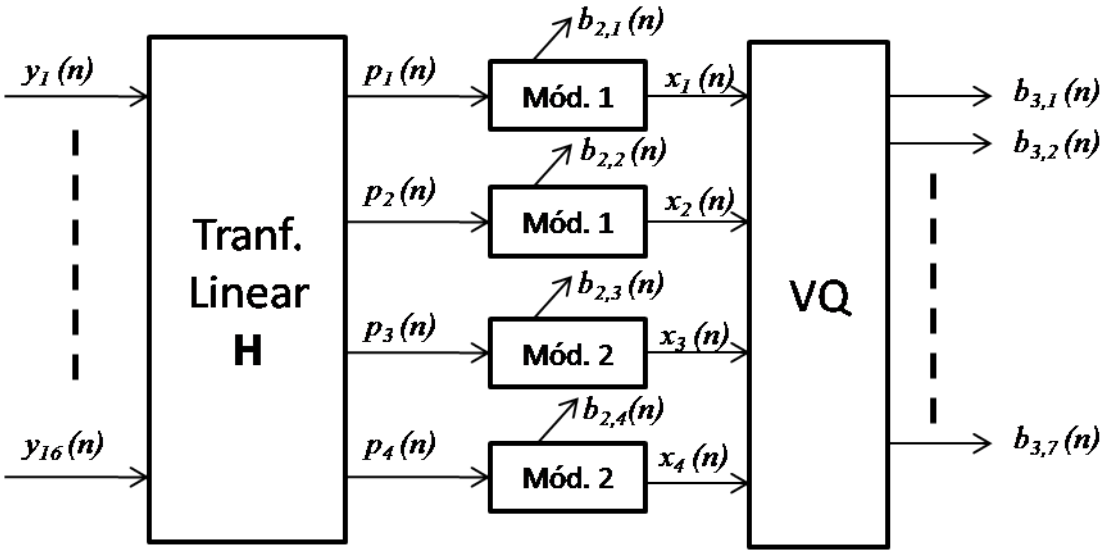



Figura 3.5: Diagrama do PCA e do VQ.

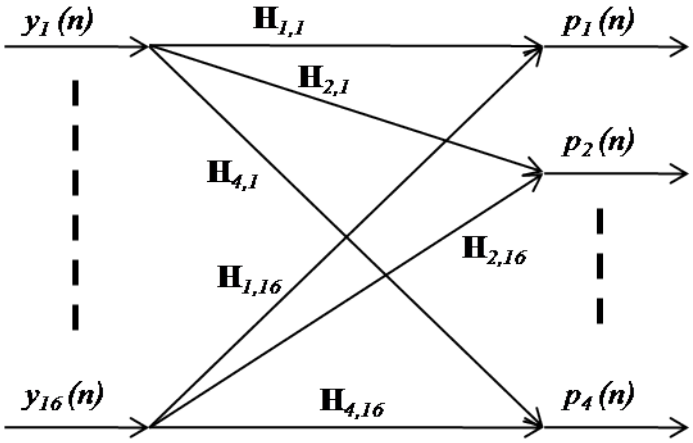



Figura 3.6: Diagrama da transformação linear $\mathbf{H}$.

zar os valores de $\mathbf{p}(n)$ (Seção 2.3) é feita nos circuitos de Módulo 1 (Seção 4.5). Estes circuitos, diferente dos circuitos de Módulo 2, dividem a corrente pela metade na saída. O resultado é que os sinais $p_1(n)$ e $p_2(n)$ também são reduzidos pela metade.

Os circuitos de módulo calculam os sinais de cada componente do vetor $\mathbf{p}(n)$, gerando a saída $\mathbf{b_2}(n)$. Esta saída é transmitida diretamente. O vetor $\mathbf{x}(n)$, calculado pelos circuitos de módulo, corresponde a:

$$\mathbf{x}(n) = |\mathbf{p}(n)|$$

$\mathbf{x}(n)$ é, então, utilizado pelo sistema de quantização vetorial como mostrado na Figura 3.7.

O sinal $\mathbf{x}(n)$ passa pela transformação linear $\mathbf{U}$, formando o vetor $\mathbf{f}(n)$:

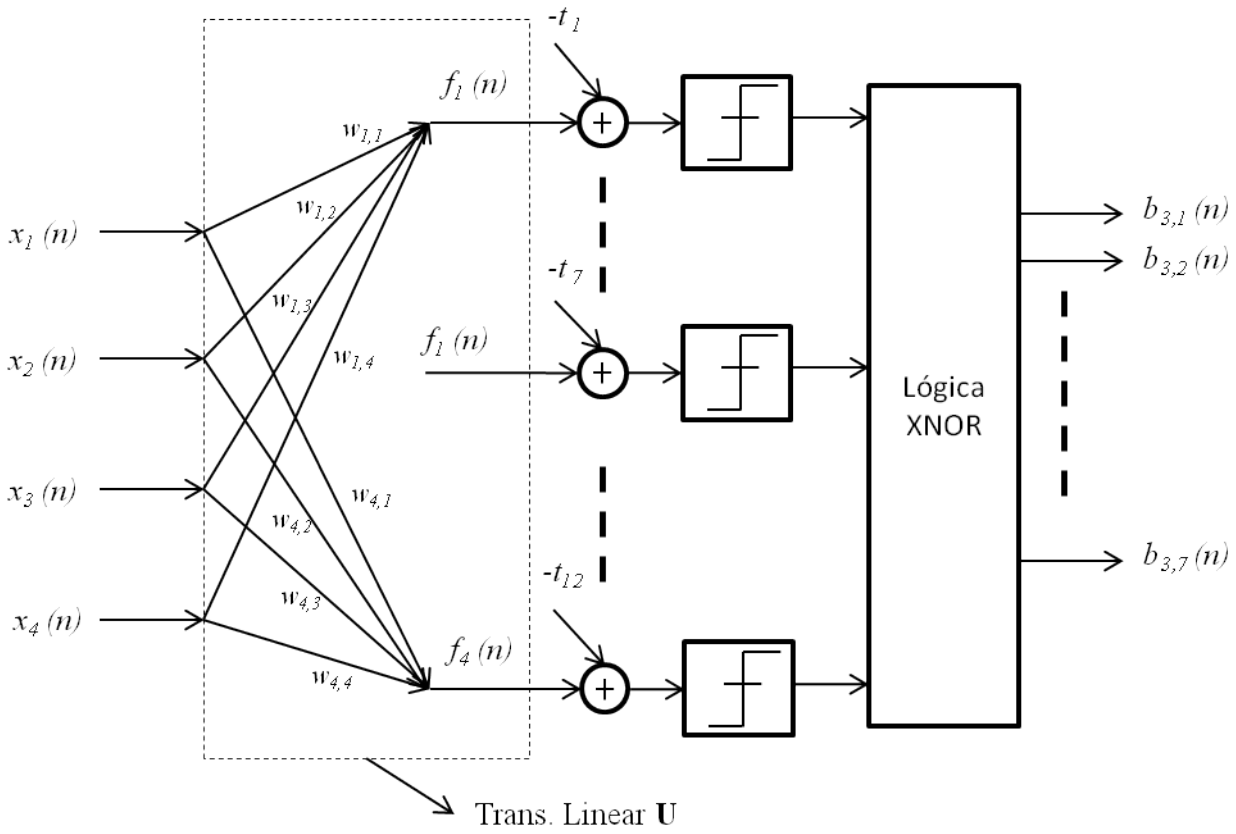



Figura 3.7: Diagrama em detalhes do VQ.

$$\mathbf{f}(n) = \mathbf{U}\mathbf{x}(n)$$

A última etapa do sistema de compressão é a quantização de $\mathbf{f}(n)$. Isto é feito por um circuito conversor do tipo *flash* (Seção 2.1). Conforme observado anteriormente, o resultado é um sinal do tipo termômetro que precisa ser convertido para uma forma mais compacta pelo decodificador lógico (Seção 4.7). O resultado desta operação é a saída $\mathbf{b_3}(n)$.

# Capítulo 4

# Circuitos Utilizados

Após a apresentação dos diagramas gerais no capítulo anterior, neste serão mostrados e explicados os esquemáticos dos circuitos utilizados para realizar as operações dos blocos 4×4.

Para representar as variáveis contidas no Capítulo 3 eletricamente serão usadas notações específicas, seguindo o exemplo a seguir:
Para representar $y_i(n)$:

- Se for uma corrente elétrica, esta será representada por $I_{y,i}(n)$, que é sempre diretamente proporcional a $y_i(n)$;
- Se for uma tensão de *gate* para cópia de $I_{y,i}(n)$, esta será representada por $V_{y,i}(n)$.

Para representar variáveis binárias são utilizadas as mesmas representações do Capítulo 3, $b_{1,1}(n)$ por exemplo.

Todos os circuitos que apresentaremos foram projetados com a ajuda de um *software* simulador de circuitos (CADENCE® Spectre®) que levou em conta parâmetros reais extraídos da tecnologia CMOS.

Estas simulações utilizaram os modelos de transistores fornecidos pela AMS (*austriamicrosystems*) para a tecnologia CMOS $0.35\mu m$ que usamos em nosso primeiro protótipo. Esta tecnologia oferece tensões de alimentação de 3,3V ou 5V. Usaremos 3,3V em todos os circuitos pois, como o processamento é em modo de corrente, não é necessário utilizar uma fonte maior de alimentação que acarretaria em maior consumo e maior área ocupada.

As simulações foram feitas de duas formas: utilizando parâmetros típicos e levando em conta variações estatísticas (simulações de Monte Carlo) dos parâmetros. Para cada uma das simulações de Monte Carlo foram executadas 20 rodadas e as variações ocorreram para parâmetros de processo e/ou descasamento entre os transistores. Diferentes topologias de circuito podem ser mais ou menos sensíveis a um ou aos dois tipos de variações apresentadas.

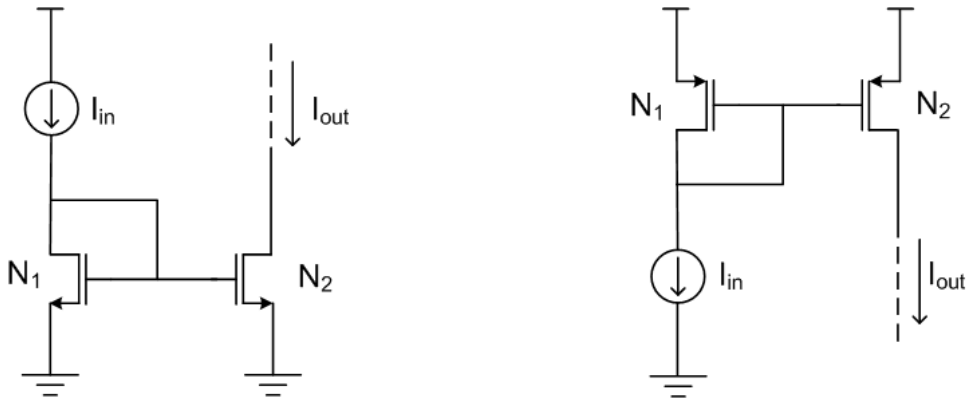
As variações do processo de fabricação são aquelas que afetam todos os transistores do circuito igualmente, isto é, os parâmetros são iguais para transistores de mesmas dimensões e tipo (PMOS ou NMOS).

As variações de descasamento levam em conta diferenças entre cada transistor do circuito, ou seja, transistores de mesmo tipo e dimensões possuirão parâmetros diferentes.

## 4.1 Espelhos de Corrente

O sistema de compressão, por ser uma implementação de uma rede neural, possui um grande número operações de soma e escalamento (multiplicação por um número fixo). Neste caso, circuitos em modo de corrente são mais indicados para sintetizar o sistema, pois essas operações podem ser realizadas de forma mais simples e com o uso de um menor número de transistores.

O circuito básico utilizado é o espelho de corrente simples (Figura 4.1) que consiste de $N_1$ transistores unitários de entrada e $N_2$ transistores unitários de saída. Para obter melhor precisão é necessário que estes transistores unitários tenham dimensões ($W$ e $L$) iguais: podemos aproveitar esta simetria para obter melhores resultados.




(a) (b)

Figura 4.1: Diagrama esquemático de espelhos de corrente simples: (a) NMOS; (b) PMOS.

### Funcionamento

Para transistores com os mesmos parâmetros a corrente de saída, idealmente, é dada pela equação:

$$I_{out} = \frac{N_2}{N_1}.I_{in}$$

Assim, o número de transistores necessários para cada multiplicação (M) é:

$$M = N_2 + N_1$$

Os transistores devem operar na região de saturação para que a diferença de potencial que controla a corrente de dreno (*drain*) $I_d$ seja a tensão entre a porta (*gate*) e a fonte (*source*) que será igual em todos os transistores. A equação que determina esta corrente é:

$$I_d = \frac{K}{2}.\frac{W}{L}.(V_{gs} - V_{th})^2.(1 + \lambda.V_{ds})$$

onde $K$ é a constante de transcondutância, $W$ a largura do transistor, $L$ o comprimento, $V_{th}$ a tensão de limiar (*threshold*) e $\lambda$ o fator de modulação de comprimento do canal. $V_{th}$ é positivo para transistores de canal N (NMOS) e negativo para transistores de canal P (PMOS).

Note que um fator $\lambda$ diferente de 0 faz com que a corrente de saída não seja exatamente proporcional à corrente de entrada, pois a operação não garante que as tensões entre dreno e fonte ($V_{ds}$) sejam iguais. Devemos minimizar a influência do parâmetro $\lambda$ para aumentar a precisão do circuito. Para reduzir este efeito, também chamado de *efeito de modulação de comprimento do canal* [22], podemos aumentar o comprimento do canal dos transistores ($L$). Este aumento fará com que os transistores ocupem uma área maior e este compromisso deve ser levado em conta.

Além deste efeito, outros parâmetros podem variar e diminuir a precisão do circuito. O parâmetro $V_{th}$ (tensão de *threshold*) e os parâmetros que compõem $K$ são ligeiramente diferentes entre transistores (variações de descasamento). Transistores maiores geralmente reduzem estes efeitos. As dimensões dos transistores unitários que fazem parte de espelhos de corrente, para uso nos circuitos deste trabalho, foram estudadas em [20] e serão usadas nos transistores unitários. As dimensões encontradas foram: $W = 1$ $\mu$m e $L = 2$ $\mu$m, os resultados da simulação de Monte Carlo para algumas dimensões de transistores estão na Tabela 4.1.

Tabela 4.1: Desvio-padrão dos espelhos de corrente para $I_{in} = 10$ $\mu$A.

| W ($\mu$m) | L ($\mu$m) | $\sigma(\mu A)$ |
|---|---|---|
| 1 | 0,35 | 1,32 |
| 1 | 1 | 0,43 |
| 1 | 2 | 0,22 |
| 2 | 2 | 0,23 |

### 4.1.1 Operações Básicas

O fator de escalamento (multiplicação) pode ser implementado variando o número de transistores de entrada e saída enquanto a soma das correntes pode ser realizada conectando as saídas dos espelhos em um mesmo nó. Vale notar que a multiplicação só pode ser feita com razões de números inteiros, desta forma quanto mais precisão utilizarmos para o escalamento, mais transistores serão necessários. Como desvantagem, este método não possui flexibilidade nos valores das multiplicações sem aumentar muito o número de transistores, ou seja, os pesos (elementos de **U** e **H**) serão fixos e definidos durante o projeto do circuito integrado. Os valores dos elementos de **U** e **H** estão no Capítulo 2.

Uma vantagem de se utilizar circuitos em modo de corrente é em relação às variações dos parâmetros de processo de fabricação: os espelhos de corrente praticamente só sofrem influência de descasamento entre transistores, e isto pode ser minimizado aplicando corretamente as técnicas de *layout* [23]. Por exemplo, minimizar distâncias entre os transistores que compõem um mesmo espelho de corrente, alinhar seus *gates* e intercalar os transistores unitários de entrada com os de saída são algumas técnicas usadas.

Para a variação de parâmetros globais do circuito, os espelhos de corrente são relativamente menos sensíveis, pois se os transistores de um mesmo espelho de corrente sofrem a mesma variação, o resultado será praticamente o mesmo.

## 4.2 Circuito de Leitura

Como estaremos utilizando circuitos em modo de corrente em toda a estrutura de compressão, foi escolhida uma topologia de circuito de leitura com saída em corrente [24] para evitar transformações tensão-corrente e corrente-tensão que exigiriam ainda mais transistores, consumiriam mais energia e introduziriam erros no processo.

Um fotodiodo é usado como elemento fotosensível, e a arquitetura implementada é muito próxima à das células 3T [25], com a diferença de utilizarmos um transistor PMOS como transcondutor ao invés de um transistor NMOS como amplificador seguidor de fonte, mais comumente utilizado. A Figura 4.2 mostra o esquemático do circuito.

### Funcionamento

Analogamente às células 3T, a tensão ($V_{pd}$) armazenada na soma da capacitância do fotodiodo ($C_{pd}$) com a capacitância entre *gate* e *source* ($C_{gs}$) do transistor $\mathbf{M_t}$ diminui conforme a corrente do fotodiodo (fotocorrente $I_{ph}$). O transistor $\mathbf{M_R}$,

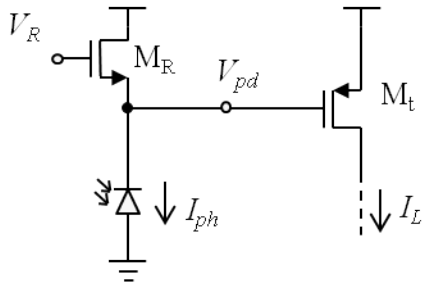



Figura 4.2: Esquemático do circuito de leitura.

através do sinal de $V_R$ leva a tensão $V_{pd}$ para seu estado inicial até $t_R$, quando cortamos o transistor $\mathbf{M_R}$ para iniciar a integração da corrente $I_{ph}$.

Sendo a fotocorrente proporcional à potência luminosa, a velocidade do decréscimo da tensão $V_{pd}$ é proporcional à incidência (potência) luminosa sobre determinado pixel, conforme mostrado na Figura 4.3. A amostragem do sinal é feita após o tempo de integração ($t_{int}$) pelo circuito de CDS (Seção 4.3).

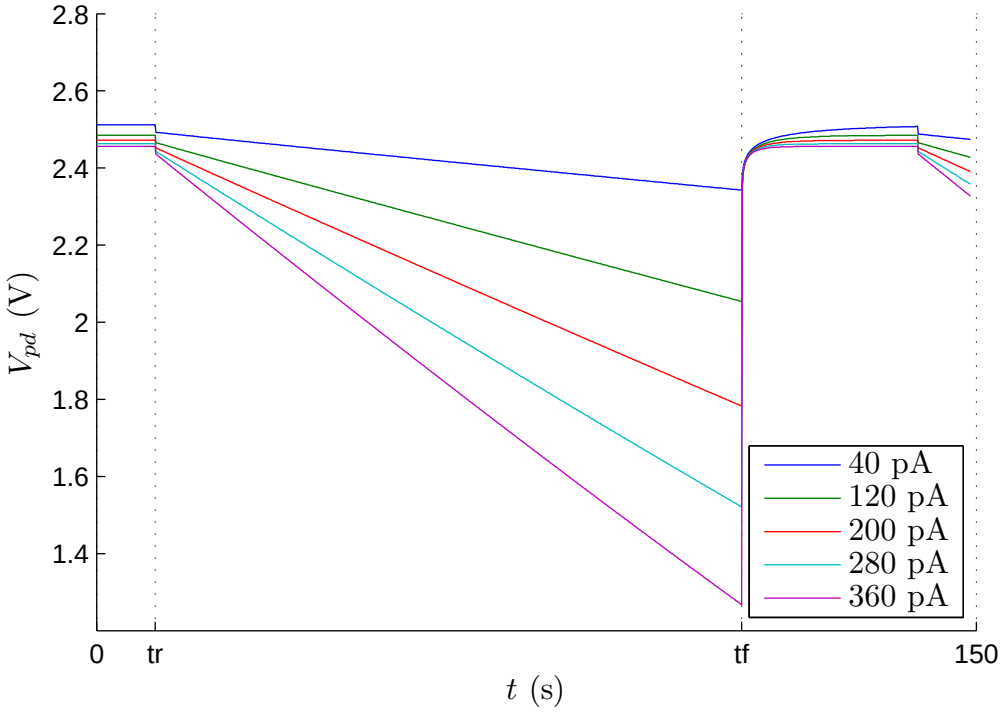



Figura 4.3: Simulação da tensão $V_{pd}$ para alguns valores de $I_{ph}$.

A corrente de saída é a corrente de dreno do transistor $\mathbf{M_t}$ ($I_L$). Esta corrente é inversamente proporcional à tensão $V_{pd}$, e portanto, diretamente proporcional à potência luminosa incidente. O transistor $\mathbf{M_t}$ deve ser mantido na região de triodo para que a corrente de saída seja linear com potência luminosa incidente [24]. Para conhecimento, a equação que define a corrente no transistor na região de triodo é:

$$I_d = \frac{W}{L}((V_{gs} - V_{th})V_{ds} - \frac{V_{ds}^2}{2})$$

onde, para o transistor $\mathbf{M_t}$, $V_{gs} = 3,3$ V - $V_{pd}$.

Assumimos que o fotodiodo terá dimensões 10 $\mu$m $\times$ 10 $\mu$m e a corrente máxima no fotodiodo será de 400 pA (para uma potência luminosa de aproximadamente 12 W/m$^2$), com isso, projetamos o circuito de leitura, com a ajuda do simulador, para atingir os seguintes objetivos:

- Minimizar área ocupada;
- Possuir uma faixa de sinal de saída de aproximadamente 10 $\mu$A;
- Ser tão linear quanto possível.

A Tabela 4.2 mostra as dimensões dos transistores usados neste circuito.

## Simulações

A Figura 4.4 mostra a simulação do circuito (com parâmetros típicos), ao longo de um tempo de integração de 100 $\mu$s , para $I_{ph} = 400$ pA. Como pode ser observado a faixa de corrente de saída é de aproximadamente 10 $\mu$A.

