PIXEL PARA *HALFTONING* NO PLANO FOCAL EM SENSORES DE IMAGEM CMOS

Saulo Avila Nunes

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.

Orientador: José Gabriel Rodriguez Carneiro Gomes

Rio de Janeiro
Março de 2012

PIXEL PARA *HALFTONING* NO PLANO FOCAL EM SENSORES DE IMAGEM CMOS

Saulo Avila Nunes

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. José Gabriel Rodriguez Carneiro Gomes, Ph.D.

Prof. Antonio Petraglia, Ph.D.

Prof. Estêvão Coelho Teixeira, D.Sc.

RIO DE JANEIRO, RJ – BRASIL
MARÇO DE 2012

Nunes, Saulo Avila

Pixel para *Halftoning* no Plano Focal em Sensores de Imagem CMOS/Saulo Avila Nunes. – Rio de Janeiro: UFRJ/COPPE, 2012.

XIII, 75 p.: il.; 29, 7cm.

Orientador: José Gabriel Rodriguez Carneiro Gomes

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

Referências Bibliográficas: p. 51 – 53.

1. Sensor de Imagem CMOS. 2. *Halftoning*. 3. Processamento no plano focal. I. Gomes, José Gabriel Rodriguez Carneiro. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia Elétrica. III. Título.

Aos meus pais George e Marília

Agradecimentos

Agradeço aos meus amigos que sempre me apoiaram, que não me deixaram desanimar. Em especial aos amigos Hugo Haas, Pedro Coelho e Ricardo Flach com quem tive o prazer de cursar várias disciplinas e que muito me ajudaram durante estes três anos do mestrado, tornando o caminho percorrido um pouco menos árduo.

Agradeço aos professores Joarez Bastos e Fernando Barúqui que sempre me ajudaram e aconselharam quando precisei.

Um agradecimento especial ao professor e orientador José Gabriel, pelo seu incentivo e paciência. Por ter perdido parte do carnaval, se não todo ele, revisando o texto.

E agradeço, claro, à minha família. Que soube entender minha ausência em muitas ocasiões. Que sempre me incentivou e apoiou.

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

PIXEL PARA *HALFTONING* NO PLANO FOCAL EM SENSORES DE IMAGEM CMOS

Saulo Avila Nunes

Março/2012

Orientador: José Gabriel Rodriguez Carneiro Gomes

Programa: Engenharia Elétrica

Neste trabalho é apresentado o projeto de um pixel APS (*active pixel sensor*) utilizando tecnologia CMOS de 0,35 μm da AMS. Circuitos adicionais, inseridos dentro do pixel, processam a imagem capturada pelos elementos foto-sensíveis, fornecendo na saída de cada um dos pixels um sinal binário, que representa a imagem comprimida com perdas a um bit por pixel. A técnica escolhida para processar o sinal consiste no algoritmo de difusão de erros de Floyd-Steinberg, uma técnica de *halftoning* digital muito utilizada em impressoras a jato de tinta e a laser para impressão de imagens. Para validar o circuito proposto, são apresentadas simulações elétricas de um bloco de pixels de dimensão 64×64.