Tabela 4.2: Dimensões dos transistores do circuito de leitura.

| Transistor | W ($\mu$m) | L ($\mu$m) |
|---|---|---|
| $\mathbf{M_R}$ | 1 | 2 |
| $\mathbf{M_t}$ | 1 | 3,7 |

A Figura 4.5 mostra os valores de $I_{Mt}$ em $t_F$, ou seja, no final do período de integração, em relação a $I_{ph}$. A corrente $I_{Mt}$ em $t_F$ será retida pelo circuito de CDS (Seção 4.3).

Note que a resposta do circuito não é linear em relação a $I_{ph}$. Isto ocorre pois $V_{ds}$ não é constante. Em região de triodo, a função de transcondutância dos transistóres CMOS é:

$$I_d = K.\frac{W}{L}.(V_{gs} - V_{th}).V_{ds} - \frac{V_{ds}^2}{2}$$

A Figura 4.6 mostra a simulação de Monte Carlo do circuito de leitura, considerando variações dos parâmetros de processo e de descasamento.

É possível observar que, ao contrário dos circuitos que utilizam espelho de corrente, este circuito é mais sensível às variações de parâmetros do processo de fabricação. Isto ocorre pois as variações no transistor $\mathbf{M_t}$ (principalmente sua transcondutância) estão diretamente relacionadas com o sinal de saída do circuito. Como

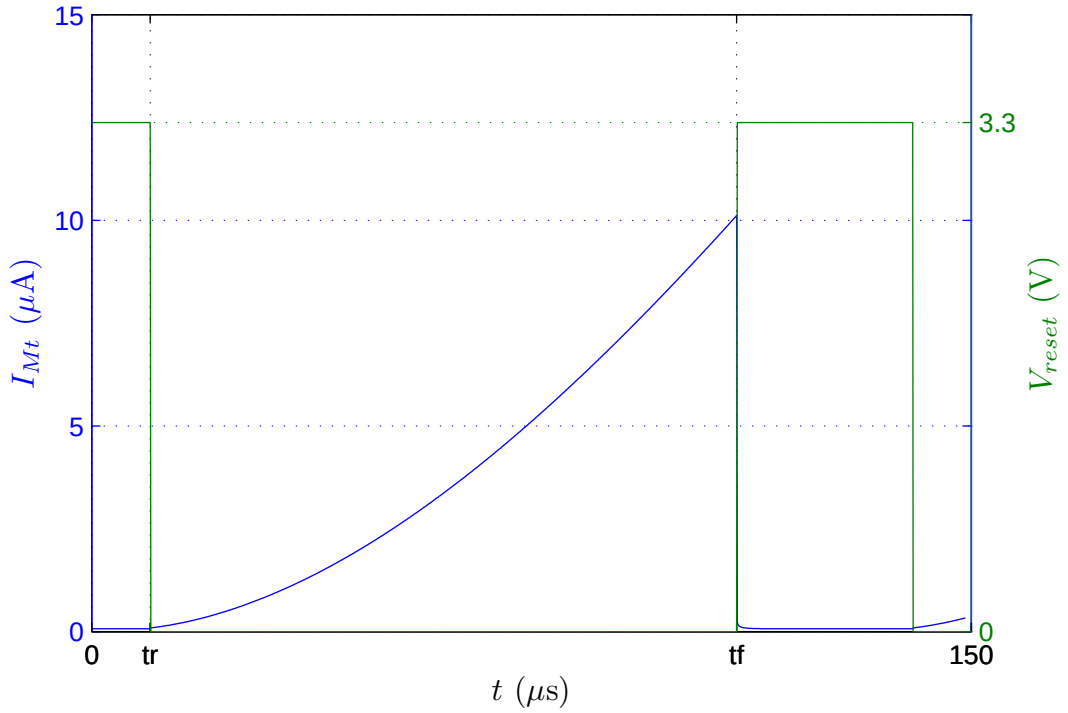



Figura 4.4: Simulação temporal do circuito de leitura.

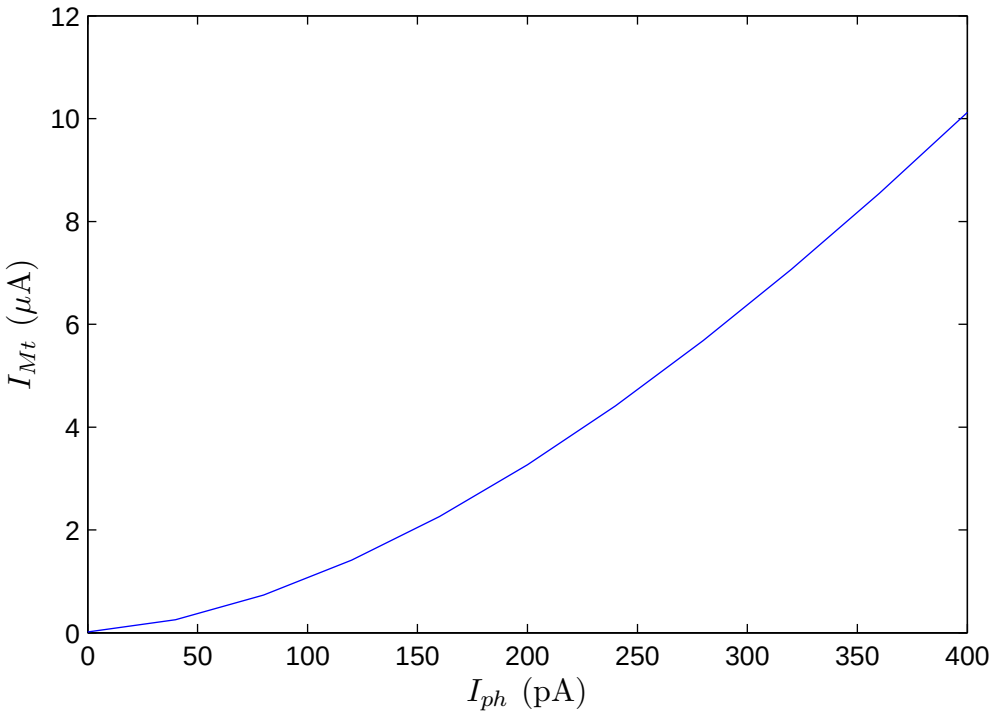
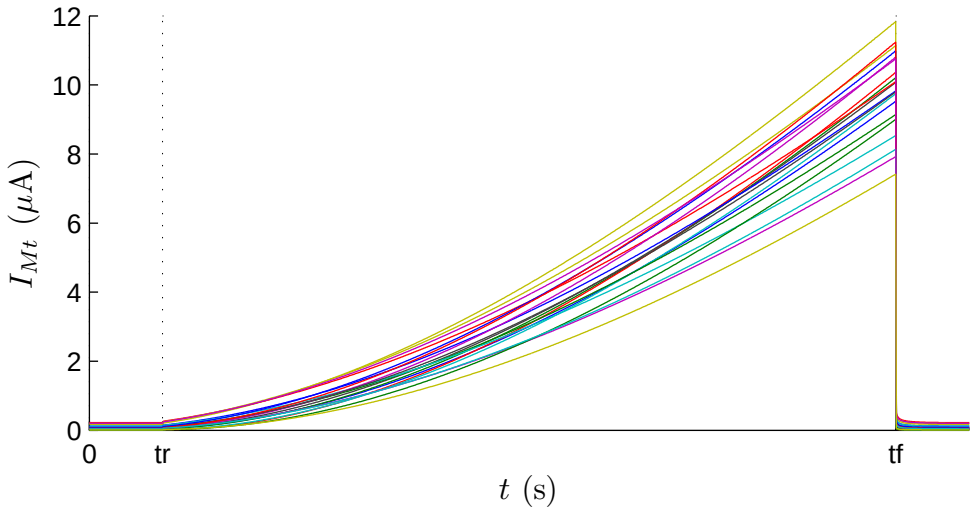
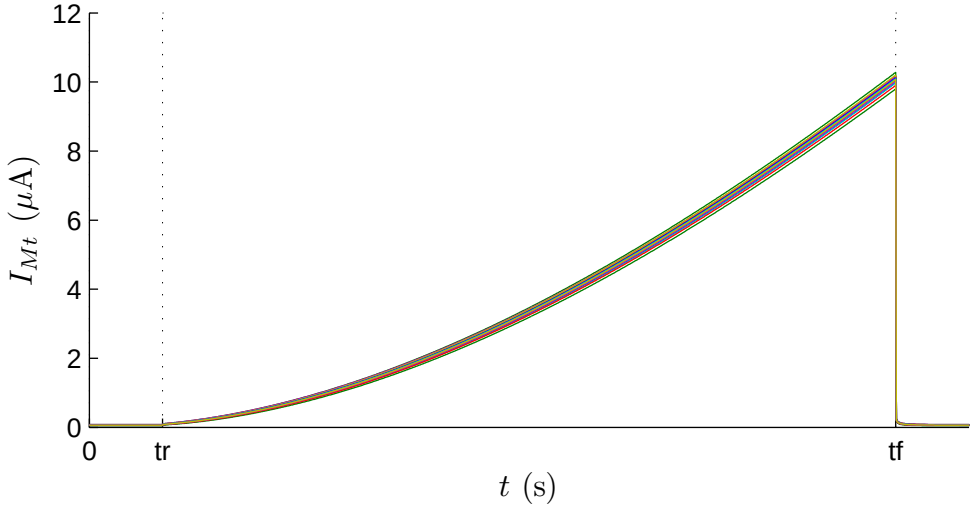



Figura 4.5: Simulação do circuito de leitura $I_{Mt} \times I_{ph}$ para $t_{int} = 100$ $\mu$s.

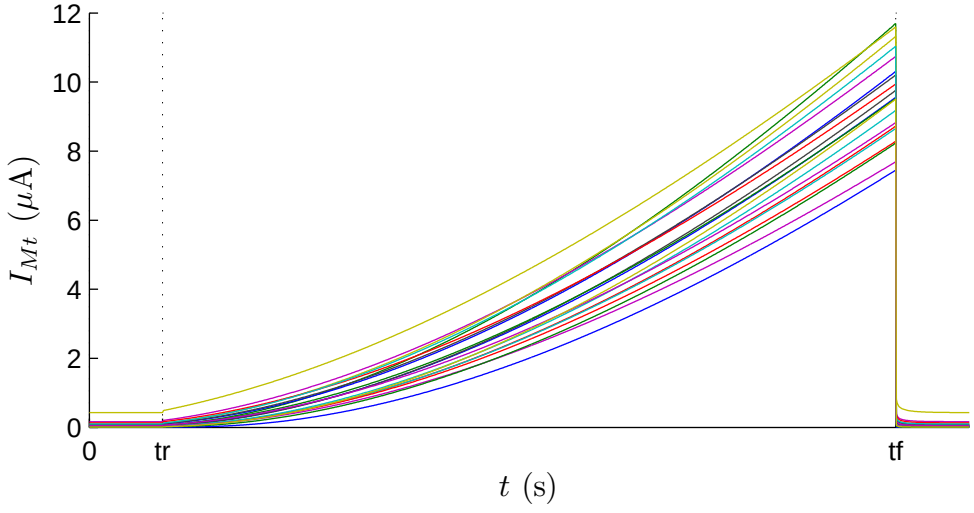
esta variação ocorre para todos os pixels do sistema, é possível corrigir esta variação alterando, por exemplo, o tempo de integração, calibrando-o de acordo com a luminosidade incidente e a resposta de cada chip fabricado.




(a)




(b)




(c)

Figura 4.6: Simulação de Monte Carlo do circuito de leitura para $t_{int} = 100$ $\mu$s: (a) apenas variações de processo; (b) apenas variações de descasamento; (c) ambas variações.

## 4.3 Correlated Double Sampling

A técnica de CDS (*Correlated Double Sampling*) é comumente utilizada para corrigir erros de FPN (*Fixed Pattern Noise*) [3]. Utilizaremos este circuito também para reduzir eventuais diferenças causadas pelos parâmetros dos transistores $\mathbf{M_R}$ e $\mathbf{M_t}$ e dos fotodiodos.

O circuito projetado neste trabalho funciona retendo $I_L$ ao final do tempo de integração (chamaremos esta corrente de $I_F$) e $I_L$ no começo da integração (chamaremos esta corrente de $I_R$). Eventuais diferenças absolutas na corrente de saída podem ser canceladas se a diferença entre estes dois sinais for calculada. A Figura 4.7 contém o esquemático do circuito de CDS.

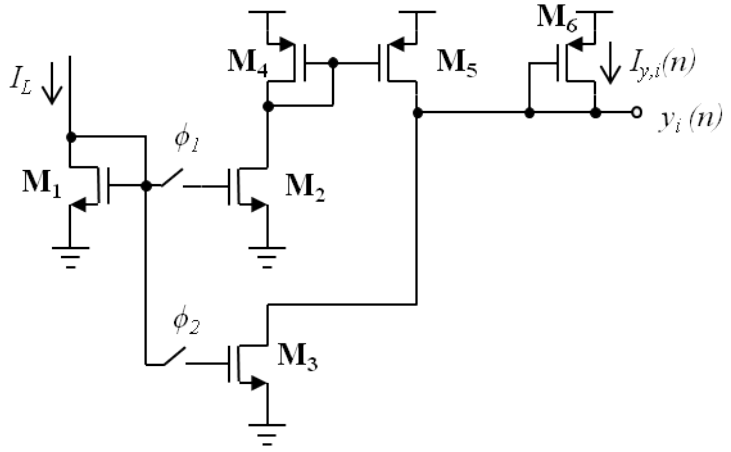



Figura 4.7: Esquemático do circuito que realiza o CDS.

Para compensar variações relativas, isto é, variações de proporção da saída com a potência luminosa, podemos modificar o tempo de integração, ajustando-o para otimizar a faixa dinâmica de leitura.

### Funcionamento

As chaves analógicas $\phi_1$ e $\phi_2$ permanecem desligadas enquanto o sinal $V_R$ permanecer em nível alto (3,3 V). Assim que $V_R$ assumir nível baixo (0 V), a chave $\phi_2$ é ligada, retendo a corrente $I_R$ na capacitância parasita do transistor $\mathbf{M_3}$. Ao fim do tempo de integração, a chave $\phi_1$ é ligada, retendo a corrente $I_F$ na capacitância parasita do transistor $\mathbf{M_2}$. Através do inversor de corrente formado pelo espelho $\mathbf{M_4}$ e $\mathbf{M_5}$, o sinal de saída do CDS, $I_{y,i}(n)$ se torna:

$$I_{y,i}(n) = I_F - I_R$$

E esta corrente corresponde ao sinal $y_i(n)$, ou seja, o sinal de luminância do $i$-ésimo pixel do $n$-ésimo bloco.

As dimensões dos transistores deste circuito estão na Tabela 4.3. O transistor $\mathbf{M_6}$ tem W = 4 $\mu$m pois o sinal deve ser dividido por 4 para compensar a norma das linhas da matriz **H**. Adicionalmente, as componentes $p_1(n)$ e $p_2(n)$ são divididas por 2 para compensar a maior norma que possuem em relação às componentes $p_3(n)$ e $p_4(n)$. Este ajuste é feito nos circuitos de módulo (Seção 4.5).

Tabela 4.3: Dimensões dos transistores do circuito de leitura.

| Transistor | W ($\mu$m) | L ($\mu$m) |
|---|---|---|
| $\mathbf{M_1}$ | 1 | 2 |
| $\mathbf{M_2}$ | 2 | 2 |
| $\mathbf{M_3}$ | 2 | 2 |
| $\mathbf{M_4}$ | 1 | 2 |
| $\mathbf{M_5}$ | 1 | 2 |
| $\mathbf{M_6}$ | 4 | 2 |

### 4.3.1 Chave Analógica

A Figura 4.8 mostra o esquemático das chaves analógicas.

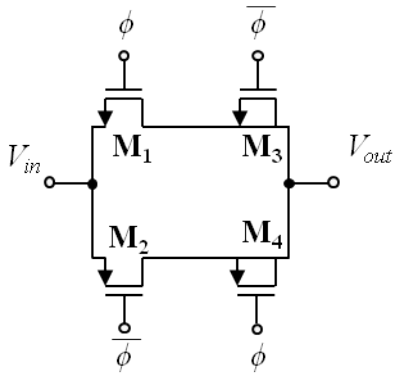



Figura 4.8: Esquemático das chaves analógicas.

Quando a chave está desligada, isto é, com $\phi = 3,3$ V e $\overline{\phi} = 0$ V, $V_{ret} = V_{in}$. Quando abrimos a chave, isto é, fazendo $\phi = 0$ e $\overline{\phi} = 3,3$ V, $V_{ret}$ retem a tensão $V_{in}$ armazenada quando a chave estava desligada.

Os transistores $\mathbf{M}_3$ e $\mathbf{M}_4$ funcionam como transistores *dummy* e servem para reduzir efeitos de injeção de carga [22]. Estes transistores só foram utilizados na saída da chave pois na entrada não há armazenamento de carga, suprimindo a necessidade dos transistores *dummy*.

As dimensões dos transistores das chaves analógicas estão na Tabela 4.4.

Tabela 4.4: Dimensões dos transistores da chave analógica.

| Transistor | W ($\mu$m) | L ($\mu$m) |
|---|---|---|
| $\mathbf{M_1}$ | 1 | 0,35 |
| $\mathbf{M_2}$ | 1 | 0,35 |
| $\mathbf{M_3}$ | 0,5 | 0,35 |
| $\mathbf{M_4}$ | 0,5 | 0,35 |

## 4.4 Transformação Linear

O circuito de transformação linear implementa o produto dos sinais dos 16 pixels de cada bloco ($y_1(n)$ a $y_{16}(n)$) com a matriz **H** (Seção 2.4) e o produto do vetor $\mathbf{x}(n)$ com a matriz **U**.

Para realizar as multiplicações necessárias das correntes de entrada dos produtos internos, utilizamos espelhos de corrente, onde a relação de número de transistores unitários entre a entrada e saída dos espelhos define o escalamento para cada elemento do sinal de entrada, e o tipo de transistor define o sinal desta multiplicação (NMOS para sinais positivos e PMOS para sinais negativos). A soma das correntes resultantes das multiplicações é feita conectando em um mesmo nó a saída dos transistores cujas correntes serão somadas (Seção 4.1.1).

Como a saída do circuito de CDS (Seção 4.3) é uma entrada de um espelho de corrente PMOS (de dimensões $W = 1$ $\mu$m e $L = 2$ $\mu$m), para realizarmos uma multiplicação com sinal positivo precisamos também de um transistor NMOS como mostra a Figura 4.9. A exceção é para a coluna 11 da matriz **H** que possui apenas elementos negativos. Neste caso não é necessário nenhum circuito para a interface.

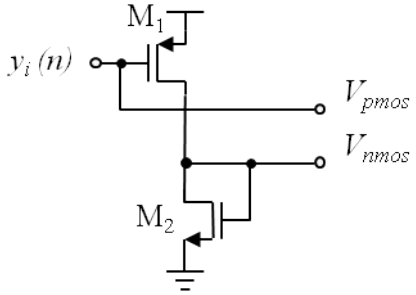



Figura 4.9: Circuito de entrada para implementação de $\mathbf{Hy}(n)$.

Com os sinais $V_{pmos}$ e $V_{nmos}$ disponíveis basta conectar os *gates* dos transistores de saída conforme a Tabela 4.5.

Para a implementação do produto do vetor $\mathbf{x}(n)$ com a matriz **U** basta conectar os transistores de acordo com a Tabela 4.6.

A Figura 4.10 mostra um exemplo de esquemático do circuito utilizado para um produto interno, neste caso para a linha 3 da matriz **U** (0 -0,5 0,5 1).

Tabela 4.5: Dimensões dos transistores para implementação da matriz **H**.

| Elemento da matriz **H** | Tipo de transistor | W ($\mu$m) | L ($\mu$m) |
|---|---|---|---|
| 1 | NMOS | 1 | 2 |
| 2 | NMOS | 2 | 2 |
| -1 | PMOS | 1 | 2 |
| -2 | PMOS | 2 | 2 |

Tabela 4.6: Dimensões dos transistores para implementação da matriz **U**.

| Elemento da matriz **U** | Tipo de transistor | W ($\mu$m) | L ($\mu$m) |
|---|---|---|---|
| 0 | - | - | - |
| 0,5 | NMOS | 1 | 2 |
| 1 | NMOS | 2 | 2 |
| -0,5 | PMOS | 1 | 2 |

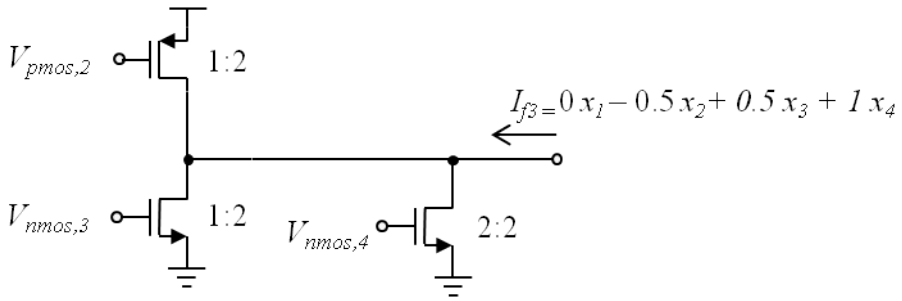



Figura 4.10: Exemplo de circuito de produto interno.

## 4.5 Circuito de Módulo

O circuito de módulo, utilizado neste trabalho, foi retirado de [26] e possui duas funções principais: permitir a cópia do módulo da corrente de entrada e determinar o sinal desta através de uma saída digital conforme descrito na Tabela 4.7. A Figura 4.11 contém o esquemático deste circuito.

Tabela 4.7: Função do circuito de módulo.

| Entrada | Saída | |
|---|---|---|
| $I_{in}$ | $V_{sinal}$(V) | $I_{M7}$ |
| $> 0$ | 3,3 | $I_{in}$ |
| $< 0$ | 0 | $-I_{in}$ |

### Funcionamento

No circuito de módulo proposto, teremos 2 situações:

Se $I_{in}$ é positiva (no sentido da seta na figura), o nó de entrada assume um valor

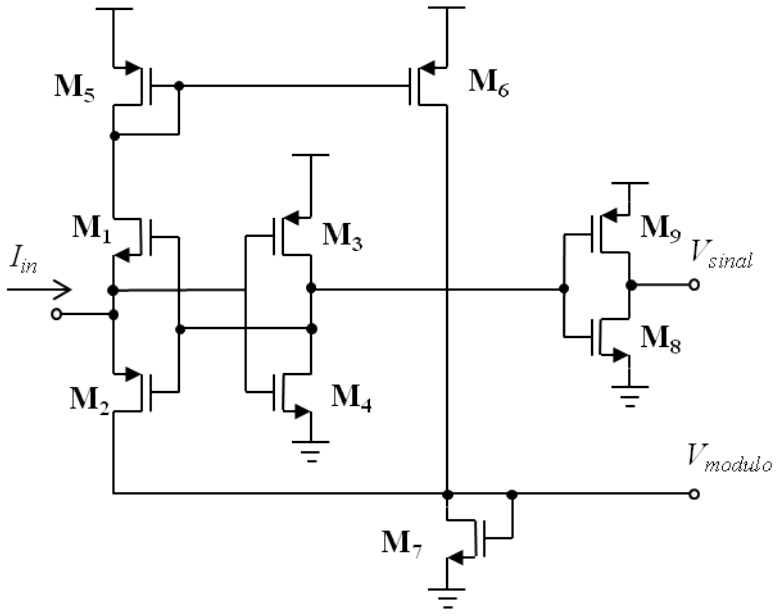



Figura 4.11: Esquemático do circuito de módulo.

próximo a 3, 3 V. Isto faz com que o inversor composto por $\mathbf{M_3}$ e $\mathbf{M_4}$ leve o transistor $\mathbf{M_1}$ para a região de corte, onde ele não conduzirá corrente alguma, fazendo $I_{in}$ passar exclusivamente por $\mathbf{M_2}$ e $\mathbf{M_7}$. A partir tensão da porta de $\mathbf{M_7}$ ($V_{modulo}$) é possível copiar múltiplos da corrente $I_{in}$. Como a saída do inversor composto por $\mathbf{M_3}$ e $\mathbf{M_4}$ é 0 V, a saída de tensão, resultante do outro inversor composto por $\mathbf{M_8}$ e $\mathbf{M_9}$, que indica o sentido da corrente $I_{in}$ ($V_{sinal}$) é 3,3 V.

Inversamente, se $I_{in}$ é negativa (no sentido contrário ao da seta na figura), o nó de entrada assume um valor próximo a 0 V. Isto faz com que o inversor composto por $\mathbf{M_3}$ e $\mathbf{M_4}$ leve o transistor $\mathbf{M_2}$ para a região de corte, onde ele não conduzirá corrente alguma, fazendo $I_{in}$ passar exclusivamente por $\mathbf{M_1}$ e $\mathbf{M_5}$ e ser copiada para $\mathbf{M_6}$ e $\mathbf{M_7}$. A partir tensão da base de $\mathbf{M_7}$ ($V_{modulo}$) é possível copiar múltiplos da corrente $I_{in}$. Como a saída do inversor composto por $\mathbf{M_3}$ e $\mathbf{M_4}$ é 3,3 V, a saída de tensão, resultante do outro inversor composto por $\mathbf{M_8}$ e $\mathbf{M_9}$, que indica o sentido da corrente $I_{in}$ ($V_{sinal}$) é 0 V.

Com isso, é possível notar que a corrente que passa por $\mathbf{M_7}$ é sempre igual ao módulo da corrente $I_{in}$, sendo ela negativa ou positiva, e a tensão $V_{sinal}$ indicará o sentido de $I_{in}$. Podemos copiar a corrente de $\mathbf{M_7}$ (ou múltiplos dela) para outros transistores NMOS, pois o transistor $\mathbf{M_7}$ funciona como transistor de entrada para um espelho de corrente, como mostrado na Seção 4.1.

As dimensões dos transistores, utilizados no DPCM, estão de acordo com a Tabela 4.8.

O circuito de módulo utilizado no VQ precisa ainda que $I_{in}$ seja copiada para transistores PMOS. Neste caso surge a necessidade de mais 2 transistores, como indicado na Figura 4.12.

Tabela 4.8: Dimensões dos transistores do circuito de módulo do DPCM.

| Transistor | W ($\mu$m) | L ($\mu$m) |
|---|---|---|
| $\mathbf{M_1}$ | 1 | 0,35 |
| $\mathbf{M_2}$ | 1 | 0,35 |
| $\mathbf{M_3}$ | 1 | 0,35 |
| $\mathbf{M_4}$ | 1 | 0,35 |
| $\mathbf{M_5}$ | 1 | 2 |
| $\mathbf{M_6}$ | 1 | 2 |
| $\mathbf{M_7}$ | 1 | 2 |
| $\mathbf{M_8}$ | 1 | 0,35 |
| $\mathbf{M_9}$ | 1 | 0,35 |

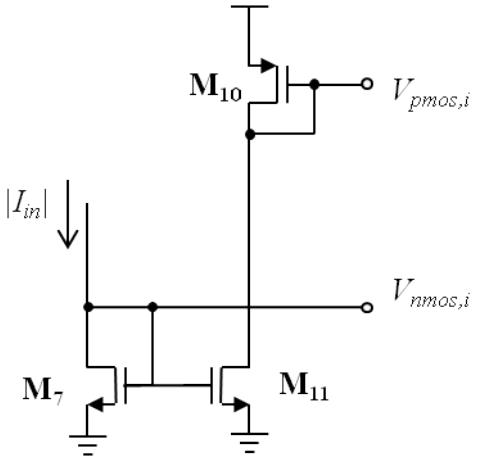



Figura 4.12: Saída do circuito de módulo com PMOS.

Com isto, a saída $V_{nmos,i}$ pode ser utilizada para copiar $I_{in}$, referente à $i$-ésima componente de $p(n)$, para um transistor NMOS e a saída $V_{pmos,i}$ pode ser utilizada para copiar $I_{in}$, referente à $i$-ésima componente de $p(n)$, para um transistor PMOS. Além disso há dois circuitos de módulo diferentes no VQ. Essa necessidade existe para implementar os multiplicadores da matriz D, conforme explicado na Seção 3.4. A diferença está em $\mathbf{M_7}$, que no "circuito de Módulo 1" tem o dobro de transistores unitários (resultando em um $W$ total duas vezes maior) que no "circuito de Módulo 2", fazendo com que a corrente copiada seja sempre a metade do que seria normalmente. A Tabela 4.9 contém as dimensões dos transistores utilizados no circuito.

## Simulações

A Figura 4.13 mostra a simulação do circuito considerando os parâmetros típicos. É possível notar que o circuito realiza as funções esperadas (calcular o módulo da corrente de entrada e o sinal desta corrente).

A Figura 4.14 mostra a simulação de Monte Carlo do circuito (variando

Tabela 4.9: Dimensões dos transistores do circuito de módulo do VQ.

| | Circuito de Módulo 1 | | Circuito de Módulo 2 | |
|---|---|---|---|---|
| Transistor | W ($\mu$m) | L ($\mu$m) | W ($\mu$m) | L ($\mu$m) |
| $\mathbf{M_1}$ | 1 | 0,35 | 1 | 0,35 |
| $\mathbf{M_2}$ | 1 | 0,35 | 1 | 0,35 |
| $\mathbf{M_3}$ | 1 | 0,35 | 1 | 0,35 |
| $\mathbf{M_4}$ | 1 | 0,35 | 1 | 0,35 |
| $\mathbf{M_5}$ | 1 | 2 | 1 | 2 |
| $\mathbf{M_6}$ | 1 | 2 | 1 | 2 |
| $\mathbf{M_7}$ | **4** | 2 | **2** | 2 |
| $\mathbf{M_8}$ | 1 | 0,35 | 1 | 0,35 |
| $\mathbf{M_9}$ | 1 | 0,35 | 1 | 0,35 |
| $\mathbf{M_{10}}$ | 2 | 2 | 2 | 2 |
| $\mathbf{M_{11}}$ | 2 | 2 | 2 | 2 |

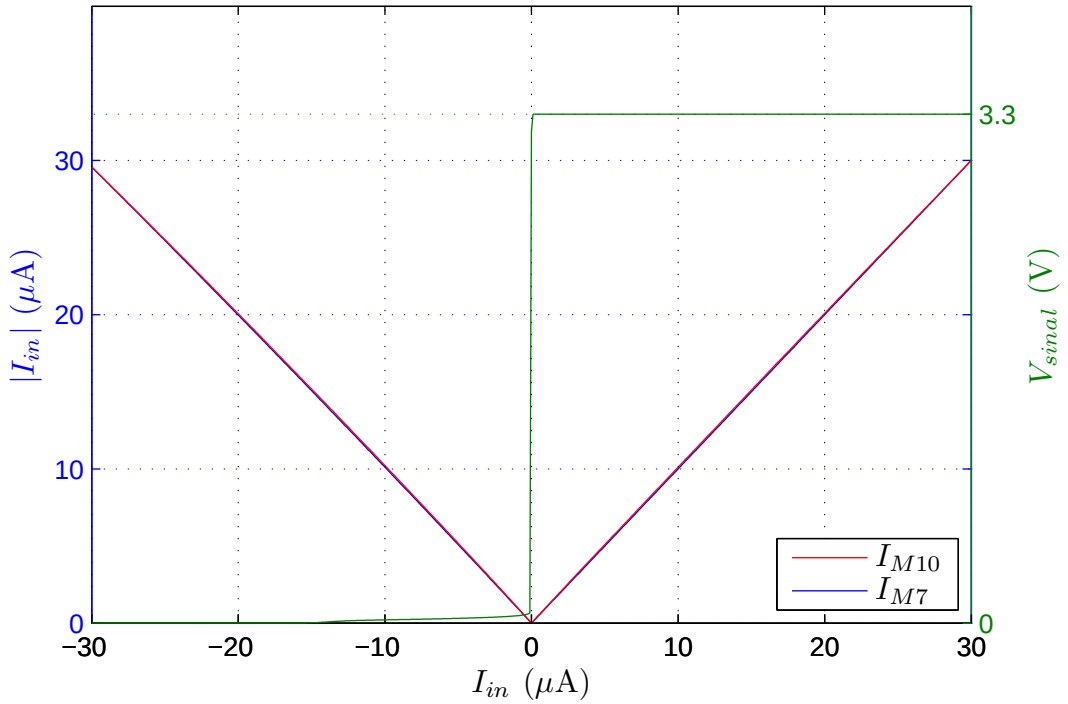
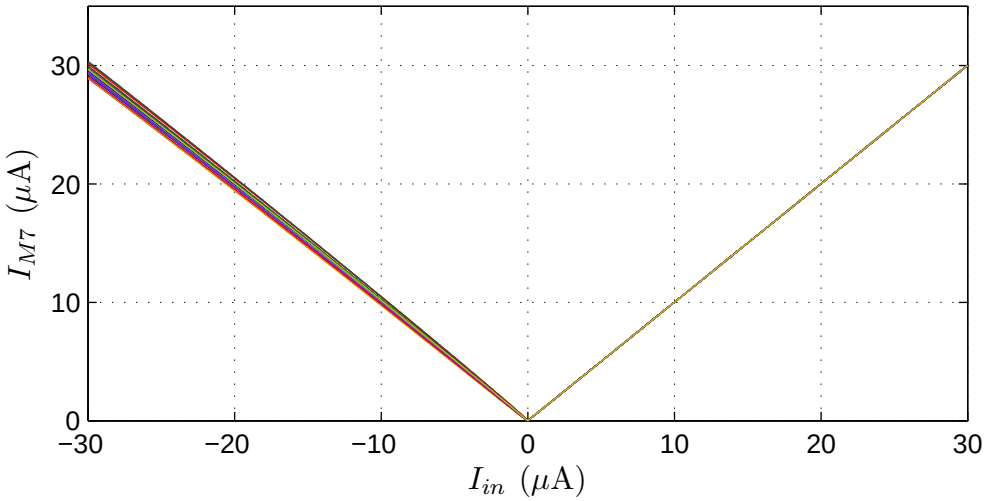



Figura 4.13: Simulação do circuito de módulo com parâmetros típicos.

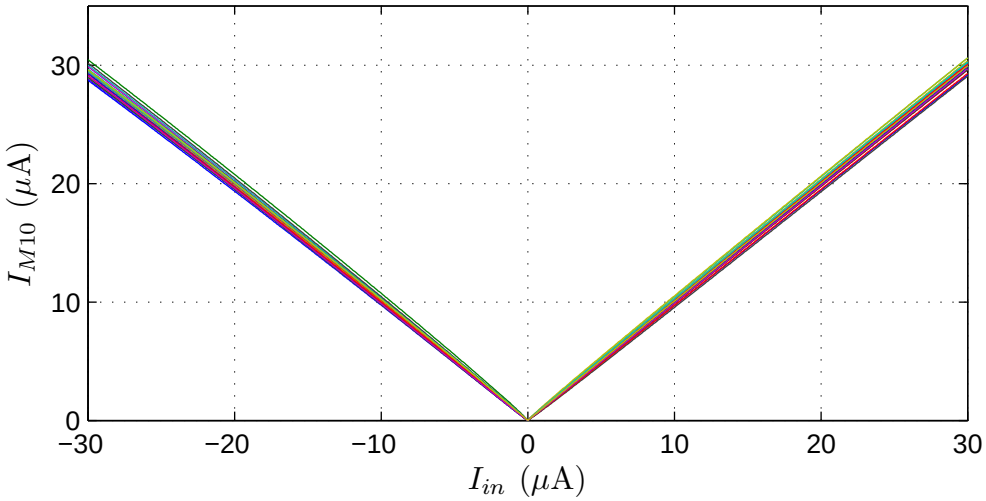
parâmetros de processo e descasamento) para os sinais de corrente. Ressalta-se que foi utilizada uma fonte de corrente ideal como entrada ($I_{in}$) para que fosse possível isolar os erros causados pelo circuito de módulo.

A variação ocorre pois este circuito possui espelhos de corrente que funcionam como descrito na Seção 4.1. Note que a corrente no transistor $\mathbf{M_7}$, quando a corrente de entrada é positiva, não sofre variações. Isto acontece porque a corrente não é copiada nenhuma vez no circuito de módulo.

A Tabela 4.10 mostra as variações das correntes $I_{M7}$ e $I_{M10}$ para alguns valores




(a)




(b)

Figura 4.14: (a) Simulação de Monte Carlo $I_{in} \times I_{M7}$; (b) Simulação de Monte Carlo $I_{in} \times I_{M10}$.

de $I_{in}$.

Tabela 4.10: Tabela de variações de Monte Carlo das correntes do circuito de módulo.

| $I_{in}(\mu A)$ | $I_{M7}$ | | | $I_{M10}$ | | |
|---|---|---|---|---|---|---|
| | média ($\mu$A) | $\delta$ ($\mu$A) | % | média ($\mu$A) | $\delta$ ($\mu$A) | % |
| -30 | 29,525 | 0,44 | 1,5 | 29,418 | 0,46 | 1,6 |
| -20 | 19,917 | 0,34 | 1,7 | 19,913 | 0,37 | 1,8 |
| -10 | 10,071 | 0,24 | 2,3 | 10,114 | 0,25 | 2,5 |
| 10 | 10 | 0 | 0 | 10,043 | 0,25 | 2,5 |
| 20 | 20 | 0 | 0 | 19,997 | 0,35 | 1,8 |
| 30 | 30 | 0 | 0 | 29,887 | 0,42 | 1,4 |

É possível perceber que apesar de o circuito possuir muitas componentes, a variação é razoável para circuitos que utilizam espelhos de corrente.

A Figura 4.15 mostra a simulação de Monte Carlo para a saída digital. Para esta saída, as variações são desprezíveis. Isto acontece pois este circuito não depende de elementos sensíveis a variações tanto do processo de fabricação quanto de descasamento dos transistores.

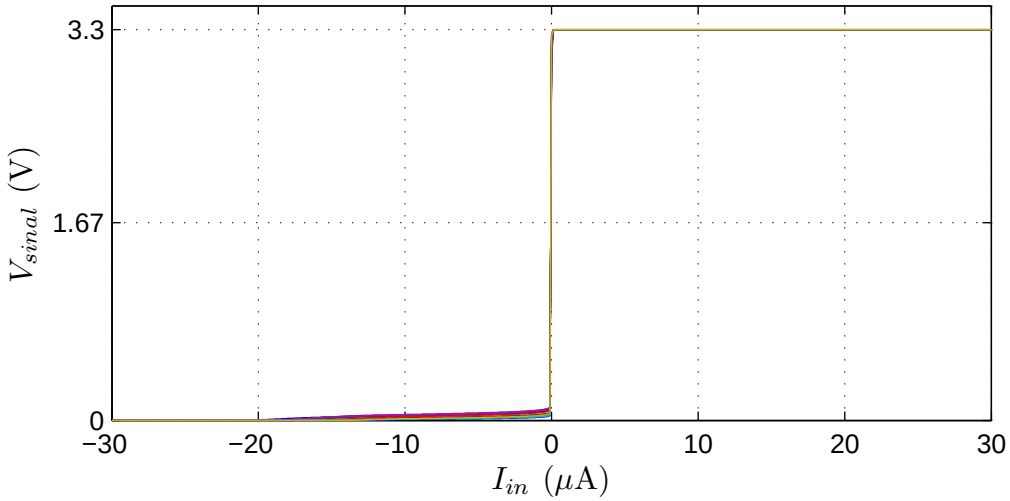



Figura 4.15: Simulação de Monte Carlo $I_{in} \times V_{sinal}$.

## 4.6 Circuito de Quantização Escalar

Como descrito na Seção 2.1, utilizamos quantizadores escalares do tipo *flash* para digitalizar as correntes necessárias. Isto é feito comparando a corrente em questão com as correntes de limiar, gerando um código digital do tipo termômetro.

Esta operação é realizada em duas partes do circuito, para quantizar o sinal $|e(n)|$ do circuito de DPCM e para quantizar os elementos do vetor $\mathbf{x}(n)$ no circuito de VQ. No quantizador do circuito de DPCM (Figura 4.16), a subtração entre $|e(n)|$ e $t_{0,k}$ (k-ésimo elemento de $\mathbf{t_0}$) é feita utilizando o circuito da Figura 4.17.

Para quantizar o sinal $\mathbf{f}(n)$ no circuito de VQ (Figura 4.18), a subtração entre $f_i(n)$ (i-ésimo elemento de $\mathbf{f}(n)$) e $t_k$ (k-ésimo limiar de $\mathbf{t}$) é feita utilizando o circuito da figura 4.19.

Pode-se notar que, para simplificar o processo e minimizar o número de transistores, o produto interno ($\mathbf{Ux}(n)$) é feito cada vez que uma comparação é necessária, desta forma, evitamos o aparecimento de mais uma cópia de corrente, diminuindo o erro final.

### 4.6.1 Comparadores de Corrente

O comparador é composto apenas de um amplificador inversor com alta impedância de entrada que funciona sempre saturado. Quando a corrente nos transistores PMOS

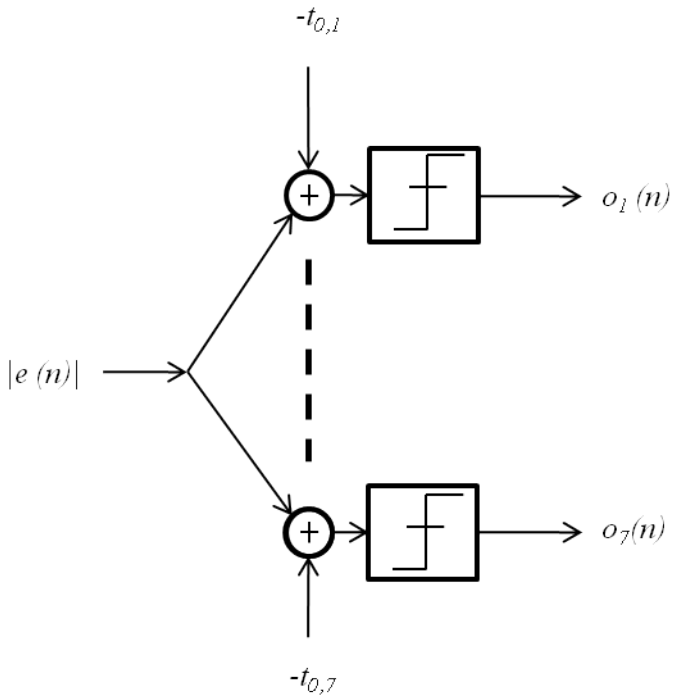



Figura 4.16: Esquema de quantização de $|e(n)|$.

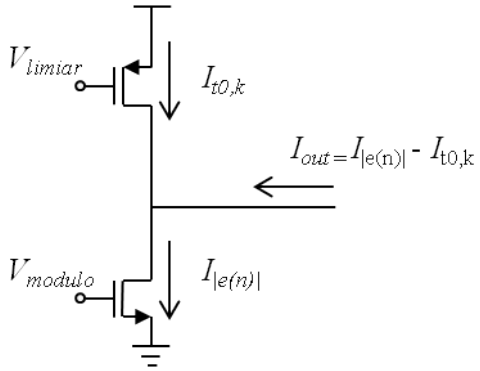



Figura 4.17: Circuito de subtração entre $|e(n)|$ e $t_{0,k}$.

é maior que a corrente nos transistores NMOS, o nó de entrada fica em tensão alta (3,3 V), fazendo com que a saída fique em baixa tensão (0 V). Se a corrente total nos transistores NMOS for maior, o nó de entrada fica em tensão baixa (0 V), fazendo com que a saída fique em alta tensão (3,3 V).

A Figura 4.20 contém o esquemático do circuito de comparador de corrente. Todos os transitores deste circuito têm $W = 1$ $\mu$m e $L = 0,35$ $\mu$m.