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

A PIXEL FOR FOCAL-PLANE HALFTONING IN CMOS IMAGE SENSORS

Saulo Avila Nunes

March/2012

Advisor: José Gabriel Rodriguez Carneiro Gomes

Department: Electrical Engineering

In this work, we present the design of a CMOS pixel using APS (active pixel sensor) technology. All circuits were designed using AMS CMOS 0.35 μm technology. Additional signal processing hardware is included at the pixel level, so that a binary stream representing the image compressed at one bit per pixel is the only output of the sensor array. The technique chosen for such binary representation is Floyd-Steinberg error diffusion, which is a well-known halftoning algorithm for inkjet and laser image printing. To validate the proposed circuit, we present electrical simulations of a 64×64 pixel block.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Na última década, a utilização de câmeras digitais como parte de sistemas, em diversas áreas, sofreu um grande aumento. Na indústria, por exemplo, são responsáveis por capturar imagens do material transportado por esteiras rolantes permitindo que um sistema externo faça a análise da imagem e identifique o material, dando a ele a destinação correta. Algumas tecnologias de painéis interativos também utilizam câmeras digitais para detectar os movimentos realizados pelo usuário e executar as ações desejadas. Os sistemas de identificação biométricos são outro exemplo. O uso das câmeras digitais nos mais variados setores da sociedade se deve à evolução da tecnologia CCD (*charge coupled device*) que permitiu uma queda brusca dos preços dos sensores nas últimas décadas. A tecnologia CMOS (*complementary metal-oxide semiconductor*), que inicialmente foi deixada de lado para fabricação de sensores de imagem devido, principalmente, a grande quantidade de ruído adicionado à imagem capturada, também apresentou extraordinários avanços nos processos de fabricação na última década. Grande parte destes avanços foi alcançado graças à acirrada disputa travada entre os fabricantes de processadores em busca de circuitos cada vez menores, mais rápidos e com baixo consumo. Essa disputa exigiu que os processos de fabricação fossem qualitativamente aprimorados, o que permitiu que os sensores de imagens utilizando tecnologia CMOS atingissem níveis de qualidade semelhantes aos oferecidos pelos sensores CCD. Os sensores CMOS, no entanto, permitem que circuitos adicionais sejam inseridos junto aos elementos foto-sensíveis, tornando possível processar a imagem capturada ainda dentro do sensor, permitindo que alguns circuitos, ou até mesmo todos os circuitos externos (dependo da aplicação) sejam eliminados. Outra característica presente nos sensores de imagem mais novos é a alta resolução. Nos dias de hoje, é muito comum encontrar sensores com resolução superior a 14 megapixels em câmeras digitais de baixo custo. As imagens capturadas por sensores com grandes resoluções, após a digitalização, necessitam de um espaço grande para serem armazenadas. Um meio de contornar esse problema é realizar a compressão de dados.

Existe hoje uma grande quantidade de algoritmos utilizados para realizar compressão de dados com ou sem perdas. Os algoritmos de compressão se baseiam em informações obtidas de análises estatísticas realizadas sobre os dados que se deseja comprimir, no caso deste trabalho, imagens. Com o auxílio das análises estatísticas, as técnicas de compressão eliminam as redundâncias existentes na imagem. Durante o processo em que as redundâncias são eliminadas, parte da informação pode ser perdida, dependendo do método utilizado. Nas câmeras digitais a compressão é realizada por um processador externo que faz a leitura dos valores analógicos de cada um dos pixels do sensor de imagem, realiza a conversão analógico-digital (A/D) [3] e então aplica algum algoritmo sobre os dados para comprimí-los. Com o avanço da tecnologia CMOS, muitos trabalhos tem sido desenvolvidos ([4]–[11]) em busca de alternativas que permitam realizar a compressão no plano focal, eliminando a necessidade de circuitos externos para essa tarefa. Em [4], é apresentado um imageador CMOS desenvolvido com o objetivo de realizar compressão no plano focal usando codificação preditiva. Em outro trabalho [12], um APS (*active pixel sensor*) operando em modo de corrente é proposto para a compressão (com perdas) sobre a imagem capturada utilizando as técnicas de *differential-pulse code modulation* (DPCM) e quantização vetorial. Um sensor de imagem CCD capaz de processar a imagem capturada utilizando o algoritmo de difusão de erros proposto por Floyd e Steinberg é apresentado em [13]. Todo o circuito é implementado utilizando tecnologia CCD e as operações de soma e multiplicação são realizadas durante os deslocamentos das cargas capturadas no processo de foto-conversão. Em [14] redes neurais CNN-UM são utilizadas para processar imagens utilizando o algoritmo de halftoning. Outro trabalho utilizando redes neurais é apresentado em [15], onde uma rede neural é utilizada para minimizar a distância quadrática entre a imagem analógica capturada e a imagem binária processada pelo algoritmo de *halftoning*, mas a implementação proposta, utilizando circuitos a capacitores chaveados, pode ser proibitiva nos dias atuais. Os sensores de imagem atingiram uma densidade muito grande de pixels, da ordem de dezenas de milhões, reduzindo muito o espaço disponível para inclusão de circuitos de processamento no sensor.