A Figura 4.21 mostra a simulação de Monte Carlo, levando em conta variações de processo de fabricação e descasamento isolados e simultaneamente, da transformação linear para a linha 2 da matriz **U** (-0,5 0,5 -0,5 0,5), mantendo $I_{x,2} = 2,0$ $\mu$A, $I_{x,3} = 3,0$ $\mu$A, $I_{x,4} = 4,0$ $\mu$A, $I_t = 1,5$ $\mu$A e variando $I_{x,1}$ de 0 a 10 $\mu$A. A saída do circuito está conectada ao comparador de corrente.

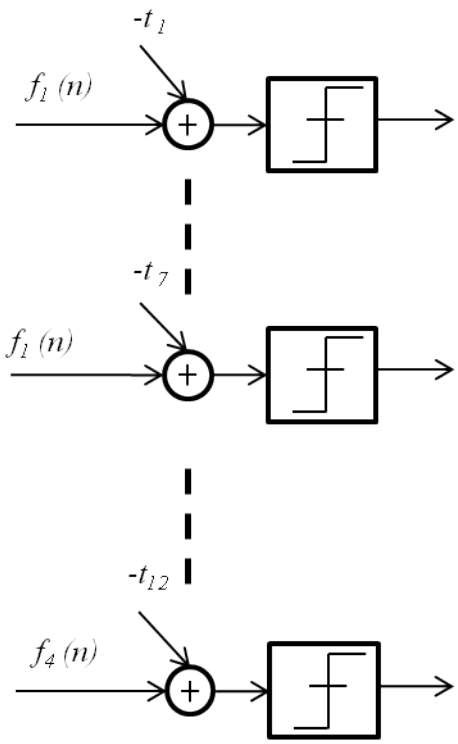



Figura 4.18: Esquema de quantização de $f_1(n)$ a $f_4(n)$.

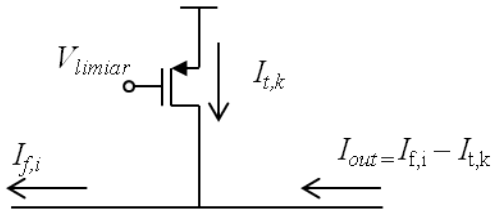



Figura 4.19: Circuito de subtração entre $f_i(n)$ e $t_k$.

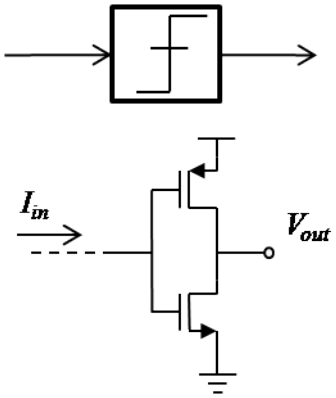



Figura 4.20: Esquemático do circuito comparador de corrente.

Como esperado, o valor de $I_{x,1}$ em que há a transição de 3,3 V para 0 V ocorre quando $I_{x,1} = 6$ $\mu$A:

$$-0,5x_1 + 0,5x_2 - 0,5x_3 + 0,5x_4 = -I_t$$

$$-0,5x_1 + 1 - 1,5 + 2,0 = -1,5$$

$$x_1 = \frac{3,0}{0,5} = 6,0$$

Como este circuito possui muitos espelhos de corrente, e somamos várias saídas destes espelhos em um mesmo nó, sendo este de alta impedância, este circuito é muito sensível às variações de descasamento. O desvio-padrão para $I_{x,1}$ quando $V_{out} = 1,67$ V foi de $0,25$ $\mu$A.

## 4.7 Codificador Lógico

Para transformar os bits de saída do DPCM e do VQ, que estão no formato termômetro, para uma forma mais compacta, iremos utilizar portas lógicas do tipo XNOR (inversa da porta lógica ou-exclusiva), formando na saída um código *gray* (Seção 2.1). A Figura 4.22 mostra o codificador de 3 bits em termômetro para 2 bits em código *gray*, que formam os bits $b_{3,4}(n)$ e $b_{3,5}(n)$ do VQ (Figura 3.7). Este circuito resulta na Tabela 4.11.

Tabela 4.11: Tabela verdade para um codificador 3 para 2.

| Entrada | Saída |
|---|---|
| 000 | 01 |
| 001 | 00 |
| 011 | 10 |
| 111 | 11 |

A expressão lógica para o codificador de 7 bits ($o(n)$) em termômetro para 3 bits em código gray do DPCM, $b_{1,2}(n)$ a $b_{1,4}(n)$, (Figura 3.3) e para os bits $b_{3,1}(n)$ ao $b_{3,3}(n)$ do VQ (Figura 3.7) é:

$$b_{1,2}(n) = o_4$$

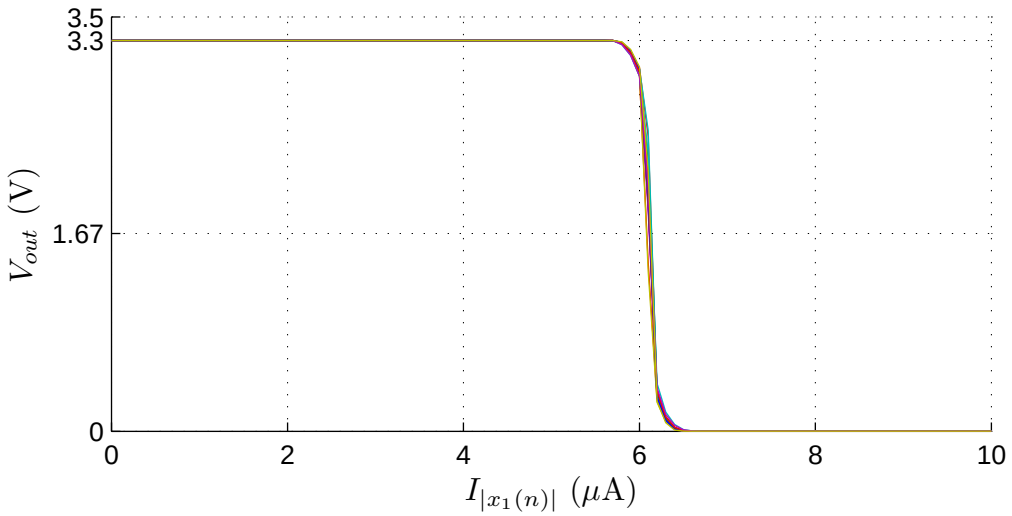
$$b_{1,3}(n) = \overline{o_2 \oplus o_6}$$

$$b_{1,4}(n) = \overline{\overline{o_1 \oplus o_3} \oplus \overline{o_5 \oplus o_7}}$$

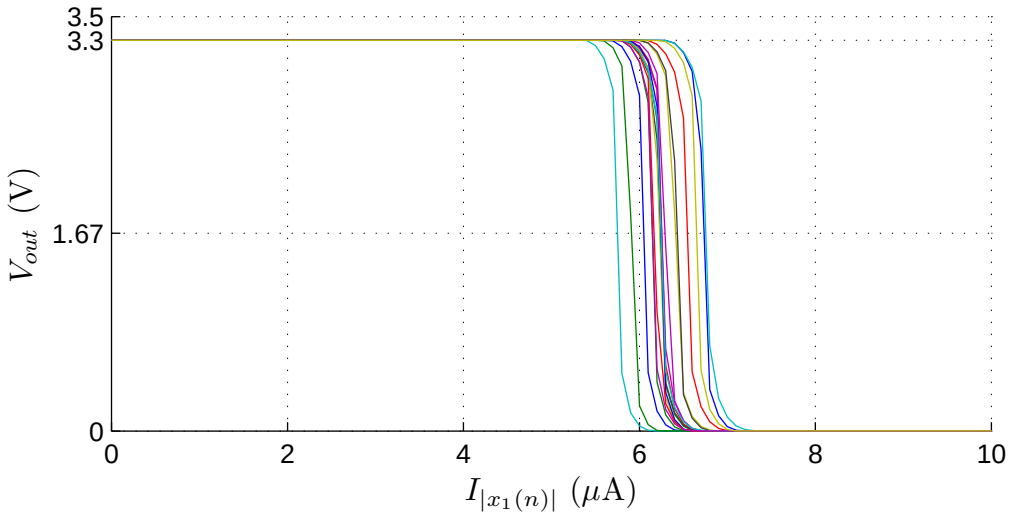
Este circuito resulta na Tabela 4.12.

### Porta XNOR

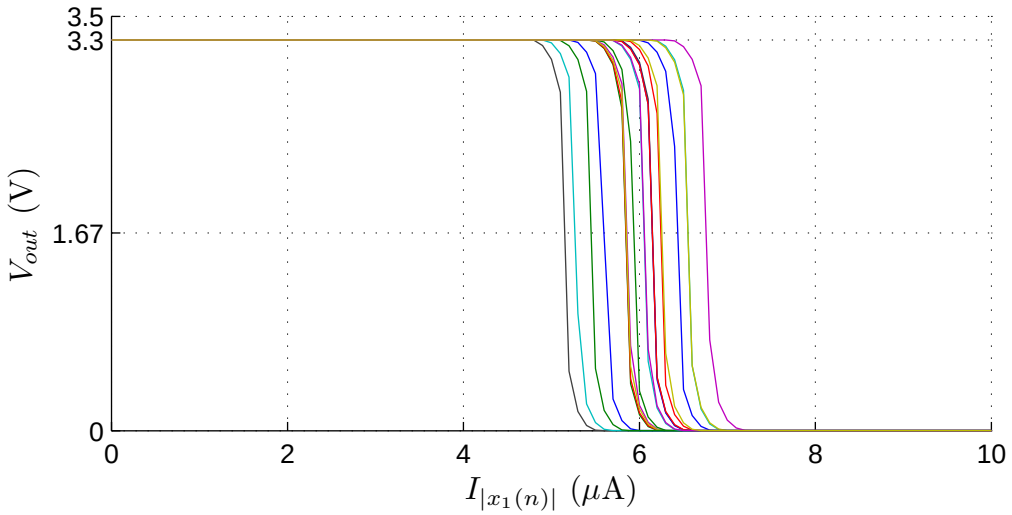
A Figura 4.23 mostra o circuito utilizado para implementar a porta lógica XNOR. Todos os transitores deste circuito têm $W = 1$ $\mu$m e $L = 0,35$ $\mu$m. Este circuito resulta na Tabela 4.13




(a)




(b)




(c)

Figura 4.21: Simulação de Monte Carlo para o circuito de produto interno: (a) apenas variações de processo; (b) apenas variações de descasamento; (c) ambas variações.

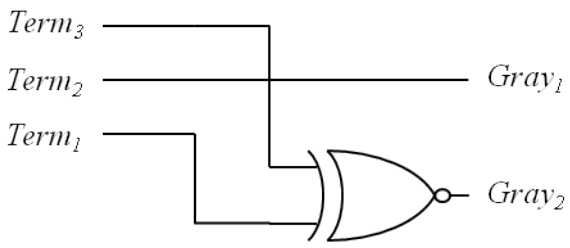



Figura 4.22: Esquema lógico para o codificador de 3 para 2 bits.

Tabela 4.12: Tabela verdade para um codificador 7 para 3.

| Entrada | Saída |
|---|---|
| 0000000 | 011 |
| 0000001 | 010 |
| 0000011 | 000 |
| 0000111 | 001 |
| 0001111 | 101 |
| 0011111 | 100 |
| 0111111 | 110 |
| 1111111 | 111 |

Tabela 4.13: Tabela verdade para a porta lógica XNOR.

| **A** | **B** | $\overline{\mathbf{A} \oplus \mathbf{B}}$ |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

## 4.8 DPCM

A primeira operação realizada no DPCM (Seção 2.2) é o cálculo da diferença entre $s(n)$ e $\hat{s}(n-1)$, resultando em $e(n)$. A média $s(n)$ é calculada copiando as correntes oriundas do circuito de CDS ($y(n)$), através de transistores PMOS, e conectando as correntes de saída ao mesmo nó. Para efetuar a subtração, basta copiar a corrente reconstruída do bloco anterior (Seção 4.8.1) com um transistor NMOS e conectar no mesmo nó de saída como mostrado na Figura 4.24.

Todos os transistores deste circuito possuem $W = 1$ $\mu$m e $L = 2$ $\mu$m. Isto faz com que a corrente resultante da média seja quatro vezes maior que o escalamento usado nos circuitos de VQ, pois a entrada dos espelhos é formada pelo transistor $\mathbf{M_6}$ do circuito de CDS (Seção 4.3) de dimensões $W = 4$ $\mu$m para $L = 2$ $\mu$m. Isto é feito desta maneira para aumentar a precisão do circuito (as correntes usadas para os limiares serão maiores) e é importante pois o DPCM acumula erros através dos blocos.

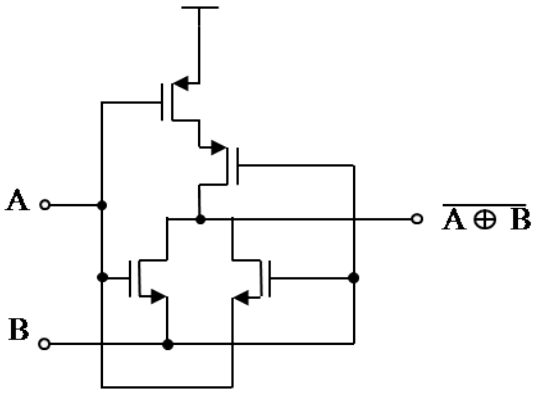



Figura 4.23: Circuito da porta XNOR.

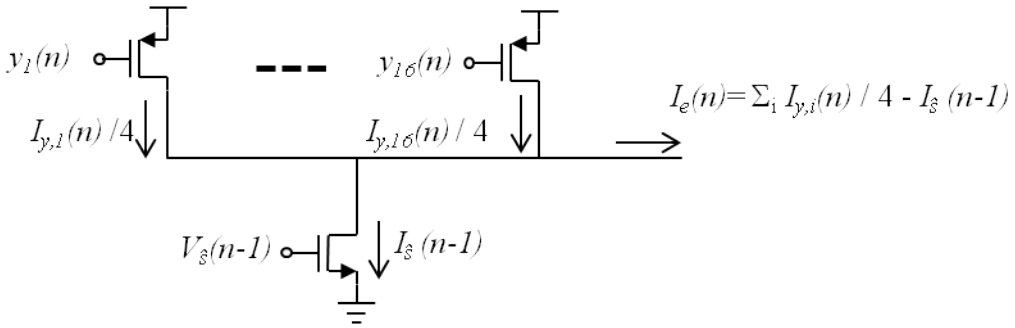



Figura 4.24: Circuito para subtrair $s(n)$ pela corrente $\hat{s}(n-1)$.

Como a corrente resultante pode ser tanto positiva quanto negativa vamos utilizar o circuito de módulo DPCM (Seção 4.5) antes de quantizarmos $e(n)$. Assim, poderemos utilizar o sinal desta corrente como $b_{1,1}(n)$ e quantizar $|e(n)|$ em $\mathbf{o}(n)$ com 7 limiares ao invés de 15 (considerendo o sinal para a quantização). Isto pode ser feito pois os limiares de quantização são simétricos.

Para quantizar $|e(n)|$, utilizaremos o circuito da Seção 4.6 para comparar $|e(n)|$ com os limiares do circuito de DPCM, $\mathbf{t_0}$, formando um código termômetro, $\mathbf{o}(n)$, que será tanto usado para compor $b_1$ quanto para reconstruir $e(n)$ (Seção 4.8.1).

A partir do codificador lógico (Seção 4.7), podemos codificar $\mathbf{o}$(n), que possui 7 bits, em $b_{1,2}(n)$, $b_{1,3}(n)$ e $b_{1,4}(n)$,ou seja, com três bits.

### 4.8.1 Circuito de Reconstrução

O circuito de reconstrução faz parte da implementação do DPCM (Seção 3.3). Este circuito permite que o sinal digitalizado seja reconstruído para uso no bloco seguinte de DPCM. A corrente deve ser reconstruída a partir do sinal digital pois deve corresponder ao sinal disponível ao decodificador.

Inicialmente o módulo da corrente $I_{e(n)}$ é reconstruído a partir de $\mathbf{o}$(n) e $\Delta$. Isto é feito a partir da soma dos elementos de $\Delta$ chaveados com o vetor binário $\mathbf{o}$(n), isto é, quando um elemento de $\mathbf{o}$(n) for igual a 1 (3,3 V), a corrente correspondente

no vetor $\Delta$ ($I_\Delta$) será somada. Cada corrente referente a um elemento de $\Delta$ é gerada pelo circuito da Figura 4.25.

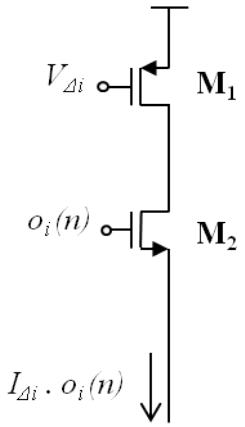



Figura 4.25: Bloco básico de reconstrução da corrente.

O transistor $\mathbf{M_1}$ copia a corrente correspondente ao elemento de $\Delta$ e o transistor $\mathbf{M_2}$ funciona como uma chave, controlada pelo sinal termômetro gerado no circuito de quantização, $o_i(n)$.

As dimensões dos transistores estão na Tabela 4.14.

Tabela 4.14: Dimensões dos transistores do bloco básico de reconstrução.

| Transistor | W ($\mu$m) | L ($\mu$m) |
|---|---|---|
| $\mathbf{M_1}$ | 1 | 2 |
| $\mathbf{M_2}$ | 1 | 0,35 |

Como as correntes que correspondem aos elementos de $\Delta$ são iguais para todo o circuito, elas são geradas externamente aos blocos e copiadas para os circuitos através de espelhos de corrente.

O esquemático completo para o circuito de reconstrução é mostrado na Figura 4.26.

## Funcionamento

Com $|\hat{e}(n)|$, precisamos apenas recuperar seu sinal e somar a $\hat{s}(n-1)$ para calcular $\hat{s}(n)$. Para tal, é utilizado o circuito da Figura 4.27. O resultado é a corrente $I_{\hat{s}}(n)$:

$$I_{\hat{s}}(n) = I_{\hat{s}}(n-1) + I_{\hat{e}}(n)$$

O transistor $\mathbf{M_1}$ copia a corrente referente a $|\hat{e}(n)|$ para os transistores $\mathbf{M_2}$ e $\mathbf{M_7}$. $\mathbf{M_3}$ por sua vez copia $I_{|\hat{e}(n)|}$ para $\mathbf{M_4}$. Então $\mathbf{M_5}$ e $\mathbf{M_6}$ funcionam como seletor de sinal para $I_{|\hat{e}(n)|}$ usando o sinal extraído no circuito de módulo DPCM, $b_{1,1}(n)$. Desta

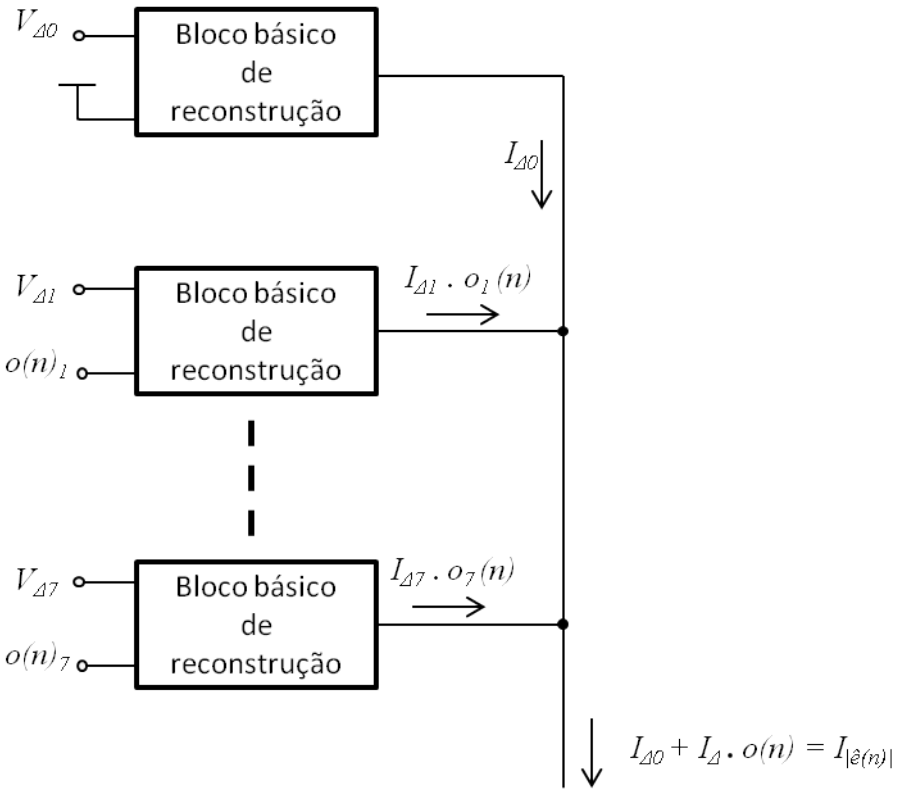



Figura 4.26: Circuito para reconstruir $|\hat{e}(n)|$.

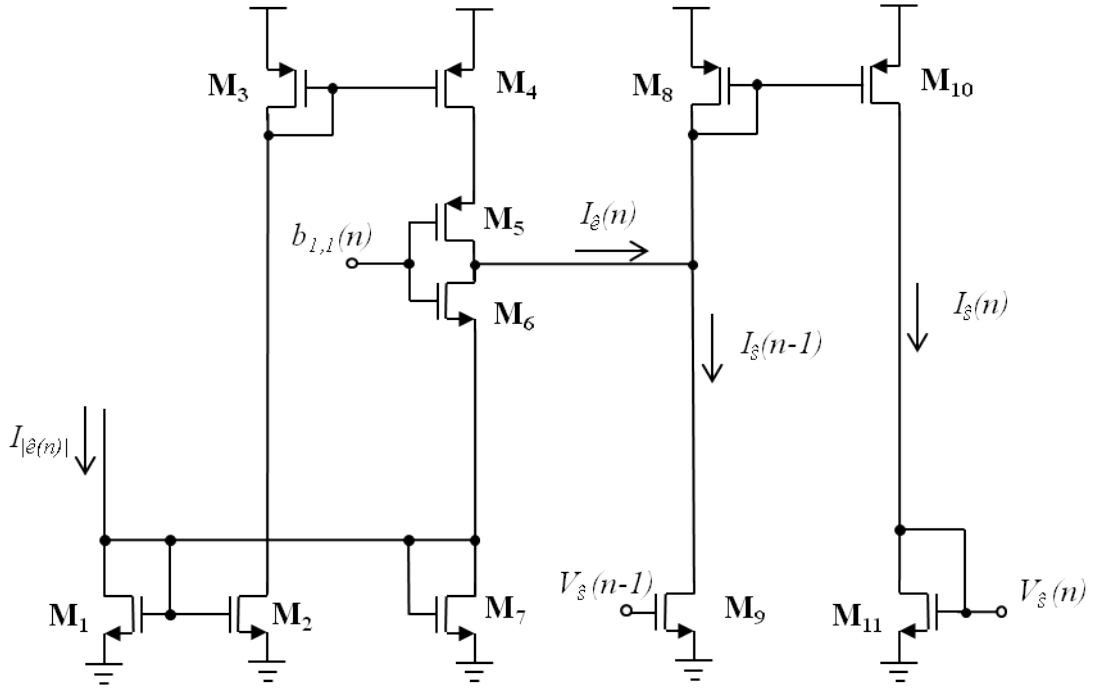



Figura 4.27: Circuito para subtrair a média das correntes de um bloco pela corrente reconstruída do bloco anterior.

forma obtemos uma corrente referente a $\hat{e}(n)$, incluindo seu sinal: para $b_{1,1}(n) = 0$ V a corrente $I_{\hat{e}(n)}$ é positiva (no sentido da figura) e para $b_{1,1}(n) = 3,3$ V a corrente

$I_{\hat{e}(n)}$ é negativa.

Para somar $\hat{e}(n)$ ao sinal reconstruído do bloco anterior, $\hat{s}(n-1)$, copiamos a corrente referente a $\hat{s}(n-1)$ com $\mathbf{M_9}$ e conectamos ao mesmo nó. A soma destas correntes é referente ao sinal $\hat{s}(n)$. Para que esta corrente seja copiada através de um transistor NMOS para o bloco $n+1$ é necessário o espelho formado pelos transistores $\mathbf{M_8}$ e $\mathbf{M_{10}}$. $\mathbf{M_{11}}$ copia a corrente referente a $\hat{s}(n)$ para o bloco seguinte.

A Tabela 4.15 contém as dimensões dos transistores utilizados neste circuito.

Tabela 4.15: Dimensões dos transistores do circuito de reconstrução.

| Transistor | W ($\mu$m) | L ($\mu$m) |
|---|---|---|
| $\mathbf{M_1}$ | 2 | 2 |
| $\mathbf{M_2}$ | 1 | 2 |
| $\mathbf{M_3}$ | 1 | 2 |
| $\mathbf{M_4}$ | 1 | 2 |
| $\mathbf{M_5}$ | 1 | 0,35 |
| $\mathbf{M_6}$ | 1 | 0,35 |
| $\mathbf{M_7}$ | 1 | 2 |
| $\mathbf{M_8}$ | 1 | 2 |
| $\mathbf{M_9}$ | 1 | 2 |
| $\mathbf{M_{10}}$ | 1 | 2 |
| $\mathbf{M_{11}}$ | 1 | 2 |

## Simulações

A Figura 4.28 mostra a simulação do circuito de reconstrução de sinal completo. Considerou-se que $I_{\hat{s}(n-1)} = 10$ $\mu$A e $I_{s(n)}$ varia de 0 $\mu$A a 30 $\mu$A.

Como pode ser observado na Figura 4.28, o sinal $I_{\hat{s}(n)}$ tem uma forma de "escada", pois passa a assumir apenas valores determinados. Estes valores são os limiares $\mathbf{t}_0$ somados ou subtraídos de $\hat{s}(n-1)$, fazendo com que este sinal corresponda ao valor do decodificador digital (*software*).

A Figura 4.29 mostra a simulação de Monte Carlo do circuito de reconstrução, levando em conta variações de processo de fabricação e descasamento isolados e simultaneamente e considerando os mesmos parâmetros de entrada da Figura 4.28. Este circuito, por possuir muitas cópias por espelhos de corrente, é mais sensível às variações de descasamento. Como qualquer erro que ocorre na reconstrução do sinal é acumulado pelo DPCM, é essencial que se empreguem técnicas de *layout* que minimizem erros provenientes do descasamento entre transistores que compõem os espelhos de corrente neste circuito. Note que os erros ocorrem tanto para os patamares de corrente reconstruídos quanto para os limiares de quantização.

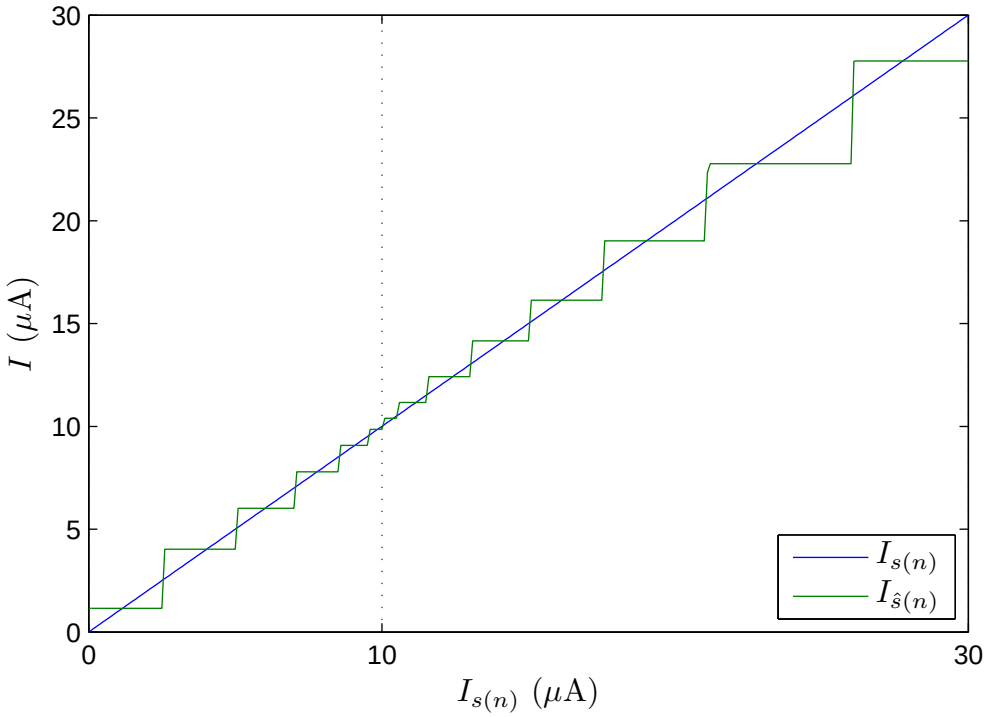
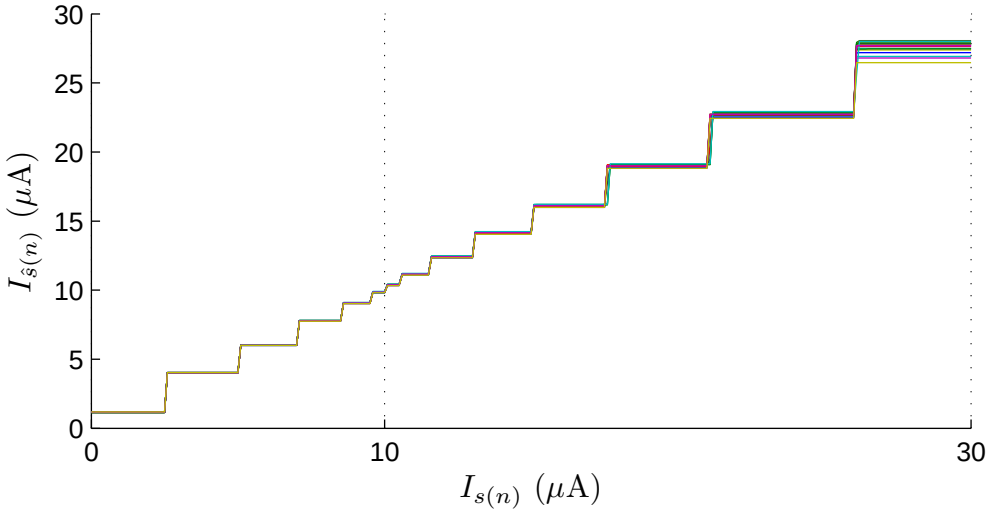
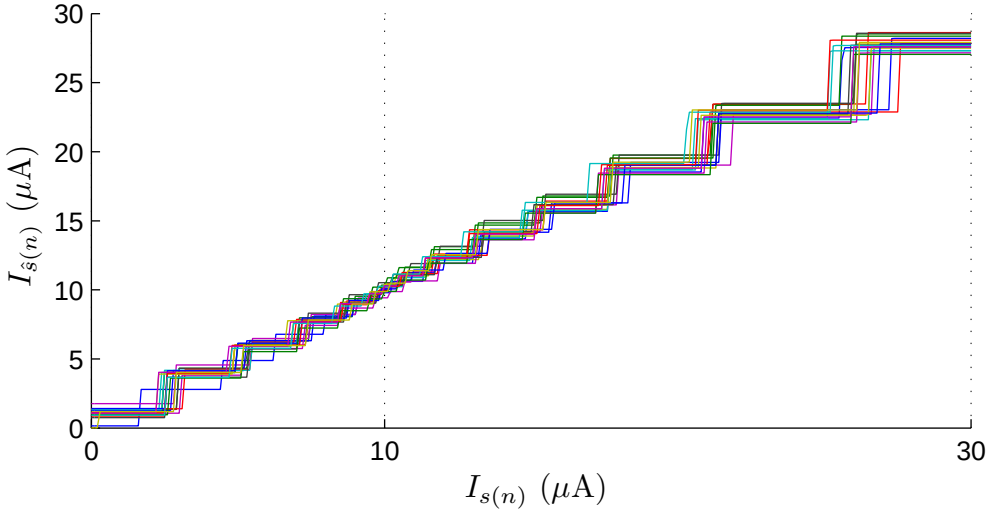


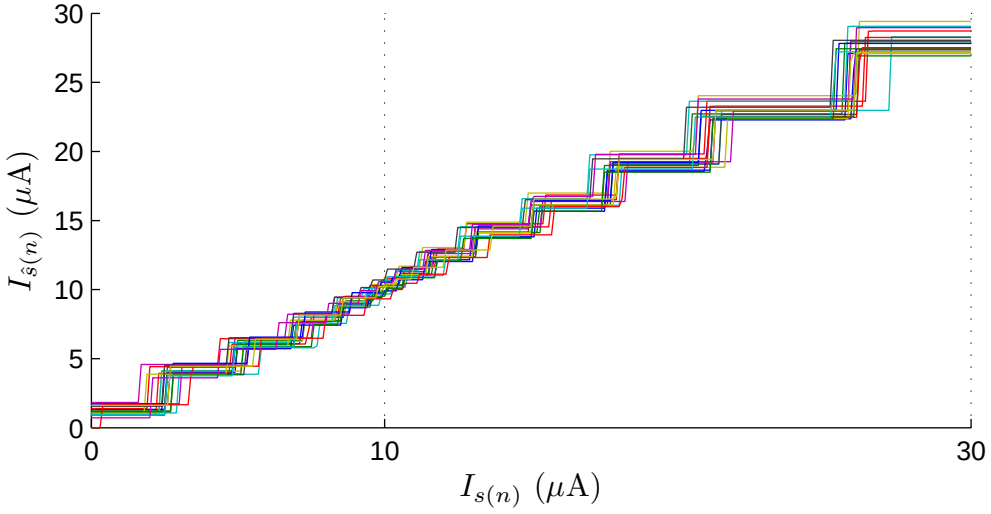

Figura 4.28: Simulação da reconstrução do sinal de entrada.




(a)




(b)




(c)

Figura 4.29: Simulação de Monte Carlo para o circuito de reconstrução: (a) apenas variações de processo; (b) apenas variações de descasamento; (c) ambas variações.

# Capítulo 5

# Resultados e Discussões

Neste capítulo serão apresentadas as figuras resultantes das simulações com os circuitos apresentados no Capítulo 4 e os métodos apresentados no Capítulo 2. Os resultados serão comparados e discutidos.

A imagem de teste que utilizaremos é uma subimagem, de 32 × 32 pixels, da imagem "lena.bmp" de 512 × 512 pixels, conforme mostrado na Figura 5.1. A utilização da imagem reduzida se dá para compatibilizá-la ao protótipo fabricado, cujos resultados estão na seção 5.4.

## 5.1 Figuras de Mérito

Para medir a qualidade da compressão de imagens utilizaremos duas figuras de mérito: PSNR (*Peak Signal-to-Noise Ratio* ou razão sinal-ruído de pico) e distorção ($D$, Seção 2.4).

Utilizando as definições dos capítulos anteriores, a PSNR, que mede a qualidade final da imagem reconstruída em relação à original, é definida como:

$$Q = 10 \log_{10} \left( \frac{255^2}{N} \sum ||\mathbf{y}(n) - \hat{\mathbf{y}}(n)||^2 \right) \text{dB}$$

É importante ressaltar que o valor da PSNR pode não corresponder à qualidade subjetiva da imagem, isto é, uma imagem com PSNR maior pode não ser percebida como mais próxima da imagem original. Utilizamos 255 como valor de referência para podermos comparar imagens geradas pelo circuito a imagens digitais considerando 8 bits por pixel, portanto uma faixa de valores de 0 a 255.

A distorção $D$, que mede a diferença entre os vetores de componentes principais comprimidos pelo VQ reconstruídos e originais, é definida como:

$$D = \frac{\sum (\mathbf{x}(n) - \hat{\mathbf{x}}(n))^2}{N}$$

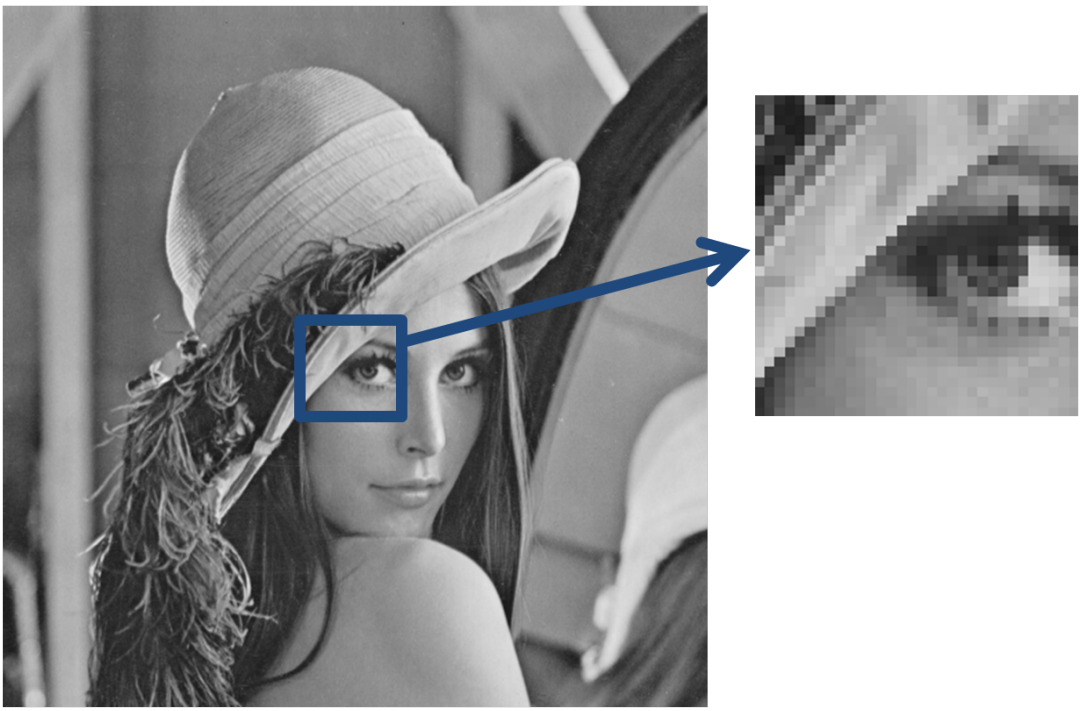


Figura 5.1: Corte feito na figura “lena.bmp”.

## 5.2 Compressão por Software

Para que possamos situar as imagens geradas pela simulação dos circuitos apresentados neste trabalho, primeiramente serão mostradas imagens de referência, contidas na Figura 5.2, geradas da seguinte forma:

- (a) Original: imagem gerada sem nenhuma compressão;
- (b) Somente DPCM: imagem gerada utilizando apenas o DPCM e descartando as informações internas dos blocos;
- (c) Somente PCA: imagem utilizando somente PCA mas sem compressão das componentes principais ou das médias dos blocos;
- (d) Pior Caso: imagem utilizando o DPCM e o PCA mas descartando as componentes principais (utiliza-se somente a componente mais provável);
- (e) Sem arredondamento: simulação completa utilizando valores ideais sem arredondamento dos parâmetros;
- (f) Com arredondamento: simulação completa utilizando valores arredondados dos parâmetros idênticos aos do circuito.

As imagens da Figura 5.2 resultam na Tabela 5.1.

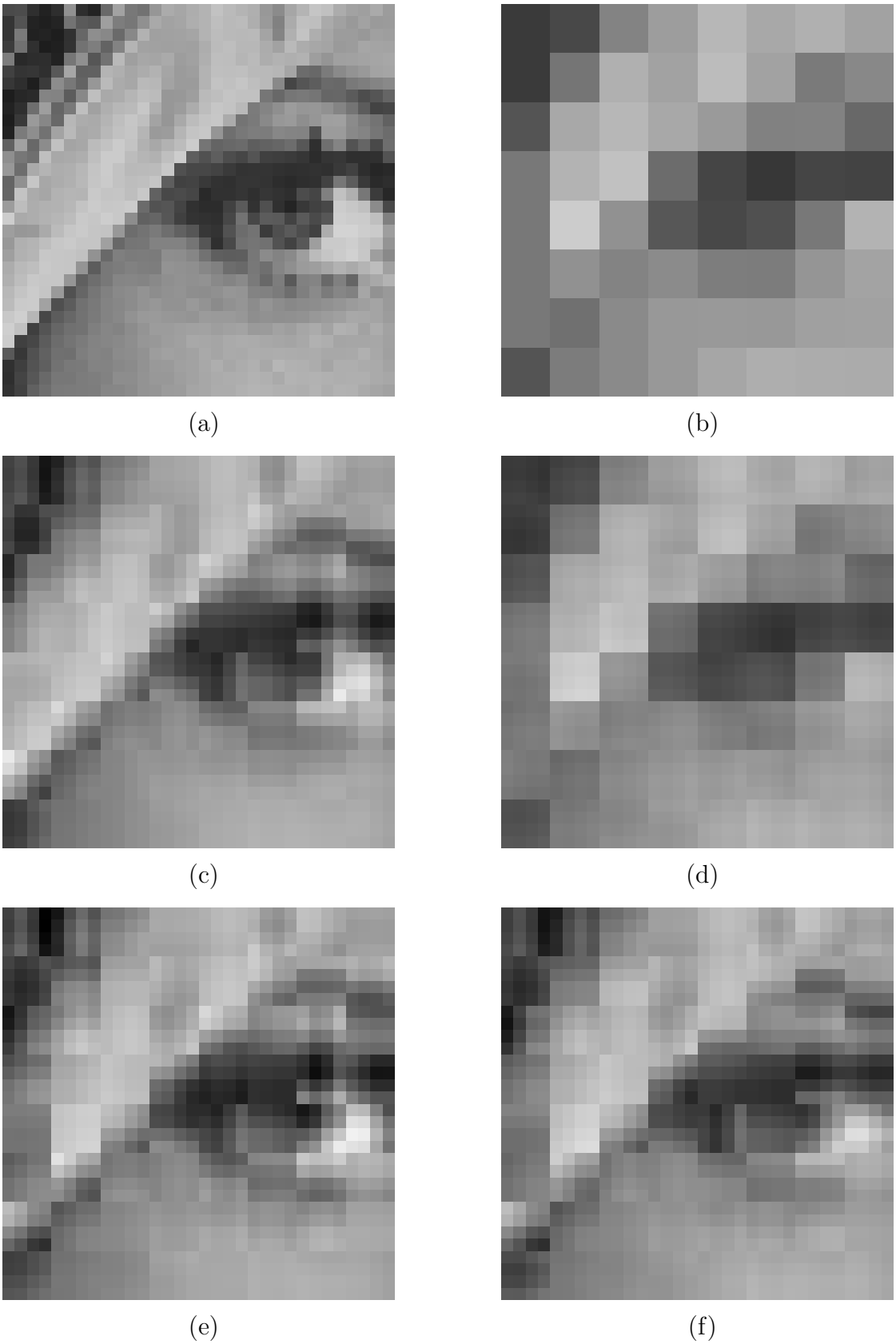


Figura 5.2: Exemplo de imagens comprimidas: (a) original; (b) somente DPCM; (c) somente PCA; (d) pior caso; (e) simulação completa sem arredondamento; (f) simulação completa com arredondamento.

Tabela 5.1: Resultados para as imagens de referência.

| Imagem | PSNR (dB) | D |
|---|---|---|
| (a) Original | - | - |
| (b) DPCM | 19,022 | - |
| (c) PCA | 24,617 | 0 |
| (d) Pior Caso | 19,626 | 0,0486 |
| (e) Sem arredondamento | 21,676 | 0,0189 |
| (f) Com arredondamento | 21,746 | 0,0086 |

Como pode ser observado, as imagens em que a informação do VQ e do PCA foram descartadas (casos (b) e (d)) são significativamente piores do que nos casos em que esta informação é aproveitada (casos (e) e (f)), como esperado.