O objetivo deste trabalho é desenvolver um circuito de pixel ativo que implemente o algoritmo de *halftoning* proposto por Floyd e Steinberg, permitindo a compressão, com perdas, sobre a imagem capturada. O projeto do pixel será desenvolvido utilizando circuitos que operam em modo de corrente. Ao processar o sinal em modo de corrente, operações básicas como soma e multiplicação podem ser implementadas utilizando circuitos bastante simples, o que é muito importante quando falamos de sensores CMOS, onde o espaço disponível para inserção de circuitos é extremamente limitado. Para codificar o valor de cada pixel (geralmente representado por oito bits em imagens monocromáticas) em apenas um bit, o circuito de cada um dos pixels

deverá estar conectado a quatro outros pixels. Para verificar o funcionamento do circuito proposto serão realizadas simulações elétricas de uma matriz de pixels de dimensão 64×64. O modo como os erros são propagados pelo algoritmo de Floyd e Steinberg (para pixels localizados abaixo e à esquerda) nos permite prever que as saídas dos pixels localizados no canto superior esquerdo deverão estabilizar primeiro e que quanto mais próximos do canto inferior direito os pixels estiverem, mais tempo necessitarão para atingir a estabilidade. Para cada pixel da matriz, um bit será lido, de forma independente, sem o auxílio de circuitos de leitura, como registradores, que fazem a interface entre o sensor de imagem e circuitos externos. O resultado dessa simulação será comparado a uma imagem de referência gerada pelo algoritmo ideal.

Nos capítulos a seguir, este trabalho é apresentado de forma mais clara e detalhada. Os conceitos teóricos nos quais o trabalho está baseado são apresentados no Capítulo 2. Nele são apresentados alguns dos primeiros algoritmos de *halftoning*, que representavam os tons de cinza desejados modulando o tamanho dos pontos. Também são apresentados algoritmos cujo funcionamento se baseia na alteração da frequência com que os pontos (de tamanho fixo) são impressos, dentre eles o algoritmo de difusão de erros, proposto por Floyd e Steinberg e algumas variações propostas para esse algoritmo ao longo dos anos. No Capítulo 3 é apresentada a forma como o circuito é implementado. Um circuito de leitura em modo de corrente é proposto operando em conjunto com um circuito de CDS (*correlated double sampling* ou amostragem dupla correlacionada). Todos os demais circuitos apresentados no Capítulo 3 implementam o algoritmo de *halftoning*, processando os sinais propagados pelos pixels vizinhos e em conjunto com o sinal obtido através do elemento foto-sensível. No Capítulo 4 são discutidos os resultados obtidos da simulação da matriz 64×64 e as conclusões do trabalho são apresentadas no Capítulo 5. No Apêndice A é apresentado o código-fonte comentado do programa escrito em C++ para geração do *netlist* da matriz 64×64. O Apêndice B traz o resultado de simulações realizadas em uma tentativa de estimar o tempo necessário para que o circuito alcance um estado de estabilidade. E o Apêndice C contém o código desenvolvido no MATLAB com o intuito de verificar a robustez do circuito.

Capítulo 2

Fundamentos Teóricos

O *halftoning* é uma técnica que permite reproduzir cores em tons contínuos através da impressão ou não de pontos. Diferentes tons de cinza, por exemplo, podem ser representados utilizando apenas pontos pretos e "pontos brancos"(ausência de pontos pretos), sendo necessário apenas controlar o espaçamento, o tamanho ou a densidade dos pontos para criar a tonalidade desejada.