Também é possível notar que o arredondamento dos parâmetros do VQ não resulta em diferença perceptível na qualidade (PSNR). Este resultado já era previsto em [20] e indica que o circuito não é sensível a pequenas variações nos parâmetros do VQ. Já para a distorção houve uma melhoria considerável, não esperada, quando aplicamos o arredondamento. Em geral, a distorção não sofre alteração significativa para este nível de arredondamento.

Os resultados das próximas seções devem ser comparados aos resultados da imagem (f), pois o circuito foi projetado para realizar exatamente as mesmas funções que foram utilizadas para obter este resultado.

## 5.3 Simulação de Compressão por Hardware

Para simular a compressão da imagem pelo circuito foi utilizado o *software* de simulação *Spectre* ® no circuito descrito por uma *netlist* construída a partir da repetição de 64 blocos idênticos $4 \times 4$ em uma matriz $8 \times 8$, formando uma imagem final de $32 \times 32$ pixels. Os circuitos descritos no Capítulo 4 foram usados para compor os blocos. A Seção B.2 mostra os códigos-fonte dos programas utilizados para criar os *netlists* e simulá-los.

Foram feitas simulações de Monte Carlo (MC) para variar os parâmetros de processo e descasamento dos transistores utilizados, aproximando os resultados das simulações dos resultados esperados em um circuito fabricado.

Utilizamos três circuitos de DPCM adicionais ao final de cada linha de blocos para implementar a correção mostrada na Seção 2.2.1. A análise dos resultados obtidos se encontra a seguir.

### 5.3.1 Simulações sem Circuito de Leitura

A Figura 5.3 mostra algumas imagens geradas pelas saídas do simulador sem utilizar os circuitos de leitura (Seção 4.2) e o circuito de CDS (Seção 4.3), isto é, usando apenas os circuitos para compressão. Este recurso foi usado para que fosse possível avaliar o sistema de compressão sem interferências do circuito de leitura, que introduz distorções na imagem capturada. Neste caso, somente foram necessárias simulações de ponto de operação (DC), já que sem o circuito de leitura não é necessário simular o comportamento do circuito no tempo.

Foram feitas 20 rodadas de Monte Carlo para esta simulação, na figura 5.3 está retratada apenas a rodada 13, escolhida aleatoriamente. O restante das imagens geradas desta forma está no Apêndice A.

As imagens da Figura 5.3 resultam na Tabela 5.2.

Tabela 5.2: Resultados para as imagens da simulação sem o circuito de leitura.

| Simulação | D | PSNR (dB) sem correção DPCM | PSNR (dB) com correção DPCM |
|---|---|---|---|
| Parâmetros Típicos | $0,0090$ | $19,9445$ | $20,2471$ |
| MC 1 | $0,0104$ | $19,8670$ | $19,8490$ |
| MC 2 | $0,0148$ | $20,2362$ | $20,3703$ |
| MC 3 | $0,0108$ | $21,2947$ | $20,5374$ |
| MC 4 | $0,0111$ | $17,5563$ | $19,4482$ |
| MC 5 | $0,0133$ | $18,9513$ | $19,6130$ |
| MC 6 | $0,0127$ | $18,5428$ | $19,2794$ |
| MC 7 | $0,0153$ | $18,3854$ | $19,6189$ |
| MC 8 | $0,0161$ | $19,0337$ | $18,9629$ |
| MC 9 | $0,0171$ | $19,5905$ | $19,7847$ |
| MC 10 | $0,0140$ | $16,9505$ | $19,8568$ |
| MC 11 | $0,0168$ | $20,3079$ | $19,7516$ |
| MC 12 | $0,0119$ | $20,3844$ | $20,4433$ |
| MC 13 | $0,0133$ | $19,4717$ | $20,0496$ |
| MC 14 | $0,0125$ | $20,1301$ | $20,3131$ |
| MC 15 | $0,0099$ | $18,7557$ | $19,4470$ |
| MC 16 | $0,0172$ | $19,4069$ | $19,4890$ |
| MC 17 | $0,0157$ | $19,3309$ | $19,6514$ |
| MC 18 | $0,0162$ | $20,5885$ | $20,2062$ |
| MC 19 | $0,0153$ | $20,3237$ | $20,3298$ |
| MC 20 | $0,0149$ | $20,9752$ | $20,1718$ |
| Média MC | $0,0140$ | $19,5042$ | $19,8587$ |
| Desvio Padrão | $0,0023$ | $1,1080$ | $0,4315$ |

Como pode ser observado, o valor da PSNR se aproxima do pior caso da compressão por *software* (Seção 5.2). Isto acontece por causa dos erros do circuito de DPCM, que se acumulam ao longo das linhas e afetam a média dos blocos, como

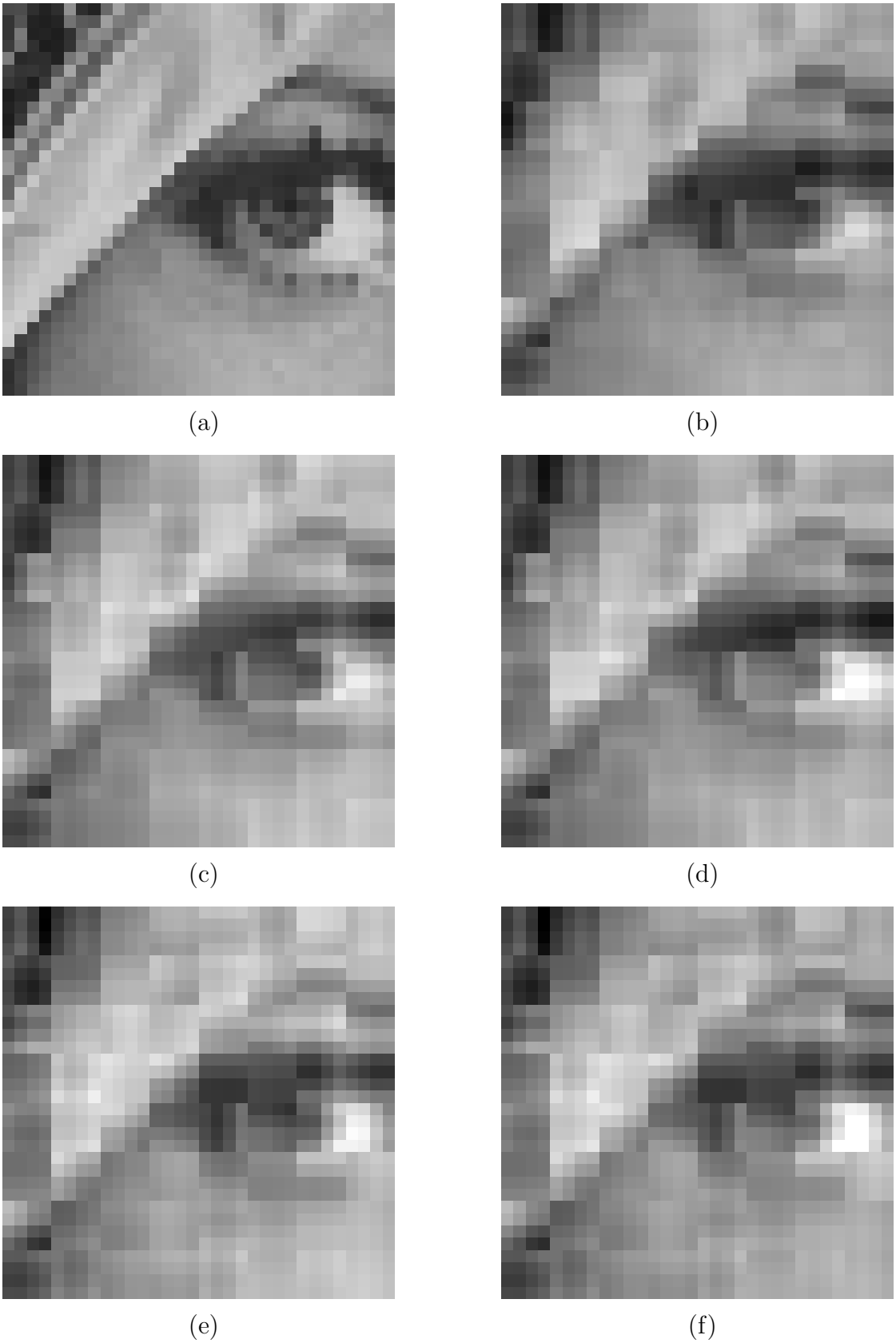


Figura 5.3: Simulações sem o circuito de leitura: (a) Original; (b) *software* com arredondamento; (c) parâmetros típicos sem correção de DPCM; (d) parâmetros típicos com correção de DPCM; (e) rodada de MC 13 sem correção de DPCM; (f) rodada de MC 13 com correção de DPCM.

visto na Seção 2.2.

Também é possível notar que a técnica de correção de DPCM aumentou a PSNR, em média, em 0,3545 dB ou 18%. Estes resultados evidenciam que a melhoria não é tão expressiva quanto a obtida na Seção 2.2.1, pois o comprimento das linhas é menor (oito blocos por linha na simulação do circuito contra 32 pixels por linha no exemplo dado na Seção 2.2.1).

Em um protótipo com maior resolução, são esperados erros maiores de reconstrução do DPCM (pois existem mais blocos para acumular erro) e portanto, a correção seria mais eficiente.

### 5.3.2 Simulações com Circuito de Leitura

A Figura 5.4 mostra algumas imagens geradas pelas saídas do simulador do circuito completo, incluindo o circuito de leitura. Neste caso, foram feitas simulações no tempo, utilizando 100 $\mu$s como tempo de integração. Antes do início da integração há um intervalo de 10 $\mu$s para acomodação do circuito. A leitura dos bits de saída é feita 10 $\mu$s após o término da integração para que os circuitos cheguem aos seus resultados finais.

Foram realizadas 5 rodadas de Monte Carlo, na figura 5.4 está retratada apenas a rodada 2, escolhida aleatoriamente. Todas as imagens geradas desta forma estão no Apêndice A.

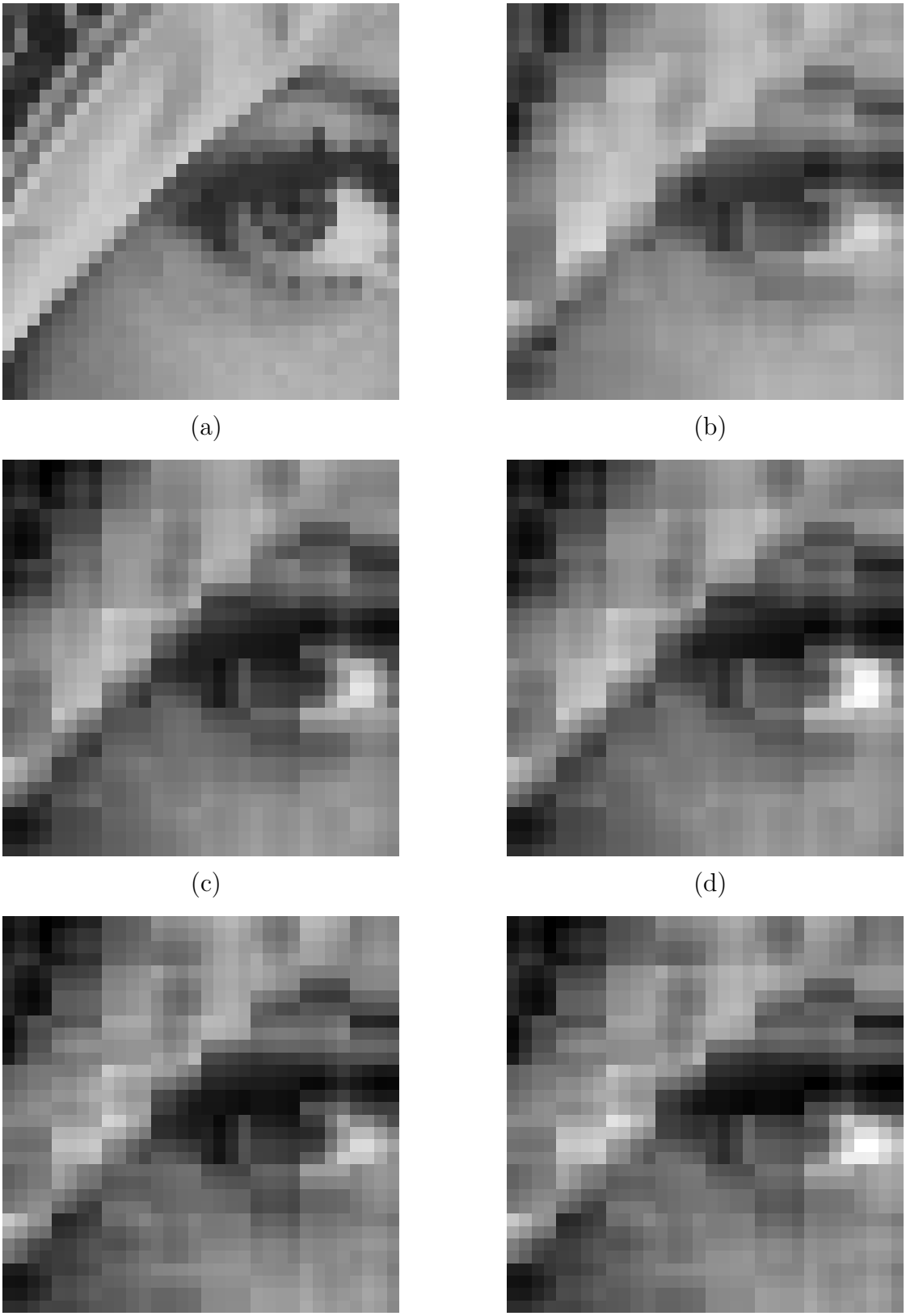
As imagens da Figura 5.4 resultam na Tabela 5.3

Tabela 5.3: Resultados para as imagens da simulação com o circuito de leitura.

| Simulação | D | PSNR (dB) sem correção DPCM | PSNR (dB) com correção DPCM |
|---|---|---|---|
| Parâmetros Típicos | 0.0156 | 17.1244 | 17.0909 |
| MC 1 | 0.0142 | 16.9272 | 16.4473 |
| MC 2 | 0.0201 | 16.5448 | 16.6099 |
| MC 3 | 0.0183 | 16.7591 | 16.3369 |
| MC 4 | 0.0212 | 17.6133 | 17.2019 |
| MC 5 | 0.0217 | 16.7983 | 16.4008 |
| Média MC | 0.0191 | 16.9285 | 16.5994 |
| Desvio Padrão MC | 0.0030 | 0.4068 | 0.3517 |

É possível perceber que as imagens comprimidas pelo circuito, quando reconstruídas, ficam mais escurecidas que a original. Isto ocorre, pois a curva de resposta do circuito de leitura em relação à luminância não é linear (Seção 4.2).

Este efeito faz com que os resultados se apresentem deteriorados. Isto se dá, pois a imagem sofre uma distorção no próprio circuito de entrada o que afetaria a imagem ainda que não houvesse compressão.



(a) (b) (c) (d) (e) (f)

Figura 5.4: Simulações com o circuito de leitura: (a) original; (b) *software* com arredondamento; (c) parâmetros típicos sem correção de DPCM; (d) parâmetros típicos com correção de DPCM; (e) rodada de MC 2 sem correção de DPCM; (f) rodada de MC 2 com correção de DPCM.

É possível perceber que a imagem se aproxima da compressão por *software* em termos subjetivos apesar da diferença de PSNR e distorção.

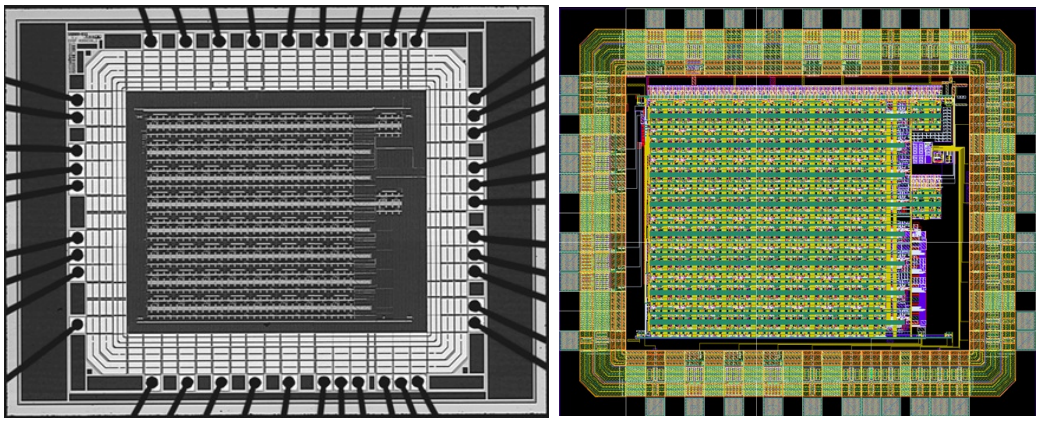
A correção do DPCM, neste caso, não surtiu o efeito desejado. Por causa das considerações acima, não foi possível determinar se a correção é efetiva. Seriam necessárias mais linhas para que a diferença fosse percebida de forma mais clara.

## 5.4 Medidas Obtidas com o Protótipo

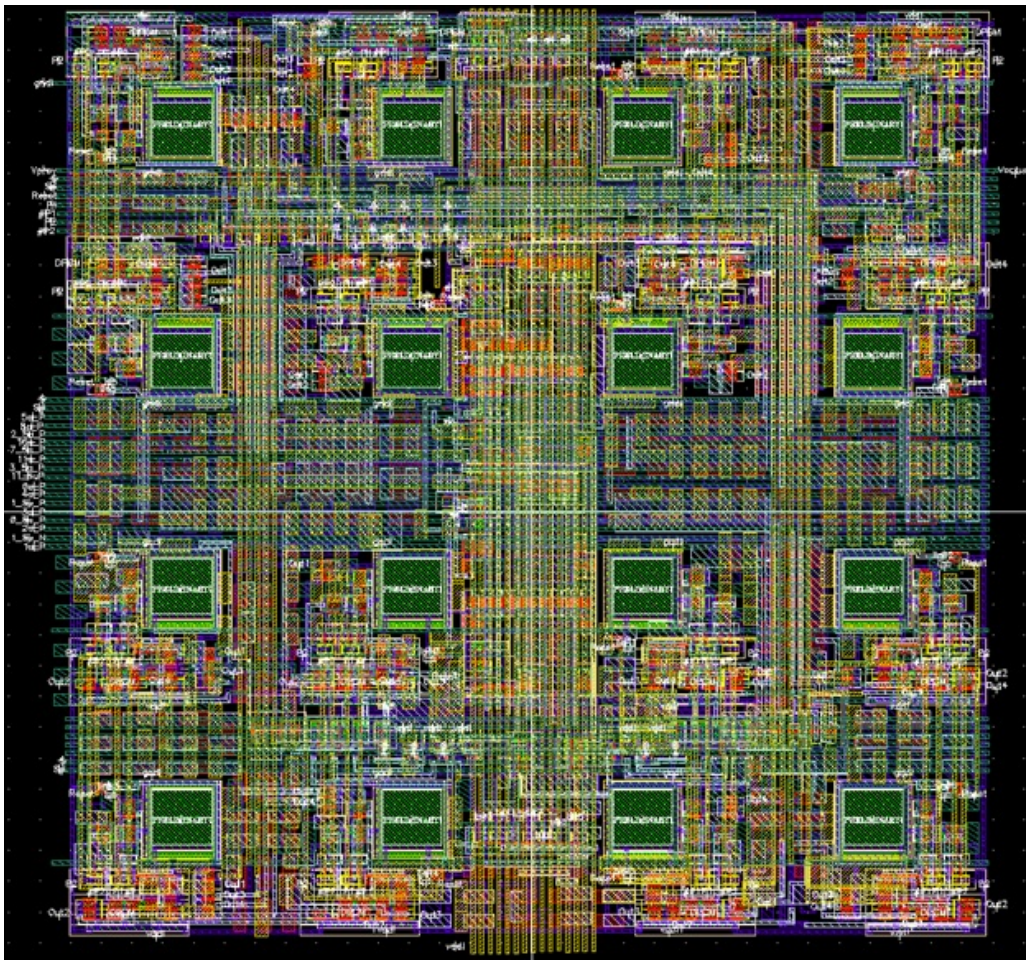
Um protótipo com $32 \times 32$ pixels foi fabricado utilizando os circuitos descritos neste trabalho com a tecnologia CMOS AMS 0,35 $\mu$m. Os detalhes da confecção do layout e das medidas se encontram em [27].

A Figura 5.4 mostra a máscara utilizada e o circuito fabricado. A Figura 5.4 mostra algumas fotografias de teste feitas com o chip fabricado e a figura 5.7 mostra uma sub-imagem de "lena.mat" de 96 $\times$ 96 pixels formada pela composição de fotografias de partes da imagem.

Cada bloco de 4×4 pixels ocupou uma área de 150 $\mu$m $\times$ 150 $\mu$m, ou seja, cada pixel ocupou 37,5 $\mu$m $\times$ 37,5 $\mu$m. O consumo de potência durante a captura variou de 30,8 mW (imagem completamente escura) a 37,0 mW (imagem branca).