O foco deste trabalho é voltado para o processamento de imagens monocromáticas, portanto, daqui em diante, sempre que nos referirmos à técnica de *halftoning*, estaremos nos referindo ao caso monocromático.

2.1 *Halftoning* AM

As primeiras técnicas de *halftoning* desenvolvidas atuam sobre o tamanho dos pontos a fim de criar a tom desejado e, por este motivo, essas técnicas ficaram conhecidas como *halftoning* AM (*Amplitude Modulated*).

Uma das técnicas de *halftoning* AM mais utilizadas é conhecida como *ordered dithering*. Esse método utiliza uma matriz de erros com um padrão de ruído calculado para processar a imagem. Supondo que a matriz de erros tenha dimensão 8×8, a imagem original é também dividida em blocos 8×8 e cada bloco é comparado, elemento a elemento, à matriz de erros, conforme mostrado na Fig. 2.1. Neste caso como a matriz de erros possui 64 posições, o valor de cada um dos pixels da imagem a ser processada deve ser multiplicado por um fator, de tal modo a mapear o valor máximo possível para um pixel (igual a 255) para o valor máximo 64, permitindo que os limiares da matriz de erro possam ser corretamente comparados aos blocos da imagem.

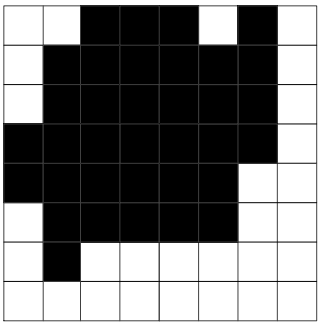
Outra técnica classificada como *halftoning* AM é denominada *patterning*. Neste método, cada pixel da imagem é transformado em uma célula de *halftoning* que é então dividida em uma matriz, de tamanho 4×4, por exemplo. O pixel da imagem é então comparado a uma matriz de limiares, também de dimensão 4×4, e o resultado

15	16	15	22	17	15	15	14
24	19	23	28	28	22	19	15
21	24	35	41	38	37	26	18
25	32	38	45	48	44	38	30
20	27	40	49	48	47	43	41
18	18	36	50	49	48	46	45
20	18	39	50	50	47	48	46
18	24	47	52	49	49	49	47

(a)

3	9	17	27	25	15	71	1
11	29	38	46	44	36	23	5
19	40	52	58	56	50	34	13
31	48	60	64	62	54	42	21
30	47	59	63	61	53	41	20
18	39	51	57	55	49	33	12
10	28	37	45	43	35	22	4
2	8	16	26	24	14	6	0

(b)



(c)

Figura 2.1: Exemplo do algoritmo de *ordered dithering*: (a) matriz de pixels da imagem de entrada, (b) matriz de limiares e (c) resultado da aplicação do algoritmo, utilizando as matrizes (a) e (b).

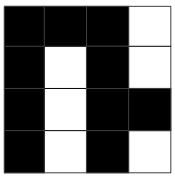
da comparação é atribuído à célula de *halftoning* do pixel. Assim como no caso do método de *ordered dithering*, se o valor do pixel da imagem for maior que o valor do limiar da matriz, a posição do mesmo, na célula de *halftoning*, será preenchida (cor preta), caso contrário, permanecerá vazia. A Fig. 2.2 ilustra este procedimento.

154

(a)

1	129	33	161
65	193	97	225
49	177	17	145
113	241	81	209

(b)



(c)

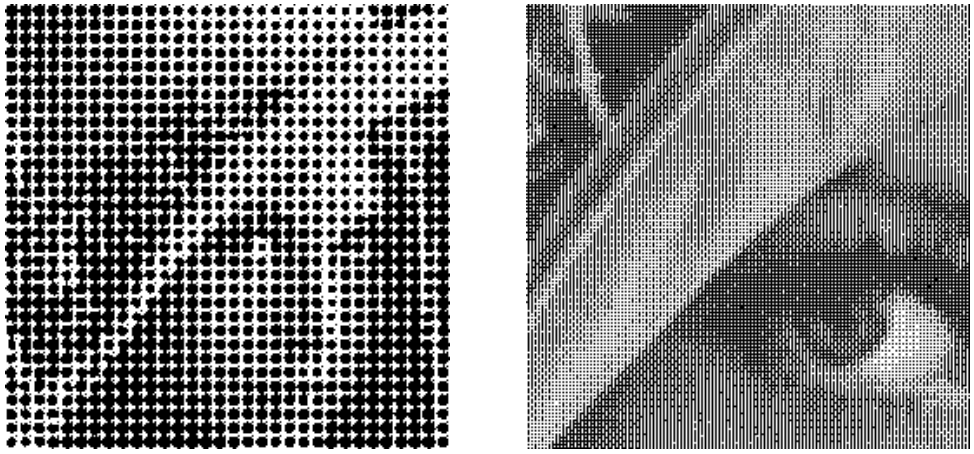
Figura 2.2: Exemplo do algoritmo de *patterning*, codificando o pixel de uma imagem: (a) pixel da imagem de entrada, (b) matriz de limiares e (c) resultado da aplicação do algoritmo.

Essa técnica possui algumas desvantagens que estão relacionadas ao mapeamento de um pixel da imagem em outros dezesseis pixels na imagem processada. Com isso a imagem final terá menor nitidez e maior dimensão do que a imagem original. Outra questão que deve ser observada, com a utilização dessa técnica, consiste no processo de escolha da matriz de limiares, pois, no caso das matrizes 4×4, só é possível atribuir 17 limiares para comparação, um número bem abaixo dos 256 valores possíveis. Por este motivo se faz necessária uma análise a fim de definir os melhores limiares para a matriz de limiares.

A Fig. 2.3 [1] apresenta os resultados obtidos ao processarmos uma imagem através dos dois algoritmos. Através dessas imagens é possível observar a presença de es-

[1]Note que as imagens (a) e (b) possuem a mesmo número de pontos. Como a técnica de *patterning* transforma cada pixel da imagem original em dezesseis pontos, a imagem resultante processada por este algoritmo tem resolução maior do que a imagem original. Por isso um pedaço menor da imagem Lena foi utilizado para exemplificar este algoritmo, fazendo com que durante

truturas indesejadas. Na Fig. 2.3(a) é notória a presença dos blocos gerados pelo algoritmos, fazendo com que a imagem tenha uma grande redução a nível de detalhes. Na Fig. 2.3(b) também são visiveis estruturas indesejadas em locais com transição de tons e também o aspecto quadriculado sobre toda a imagem, devido aos padrões horizontais e verticais criados pelas células de *halftoning* que representam cada pixel.



(a) (b)

Figura 2.3: Processamento da imagem Lena [1] por *halftoning*: (a) imagem processada através do algoritmo de *ordered dithering*; (b) imagem processada através do algoritmo de *patterning*.

2.2 *Halftoning* FM

As técnicas de *halftoning* surgiram devido à necessidade de reprodução de imagens e com o advento das impressoras digitais, que reproduzem imagens através da impressão de pontos isolados, novos algoritmos de *halftoning* foram desenvolvidos com a filosofia de que cada pixel fosse representado por apenas um ponto. Um dos objetivos destes algoritmos consistia na redução das estruturas indesejadas, criadas pela percepção visual de pontos individuais, presentes nas técnicas de *halftoning* AM.

Fazendo com que cada ponto represente um pixel, as diferentes tonalidades de cinza passaram a ser representadas pela frequência com que os pontos pretos e brancos são dispostos na imagem e por isso essas técnicas passaram a ser conhecidas como *halfoning* FM (*frequency modulated*).

Bayer desenvolveu um dos primeiros algoritmos [16] de *halftoning* FM. Assim como as técnicas AM, este algoritmo FM também quantizava cada um dos pixels

a impressão, os pixels pretos e brancos não sejam fundidos em regiões cinza. Ao contrário, pixels pretos e brancos vizinhos devem ser tratados pelo algoritmo de impressão como sendo pixels separados.