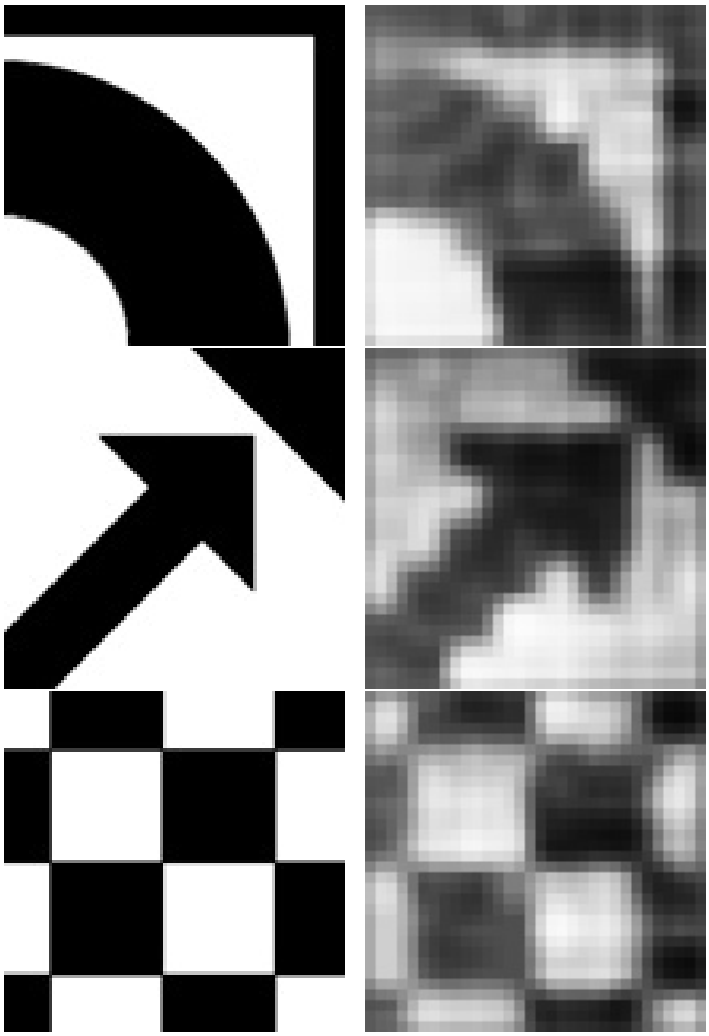


(a) (b)



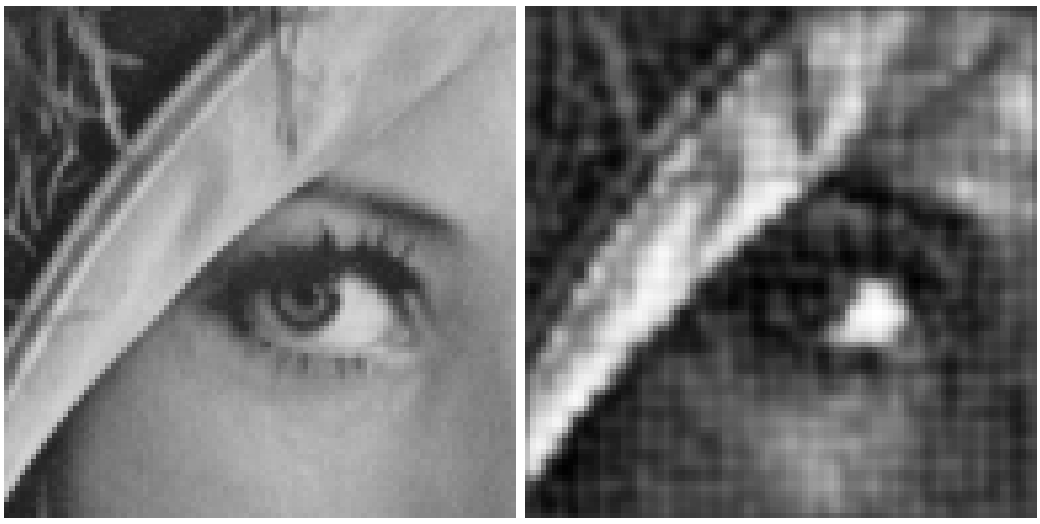
(c)

Figura 5.5: Imagens do chip fabricado: (a) fotografia do chip; (b) layout do chip; (c) layout do bloco 4×4 pixels.



(a) (b)

Figura 5.6: Fotografias realizadas pelo chip: (a) imagens originais (alvos); (b) fotografias.



(a) (b)

Figura 5.7: "Lena.mat": (a) imagem original; (b) fotografia.

# Capítulo 6

# Conclusões

O objetivo deste trabalho foi projetar um circuito capaz de comprimir imagens capturadas no próprio plano focal de câmeras. Isto é, a compressão e a captura das imagens deve ser realizada na mesma área de silício.

O método de compressão (Capítulo 2) utilizou as técnicas de DPCM, PCA e VQ, adequados à implementação no plano focal. Os circuitos que implementaram este método foram projetados em tecnologia CMOS $0,35$ $\mu$m.

O circuito foi simulado a fim de confirmar seu funcionamento e os resultados foram comparados à simulação feita por *software* pelo mesmo método de compressão. Os resultados apresentados demonstraram que, apesar das limitações e ruídos introduzidos pela realização das operações por circuitos analógicos, as imagens são comprimidas e podem ser reconstruídas de forma satisfatória.

## 6.1 Trabalhos Futuros

### 6.1.1 Controle Automático do Tempo de Integração

Como mostrado acima, é possível ajustar as correntes de referência e o tempo de integração à potência luminosa, mas este processo precisa ser feito toda vez que a luminância média da imagem projetada for modificada.

Para resolver este problema é possível gerar uma única corrente de referência dentro do circuito ($I_{ref}$), escalá-la para calcular todas as outras correntes necessárias (limiares e a corrente máxima esperada) e fazer com que o próprio circuito de leitura regule o tempo de integração.

Uma forma possível é através da comparação da corrente de saída de todos os circuitos de CDS ($I_{y,i}(n)$) com a corrente máxima esperada. Quando $I_{y,i}(n)$ ultrapassar a corrente máxima esperada, o tempo de integração deve cessar, fazendo com que o maior valor do circuito seja sempre igual ao valor máximo esperado. Desta forma, a faixa dinâmica do circuito sempre é preservada sem ajustes externos.

### 6.1.2 Melhoria da Precisão dos Circuitos

Como visto no Capítulo 4, os circuitos projetados levaram em conta o compromisso entre área ocupada e precisão. No entanto, é possível investigar mais profundamente quais partes do circuito afetam a compressão de forma mais expressiva e portanto necessitam de maior precisão e vice-versa.

Para aumentar a precisão dos circuitos, que são basicamente formados por espelhos de corrente, podem-se utilizar transistores de dimensões maiores (principalmente o comprimento $L$) ou espelhos do tipo *cascode*, que aumentam consideravelmente a impedância de saída dos espelhos de corrente.

### 6.1.3 Parâmetros Variáveis

Neste trabalho os parâmetros utilizados para implementar o esquema de compressão (limiares e elementos das matrizes de produto interno) foram fixados no *hardware*. No entanto é possível tornar estes parâmetros variáveis.

Para variar os limiares, bastaria tornar as correntes que correspondem aos limiares controláveis por um circuito digital externo. É preciso apenas que estas novas correntes sejam múltiplos de uma única corrente de referência. Esta modificação não geraria mudanças no *hardware* dos blocos, podendo ser integralmente realizada fora da matriz de pixels.

Já para tornar as matrizes $\mathbf{H}$ e $\mathbf{W}$ variáveis, bastaria mudar a relação entre transistores de entrada e saída. Isto pode ser feito através do chaveamento dos transistores. No entanto esta solução exige mais transistores no circuito dos blocos, o que poderia aumentar consideravelmente a área ocupada pela matriz de pixels.

### 6.1.4 Outros Esquemas de Compressão

No Capítulo 2 apresentamos um método de compressão baseado em DPCM, PCA e VQ. Estas técnicas foram sugeridas em [15] pois a complexidade do circuito que as implementa é pequena. Esta complexidade reduzida é importante quando realizamos todas as funções necessárias dentro dos blocos $4 \times 4$ de forma a tornar todo o processamento paralelo.

Se considerarmos fazer parte do processamento fora da matriz de pixels, poderemos utilizar circuitos mais complexos ao custo de uma velocidade de processamento reduzida pela perda do paralelismo. Nesta hipótese, poderiam ser utilizadas topologias que possibilitam até mesmo realizar a busca completa ([15]), a qual as limitações do circuito de VQ apresentado aqui não permitem. Poderia ser usada, por exemplo, a topologia *Winner Takes All* [28], na qual é possível determinar o centróide mais próximo de $\mathbf{x}(n)$.

# Referências Bibliográficas


[1] LINÁN-CEMBRANO, G., RODRÍGUEZ-VÁZQUEZ, A., CARMONA-GALÁN, R., et al. "A 1000 FPS at 128×128 Vision Processor with 8-Bit Digitized I/O", *IEEE J. Solid-State Circuits*, v. 39, n. 7, Julho de 2004.

[2] LITWILLER, D. "CMOS vs. CCD: Maturing Technologies, Maturing Markets", *Photonics Spectra*, Agosto de 2005.

[3] FOSSUM, E. R. "Active Pixel Sensors: Are CCD's Dinosaurs?" *IEEE Trans. on Electron Devices*, v. 41, n. 3, pp. 452–453, Março de 1994.

[4] LEÓN-SALAS, W. D., BALKIR, S., SAYOOD, K., et al. "A CMOS Imager With Focal Plane Compression using Predictive Coding", *IEEE J. Solid-State Circuits*, v. 42, n. 11, pp. 2555–2572, Novembro de 2007.

[5] LIÑÁN, G. *Diseño de Chips Programables de Señal Mixta con Bajo Consumo de Potencia para Sistemas de Vision en Tiempo Real*. Tese de Doutorado, University of Seville, Spain, Junho de 2002.

[6] MASSARI, N., GOTTARDI, M. "A 100 dB Dynamic-Range CMOS Vision Sensor with Programmable Image Processing and Global Feature Extraction", *IEEE J. Solid-State Circuits*, v. 42, n. 3, pp. 647–657, Março de 2007.

[7] KITCHEN, A., BERMAK, A., BOUZERDOUM, A. "A Digital Pixel Sensor Array with Programmable Dynamic Range", *IEEE Trans. Electron Devices*, v. 52, n. 12, pp. 2591–2601, Dezembro 2005.

[8] LIN, Z., HOFFMAN, M. H., SCHEMM, N., et al. "A CMOS Image Sensor for Multi-level Focal Plane Image Decomposition", *IEEE Trans. Circuits and Systems I: Regular Papers*, v. 55, n. 9, pp. 2561–2572, Outubro de 2008.

[9] CHI, Y. M., ABBAS, A., CHAKRABARTTY, S., et al. "An Active Pixel CMOS Separable Transform Image Sensor". In: *Proc. IEEE Int. Symp. Circuits and Systems*, pp. 1281–1284, Taipei, Taiwan, Maio de 2009.

[10] NILCHI, A., AZIZ, J., GENOV, R. "CMOS Image Compression Sensor with Algorithmically-multiplying ADCs". In: *Proc. IEEE Int. Symp. Circuits and Systems*, pp. 1497–1500, Taipei, Taiwan, Maio de 2009.

[11] ZHANG, M., BERMAK, A. "Architecture of a Digital Pixel Sensor Array using 1-bit Hilbert Predictive Coding". In: *Proc. IEEE Int. Symp. Circuits and Systems*, pp. 1501–1504, Taipei, Taiwan, Maio de 2009.

[12] ARTYOMOV, E., YADID-PECHT, O. "Adaptive Multiple-resolution CMOS Active Pixel Sensor", *IEEE Trans. Circuits and Systems I: Regular Papers*, v. 53, n. 10, pp. 2178–2186, Outubro de 2006.

[13] GOMES, J. G. R. C., MITRA, S. K. "A Comparative Study of the Complexities of Neural Network Based Focal-Plane Image Compression Schemes", *IEICE Trans. Fundamentals of Electronics, Communications and Computer Sciences*, v. J88-A, n. 11, pp. 1185–1196, Novembro de 2005.

[14] HAAS, H. L., GOMES, J. G. R. C., PETRAGLIA, A. "Analog Inner Product Operations for Image Compression in 0.35$\mu$m CMOS", *Analog Integrated Circuits and Signal Processing*, v. 57, pp. 141–150, 2008.

[15] GOMES, J. G. R. C. *Mixed-Signal Multilayer Perceptron Implementation of Low-Complexity Vector Quantizers for Image Compression*. Tese de Doutorado, University of California at Santa Barbara, Setembro de 2004.

[16] HUFFMAN, D. A. "A Method for the Construction of Minimum-Redundancy Codes". In: *Proceedings of the IRE*, pp. 1098–1102, Setembro de 1952.

[17] GERSHO, A., GRAY, R. M. *Vector Quantization and Signal Compression*. New York, USA, Ed. Springer-Verlag, New York, 1992. ISBN: 0792391810.

[18] MALVAR, H., HALLAPURO, A., KARCZEWICZ, M., et al. "Low-Complexity Transform and Quantization in H.264/AVC", *IEEE Transactions on Circuits and Systems for Video Technology*, v. 13, pp. 598–603, Julho de 2003.

[19] BOSE, N. K., GARGA, A. K. "Neural Network Design using Voronoi Diagrams", *IEEE Trans. Neural Networks*, v. 4, n. 5, pp. 778–787, Setembro de 1993.

[20] HAAS, H. L. *Análise da Sensibilidade de Compressão de Imagens no Plano Focal de Câmeras CMOS*. Universidade Federal do Rio de Janeiro, Projeto Final de Curso, Dezembro de 2008.

[21] OHTA, J. *Smart CMOS Image Sensors and Applications*. Primeira ed. Florida, USA, CRC Press, 2007.

[22] RAZAVI, B. *Design of Analog CMOS Integrated Circuits*. New York, USA, McGraw-Hill, 2000. ISBN: 0072380322.

[23] SAINT, C., SAINT, J. *IC Mask Design: Essential Layout Techniques*. New York, USA, McGraw-Hill, 2002. ISBN: 0071500936.

[24] PHILIPP, R. M., ORR, D., GRUEV, V., et al. "Linear Current-Mode Active Pixel Sensor", *IEEE J. Solid-State Circuits*, v. 42, n. 11, pp. 2482–2491, Novembro de 2007.

[25] ZIMMERMANN, H. K. *Integrated Silicon Optoelectronics*. Berlin, Germany, Springer-Verlag, 2000. ISBN: 3540666621.

[26] MEHTA, S., ETIENNE-CUMMINGS, R. "A Simplified Normal Optical Flow Measurement CMOS Camera", *IEEE Trans. Circuits and Systems I: Regular Papers*, v. 53, n. 6, pp. 1223–1234, Junho de 2006.

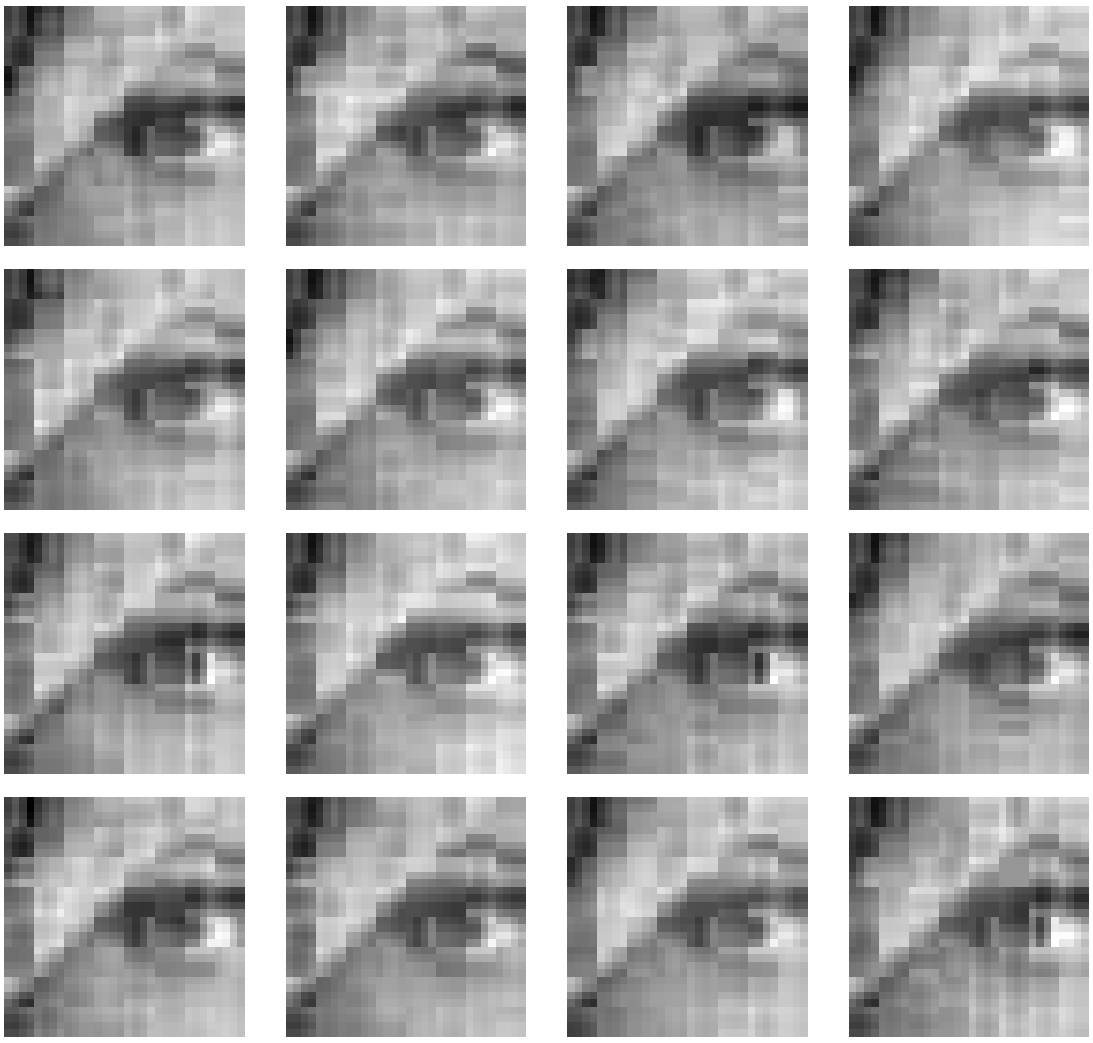
[27] OLIVEIRA, F. D. V. R. *Circuito Integrado para Compressão de Imagens no Plano Focal utilizando Quantização Vetorial e DPCM*. Universidade Federal do Rio de Janeiro, Projeto Final de Curso, Janeiro de 2012.

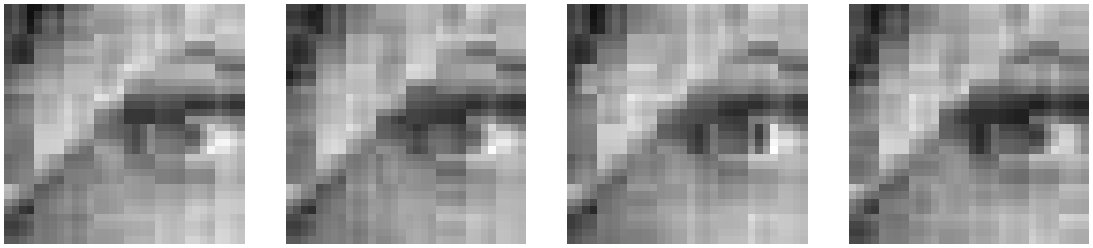
[28] LAZZARO, J., RYCKEBUSCH, S., MAHOWALD, M. A., et al. "Winner-take-all Networks of O(n) Complexity", *Advances in neural information processing systems*, v. 2, pp. 703–711, 1989.

# Apêndice A

# Simulações de Monte Carlo

## A.1 Sem circuito de Leitura e sem Correção de DPCM

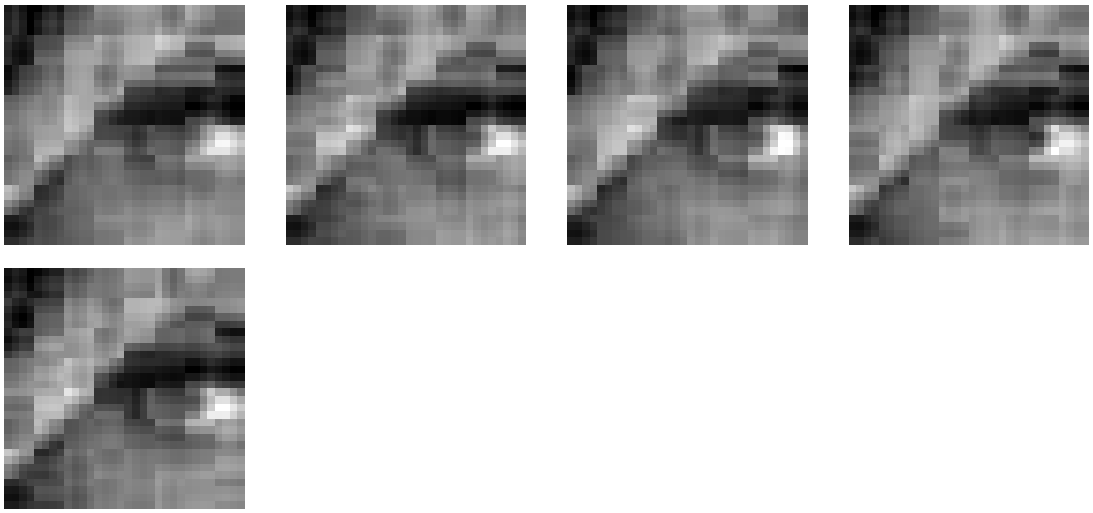




## A.2 Sem circuito de Leitura e com Correção de DPCM

## A.3 Com circuito de Leitura e sem Correção de DPCM

## A.4 Com circuito de Leitura e com Correção de DPCM



# Apêndice B

# Códigos Utilizados

## B.1 Código para Codificação e Decodificação de Imagens

O código a seguir, escrito em linguagem Matlab®, foi utilizado para gerar as imagens, é capaz de codificar e decodificar diretamente de uma figura ou apenas reconstruir uma imagem a partir dos dados da simulação.

Decoder.m

```
% Hugo de Lemos Haas - 29 / 09 / 2009 -  hugohaas@pads.
   ufrj.br
% Decoder.m
% Encoder-Decoder code
% This code can be used to code/decode an image
% Partly copied from other sources
% options: 1-ideal 2-nonideal 3-cadence 4-worse 5-
   cadence with correction

% Code begin

function [Dist PSNR Im Y ] = Decoder(image,option,file);
%
load (image);

G=rgb2gray(F);

% F is a N1xN2 color pixel image
% G is a N1xN2 b&w pixel image
```

```
% 4x4 to 1x16 - divide the image in 4x4 blocks
% X is 16x(N1.N2)/16

i=1;
X=[];
while (i<size(G,1));
    k=1;
    while (k<size(G,2));
        X=[X [G(i,k:k+3) G(i+1,k:k+3) G(i+2,k:k+3) G(i+3,
   k:k+3)]'];
        k=k+4;
    end;
    i=i+4;
end;

% X from 0 to 1
X=double(X)/256;

% P is the principal components of each block
% P is 4x(N1.N2)/16

%%%%%%%%%%%%%%%%%%%
%VQ
%%%%%%%%%%%%%%%%%%%
H=[  2     1     -1     -2      2      1     -1     -2      2
     1     -1     -2      2      1     -1     -2;
     2      2      2      2      1      1      1      1     -1
    -1     -1     -1     -2     -2     -2     -2;
     1     -1     -1      1      1     -1     -1      1      1
    -1     -1      1      1     -1     -1      1;
     1      1      1      1     -1     -1     -1     -1     -1
    -1     -1     -1      1      1      1      1];

 % This option only sets for VQ (changes in H, W and T)

if option==1
    H(1:2,:)=H(1:2,:)/norm(H(1,:));
    H(3:4,:)=H(3:4,:)/norm(H(3,:));
else
```

```
    H(1:2,:)=H(1:2,:)/4/1.5811;
    H(3:4,:)=H(3:4,:)/4;
end;

% P is normalized in code differently of implementation (
   yet to be done)
P=H*X;

% Power Adjustment
P(1:2,:)=P(1:2,:)*1.5811/2;

% S is the binary output of the P signals
S=zeros (4,size(P,2));
for i=1:size(P,2)
    S(1,i)=(P(1,i)>=0);
    S(2,i)=(P(2,i)>=0);
    S(3,i)=(P(3,i)>=0);
    S(4,i)=(P(4,i)>=0);
end;

% VQ parameters
W  =   [ -0.41998907763678  -0.13141444169639
   -0.71812032458305   0.53911280693391 ;  ...
          0.23365707958990  -0.42692361948093
   0.48628829725329   0.72571639380775 ;  ...
          0.79742076822793   0.41786811040985
   -0.36661165734511   0.23473869174029 ;  ...
         -0.36487485830363   0.79110853234270
   0.33678299255441   0.35719860548241 ];

if option==1 % if ideal...
    T =   [ 0.03052458445504   0.07260194033633
   0.12206100977425   0.18906590010944    ...
            0.28609473925358   0.40069465657507
   0.59309040792509  -0.15117969689972    ...
           -0.01251579766665   0.10601821032510
   0.03236954669894   0.00653912954945 ];
else % or if anything else
    W = (round (W*2))/2;
```

```
    T = [0 0.05 0.1 0.2 0.3 0.4 0.6 -0.15 0 0.1 0.05 0];
end;

clear F; % F will be reused

F = fliplr(W)'*abs(P);

% Th is the thermometer of the VQ
Th = zeros (12,size(F,2));
for i=1:size(F,2)
    Th(1,i)=(F(1,i)-T(1))>=0;
    Th(2,i)=(F(1,i)-T(2))>=0;
    Th(3,i)=(F(1,i)-T(3))>=0;
    Th(4,i)=(F(1,i)-T(4))>=0;
    Th(5,i)=(F(1,i)-T(5))>=0;
    Th(6,i)=(F(1,i)-T(6))>=0;
    Th(7,i)=(F(1,i)-T(7))>=0;
    Th(8,i)=(F(2,i)-T(8))>=0;
    Th(9,i)=(F(2,i)-T(9))>=0;
    Th(10,i)=(F(2,i)-T(10))>=0;
    Th(11,i)=(F(3,i)-T(11))>=0;
    Th(12,i)=(F(4,i)-T(12))>=0;
end;


% B is the binary output matching the matlab
B = fliplr(dec2bin(sum(Th(1:7,:),1),3)=='1')';
B = [B; fliplr(dec2bin(sum(Th(8:10,:),1),2)=='1')'];
B = [B; Th(11,:); Th(12,:)];

% ivq is the index to the codebook
ivq = fliplr(2.^((1:7)-1))*B+1;

% BC is the binary output matching the cadence (fix
   transposes)
BM = mat2cad(B')';


%%%%%%%%%%%%%%%%%%%
```

```
%DPCM
%%%%%%%%%%%%%%%%%%

T= [0.0125    0.0375    0.0750    0.1250    0.1875
   0.2750    0.4000]';
C= [0.00625    0.0250    0.05625    0.1000    0.1500
   0.2250    0.3250    0.46875]';


% Mu is the block mean
Mu=zeros(1,size(X,2));
Mu=mean(X,1);

% Muhat is the reconstructed block mean
MuHat=zeros(1,size(X,2)+1);

% D is the DPCM output
D=zeros(4,size(X,2));
index = zeros (1,size(X,2));

% DPCM as done in the encoder - 0 at start of each line
for i=1:size(X,2);

    if mod(i-1,size(G,2)/4)==0 % if is the first of the
   line...
        Dif=Mu(i);
    else
        Dif=Mu(i)-MuHat(i);
    end;

    D(1,i)=(Dif>=0);

    Dif=abs(Dif);

    index(i) = sum(Dif>T)+1;

    D(2:4,i)=Term2Gray(index(i));

    if mod(i-1,size(G,2)/4)==0
```

```
        MuHat(i+1)= C(index(i))*(D(1,i)*2-1);
    else
        MuHat(i+1)= MuHat(i)+C(index(i))*(D(1,i)*2-1);
    end;
end;

%%%%%%%%%%%%%%%%%%%
% Reconstruction
%%%%%%%%%%%%%%%%%%%

%saving matlab outputs
SM=S;
DM=D;
ivqM=ivq;
MuHatM=MuHat;

% using cadence outputs
if (option==3)||(option==5) % if using electric
   simulation output

    [B1 S1 D1 DC1]=csv2bits(file,64,8);

    ivq=cad2index(B1); % VQ
    S=S1; % VQ signal

    %     DPCM
    l=1; %
    L=size(G,2)/4; %total number of lines.
    for i=1:size(X,2);
        index(i)=Gray2Term(D1(2:4,i)'); % From gray to
   index

        if mod(i-1,size(G,2)/4)==0
            MuHat(i+1)= C(index(i))*(D1(1,i)*2-1);
        else
            MuHat(i+1)= MuHat(i)+C(index(i))*(D1(1,i)
   *2-1);
        end;
```

```
        % If using DPCM correction
        if (mod(i,L)==0)&&(option==5)
            Error= MuHat(i)+C(Gray2Term(DC1(2:4,l,1)'))*(
   DC1(1,l,1)*2-1);
            Error= Error+C(Gray2Term(DC1(2:4,l,2)'))*(DC1
   (1,l,2)*2-1);
            Error= Error+C(Gray2Term(DC1(2:4,l,3)'))*(DC1
   (1,l,3)*2-1);
%             MuHatC((l-1)*L+1:(l-1)*L+L)=MuHat((l-1)*L
   +1:(l-1)*L+L);
            for k=1:L
                MuHat(i-L+k+1)=MuHat(i-L+k+1)-(Error/(L
   +4))*k;
            end;

            l=l+1;
        end;
        %%%%%%%%%%

    end;
end;

% Im is the reconstructed image from the DPCM
Im= zeros (size(G));
% DPCM - just rearrange MuHat
for i=1:4:size(Im,1)
    for k=1:4:size(Im,2)
        Im(i:i+3,k:k+3) = MuHat(((i-1)*size(Im,2)/4+(k-1)
   )/4+2);
    end;
end;


% VQ
% C is the VQ codebook
load C;% From KLVQExperiment2 - NonIdeal1.m (LMLP2 #15)

% VQ block reconstruction
```

```
% Phat is the reconstructed principal component vector
Phat=zeros(size(F));
for i=1:size(ivq,2)
    Phat(:,i)=C(:,ivq(i));
end;

if option==4 % This is the most probable centroid.
    CT=[ 0.03197807619525  0.03549316502816
   0.01935453218390  0.02202707650576]';
    CT=[0 0 0 0]';
    Phat=repmat(CT,1,size(Phat,2));
end;

% Principal Component signal reconstruction
for i=1:size(Phat,1)
    Phat(i,:)= Phat(i,:).*(S(i,:)*2-1);
end;

% Reverse Power Adjustment
Phat(1:2,:)=Phat(1:2,:)/1.5811*2;

% Xhat is the image reconstructed minus the mean of each
   block

% Just applying the transpose of H
Xhat=H'*Phat;

% Y is the VQ component of the image, just Xhat
   rearranged in the image field
Y=zeros(size(Im));
for i=1:size(X,2);
    k=floor((i-1)/(size(G,2)/4));
    j=i-k*(size(G,2)/4)-1;
    Y((k*4)+1:(k*4)+4,(j*4)+1:(j*4)+4)= [Xhat(1:4,i) Xhat
   (5:8,i) Xhat(9:12,i) Xhat(13:16,i)]';
end;

% Final image is Im+Y (DPCM + VQ)
% figure;imshow(Im+Y);
```

```
% Only DPCM
% figure;imshow(Im)

PSNR=10*log10(((1-1/256)^2)/mean(mean((Im+Y-double(G)
   /255).^2)));
Dist = mean(sum((abs(P)-abs(Phat)).^2,1));

[PSNR Dist];
```

## B.2 Código para Simulação do Circuito

Os códigos fonte a seguir, escritos em linguagem C++, foram usados para gerar as *netlists* e simulá-las. O primeiro programa (`createInput.cpp`) gera, a partir de um *netlist* feito no ambiente do software Cadence®, um novo *netlist* com uma linha de blocos.

createInput.cpp

```
// Hugo de Lemos Haas - hugohaas@pads.ufrj.br
// Used to generate a test.scs for Spectre

#include <iostream>
#include <fstream>
#include <string>
using namespace std;

int main (int argc, char ** argv)
{
        ifstream ifile;
        ofstream ofile;
        ifstream input;
        string s;
        int n,i,j;
        char saux[256];

        // Input number test
        if (argc < 3)
        {
```

```
            cout<< argv[0] << " <netlist file name> <
block number> <start block>"<< endl;
            return (0);
      }

      // Open base netlist. From this file will be
created the new netlists changing only the desired
parameters
      ifile.open(argv[1],ios::in);
      if (!ifile.is_open())
      {
            cout<< "file"<< argv[1] <<" not opened"
<< endl;
            return(0);
      }

      // Creates a new file to be used as the netlist
      ofile.open("teste.scs");
      if (!ofile.is_open())
      {
            cout<< "teste.scs not opened" << endl;
            return(0);
      }

      // Opens the input file for the photocurrents
      input.open("lena32x32.txt");
      if ( !input.is_open() )
      {
            cout<< "lena32x32.txt not opened" << endl
;
            return(0);
      }

      s="";

      // Skip to the beginning of the circuit (after
the subcircuits)
      while (ifile.getline(saux,256) && (s!="// Cell
name: teste"))
```

```
        {
                s=saux;
                ofile << s << endl;
        }
        s=saux;

        // Write the sources
        ofile<<"// View name: schematic"<<endl;
        ofile<<"V0 (vdd! 0) vsource dc=3.3 type=dc"<<endl
;
        ofile<<"V1 (Reset 0) vsource type=pulse val0=3.3
val1=0 period=tint+tr+td delay=td+1u rise=tf fall=tf
width=tint"<<endl;
        ofile<<"V2 (P1 0) vsource type=pulse val0=3.3
val1=0 period=tint+tr+td delay=td rise=tf fall=tf
width=tint+tr"<<endl;
        ofile<<"V3 (\\#P1 0) vsource type=pulse val0=0
val1=3.3 period=tint+tr+td delay=td rise=tf fall=tf
width=tint+tr"<<endl;
        ofile<<"V4 (P2 0) vsource type=pulse val0=3.3
val1=0 period=tint+tr+td delay=td+tint rise=tf fall=tf
 width=tr"<<endl;
        ofile<<"V5 (\\#P2 0) vsource type=pulse val0=0
val1=3.3 period=tint+tr+td delay=td+tint rise=tf fall=
tf width=tr"<<endl<<endl;

        // Number of blocks per line, read from the
parameter in the command line
        n=atoi(argv[2]);

        // Goes to the first block to be simulated -
command line parameter
        for (i=16;i<atoi(argv[3])*16;i=i+1)
        {
                input.getline(saux,256);
        }

        // Start writing the blocks and the connections
        for (i=atoi(argv[3]); i<=n+atoi(argv[3])-1; i++)
```

```
      {

                for (j=1;j<=16;j++)
                {
                        input>>s;
                        ofile <<"Is"<<j<<"_"<<i<<" (Iin"
<<j<<"_"<<i<<" 0) isource dc="<<s<<" type=dc" <<endl;
                }

                if (i==atoi(argv[3]))
                {
                        ofile <<"I_"<<i<<" (\\#P1 \\#P2
Iin1_"<<i<<" Iin10_"<<i<<" Iin11_"<<i<<" Iin12_"<<i<<"
 Iin13_"<<i<<" Iin14_"<<i<<" Iin15_"<<i<<" Iin16_"<<i;
                        ofile<< " Iin2_"<<i<<" Iin3_"<<i
<<" Iin4_"<<i<<" Iin5_"<<i<<" Iin6_"<<i<<" Iin7_"<<i<<
" Iin8_"<<i<<" Iin9_"<<i<<" P1 P2 Reset Vactual_"<<i;
                        ofile<<" 0";
                        for (j=1;j<=7;j++)
                        {
                                ofile <<" B"<<j<<"_"<<i;
                        }
                        for (j=1;j<=4;j++)
                        {
                                ofile <<" D"<<j<<"_"<<i;
                        }
                        for (j=1;j<=4;j++)
                        {
                                ofile <<" S"<<j<<"_"<<i;
                        }
                        ofile<<") bloco"<<endl<<endl;
                }
                else
                {
                        ofile <<"I_"<<i<<" (\\#P1 \\#P2
Iin1_"<<i<<" Iin10_"<<i<<" Iin11_"<<i<<" Iin12_"<<i<<"
 Iin13_"<<i<<" Iin14_"<<i<<" Iin15_"<<i<<" Iin16_"<<i;
                        ofile<< " Iin2_"<<i<<" Iin3_"<<i
<<" Iin4_"<<i<<" Iin5_"<<i<<" Iin6_"<<i<<" Iin7_"<<i<<
```

```
" Iin8_"<<i<<" Iin9_"<<i<<" P1 P2 Reset Vactual_"<<i;
                        ofile<<" Vactual_"<<(i-1);
                        for (j=1;j<=7;j++)
                        {
                                ofile <<" B"<<j<<"_"<<i;
                        }
                        for (j=1;j<=4;j++)
                        {
                                ofile <<" D"<<j<<"_"<<i;
                        }
                        for (j=1;j<=4;j++)
                        {
                                ofile <<" S"<<j<<"_"<<i;
                        }
                        ofile<<") bloco"<<endl<<endl;
                }
        }

        // DPCM correction circuits
        ofile<<"IsDC1 (IinDC1 0) isource dc=0 type=dc"<<
endl;
        ofile<<"I_D1(IinDC1 VactualDC1 Vactual_"<<(i-1)<<
" DC1_1 DC2_1 DC3_1 DC4_1) dpcmComplete"<<endl<<endl;

        ofile<<"IsDC2 (IinDC2 0) isource dc=0 type=dc"<<
endl;
        ofile<<"I_D2(IinDC2 VactualDC2 VactualDC1 DC1_2
DC2_2 DC3_2 DC4_2) dpcmComplete"<<endl<<endl;

        ofile<<"IsDC3 (IinDC3 0) isource dc=0 type=dc"<<
endl;
        ofile<<"I_D3(IinDC3 VactualDC3 VactualDC2 DC1_3
DC2_3 DC3_3 DC4_3) dpcmComplete"<<endl<<endl;

        while (ifile.getline(saux,256) && (s!="
simulatorOptions options reltol=100e-6 vabstol=1e-6
iabstol=1e-12 temp=27 \\"))
        {
                s=saux;
```

```
        }
        ofile << s << endl;
        ofile << saux << endl;

        // while (ifile.getline(saux,256)&& (s!="subckts
info what=subckts  where=rawfile"))
        while (ifile.getline(saux,256)&& (s!="}")))
        {
                s=saux;
                ofile << s<<endl;
        }

        // Save the binary outputs
        s=saux;
        ofile<<"save";
        for (i=atoi(argv[3]); i<= n+atoi(argv[3])-1;i++)
        {
                for (j=1;j<=7;j++)
                {
                        ofile <<" B"<<j<<"_"<<i;
                }
                for (j=1;j<=4;j++)
                {
                        ofile <<" D"<<j<<"_"<<i;
                }
                for (j=1;j<=4;j++)
                {
                        ofile <<" S"<<j<<"_"<<i;
                }
        }

        // Save the outputs from the DPCM correction
        ofile<<" DC1_1 DC2_1 DC3_1 DC4_1 DC1_2 DC2_2
DC3_2 DC4_2 DC1_3 DC2_3 DC3_3 DC4_3";
        ofile<<endl;
        ofile<<"saveOptions options save=selected"<<endl;
        ifile.close();
        ofile.close();
        input.close();
```

```
        return (0);
}
```

Para criar um arquivo de lote (*batch*) e simular todas as linhas de uma imagem foi utilizado o programa `main.cpp` mostrado a seguir. Como saída, este programa cria um arquivo "Run" que cria uma *netlist* para cada linha e a simula em seguida.

main.cpp

```
//Creates Run batch file for simulation
#include <iostream>
#include <fstream>
#include <string>
#include <sstream>
using namespace std;

int main (int argc, char ** argv)
{
        int i, n;
        string s,fileName;
        stringstream ss;
        ifstream outputFile;
        ofstream runfile;

        runfile.open("Run");
        runfile<<"#!/bin/bash"<<endl;

        //number of blocks in each direction
        n=8;

        for (i=1; i<=n*n; i=i+n)
        {
                s="";
                ss.str("");
                ss<<i;
                s+="./create input.scs 8 "; // Number
   after input.scs must be n
                s+=ss.str();
                s+=" \n";
                runfile<<s<<endl;
```

```
                s="";
                s+="spectre -env artist5.1.0 +escchars +
   log ./psf/spectre.out -format psfascii -raw ./psf";
                s+=ss.str();
                s+=" +lqtimeout 900 +param /cds/Opus/user
   /a370/spectre/ams_range.lmts teste.scs\n";

                runfile<<s<<endl;
        }

        return(0);
}
```

Para gerar um arquivo que possa ser usado no Matlab® a partir da saída das simulações foi utilizado o programa `catFiles.cpp`, mostrado a seguir.

catFiles.cpp

```
// Creates input file for matlab decoder
#include <iostream>
#include <fstream>
#include <string>
#include <sstream>
#include <iomanip>
using namespace std;

int main (int argc, char ** argv)
{
        ofstream ofile;
        ifstream ifile;
        stringstream filename,ss,ssaux;
        string str,s,s1,s2;
        char saux [256];
        char *token = NULL;

        int i,k;

        // Input number test
        if (argc < 4)
        {
```

```
            cout<< argv[0] << " <out file name> <
block number> <start block> <block per line> <MC runs>
"<< endl;
            return (0);
      }

      ofile.open(argv[1]);
      if (!ofile.is_open())
      {
            cout<<argv[1]<< " not opened" << endl;
            return(0);
      }

      for (i=atoi(argv[3]);i<atoi(argv[2]);i=i+atoi(
argv[4]))
      {
            filename.str("");
            filename<<"./psf"<<i<<"/tran.tran";

            str=filename.str();
            ifile.open(str.c_str());
            ifile.seekg(ios::beg);

            if (!ifile.is_open())
            {
                  cout << "File " << filename.str()
 << " can't be opened!" << endl;
                  return (0);
            }

            // Skip to the end of integration time
            s="";
            while (ifile.getline(saux,256) && (s!="\"
time\" 0.000120000"))
            {
                  s=saux;
            }

            token = strtok(saux," ");
```

```
                ofile<<token<<",";
                s2=token;
                token= strtok(NULL," ");
                ofile<<token<<endl;

                ss.str("");
                ss<<"\"S4_"<<i-1+atoi(argv[4])<<"\"";
                s1=ss.str();

                while (ifile.getline(saux,256) && (s1!=s2
))
                {
                        token = strtok(saux," ");
                        ofile<<token<<",";
                        s2=token;
                        token= strtok(NULL," ");
                        ofile<<token<<endl;
                }
                ifile.close();
        }
        ofile.close();

        for (k=1;k<=atoi(argv[5]);k=k+1)
        {
                filename.str("");
                filename<<"MC"<<k<<"_"<<argv[1];
                str=filename.str();
                ofile.open(str.c_str());

                for (i=atoi(argv[3]);i<atoi(argv[2]);i=i+
atoi(argv[4]))
                {
                        filename.str("");
                        filename<<"./psf"<<i<<"/mc1-"<<
setw(3)<<setfill('0')<<k<<"_tran.tran";

                        str=filename.str();
                        ifile.open(str.c_str());
                        ifile.seekg(ios::beg);
```

```
                        if (!ifile.is_open())
                        {
                                cout << "File " <<
filename.str() << " can't be opened!" << endl;
                                return (0);
                        }
                        s="";
                        //while (ifile.getline(saux,256)
&& (s!="VALUE"))
                        while (ifile.getline(saux,256) &&
 (s!="\"time\" 0.000120000"))
                        {
                                s=saux;
                        }

                        token = strtok(saux," ");
                        ofile<<token<<",";
                        s2=token;
                        token= strtok(NULL," ");
                        ofile<<token<<endl;


                        ss.str("");
                        ss<<"\"S4_"<<i-1+atoi(argv[4])<<"
\"";
                        s1=ss.str();

                        while (ifile.getline(saux,256) &&
 (s1!=s2))
                        {
                                token = strtok(saux," ");
                                ofile<<token<<",";
                                s2=token;
                                token= strtok(NULL," ");
                                ofile<<token<<endl;
                        }
                        ifile.close();
                }
```

```
                ofile.close();
        }

        return(0);
}
```