0	58	14	54	3	57	13	53
32	16	46	30	35	19	45	29
8	48	4	62	11	51	7	61
40	24	36	20	43	27	39	23
2	56	12	52	1	59	15	55
34	18	44	28	33	17	47	31
10	50	6	60	9	49	5	63
42	26	38	22	41	25	37	21

Figura 2.4: Matriz de erros desenvolvida por Bayer.

independentemente dos pixels vizinhos, utilizando também uma matriz de erros (Fig. 2.4), como feito na técnica de *ordered dithering*. Os limiares dessa matriz são arranjados de forma tão dispersa quanto possível, mas ainda assim as imagens resultantes apresentam um aspecto não-natural, devidos às estruturas periódicas presentes, como mostrado na Fig. 2.5.

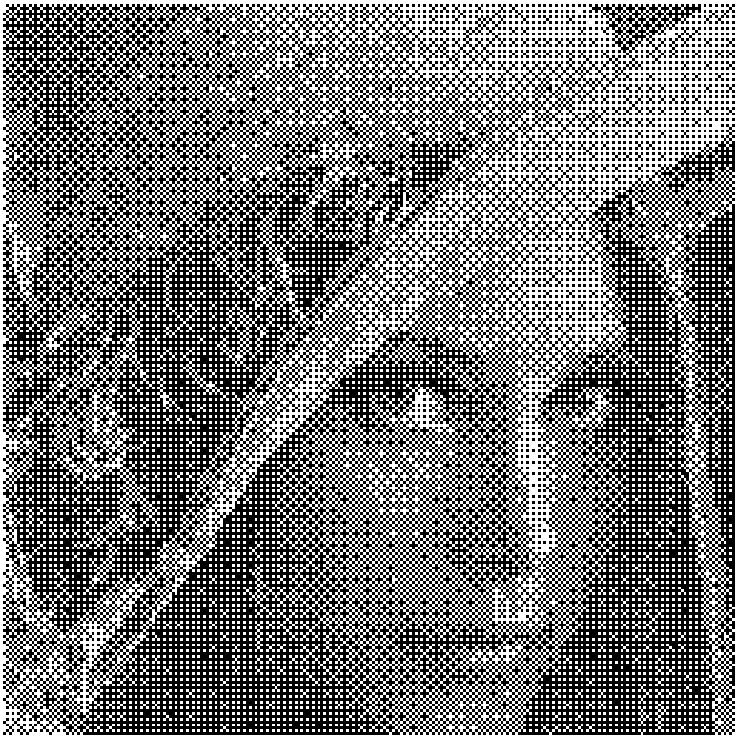


Figura 2.5: Imagem processada utilizando a matriz de erros de Bayer.

A técnica AM de *ordered dithering* (Fig. 2.3(a)) e a técnica FM de Bayer (Fig. 2.5) utilizam algoritmos muito parecidos. A única diferença entre eles está na matriz de erros utilizada por cada algoritmo. A matriz de Bayer (Fig. 2.4), que considera

cada pixel como um ponto, dispersa os limiares consecutivos tanto quanto possível a fim de evitar a formação de estruturas na imagem final enquanto que a matriz de erros para *ordered dithering*, que transforma um conjunto de pixels em um ponto, concentra os níveis mais altos no centro da matriz e os mais baixos nas bordas (Fig. 2.1(b)) criando o efeito de ponto desejado. Diante da presença de estruturas periódicas nas imagens obtidas, utilizando estes algoritmos, outro algoritmo foi criado por Floyd e Steinberg [17]. É chamado de algoritmo de difusão de erros. Apesar do maior custo computacional deste método, ele apresenta imagens com aspecto visual muito superior ao aspecto daquelas fornecidas pelos outros métodos apresentados.

2.3 Difusão de Erros

O algoritmo de difusão de erros é uma técnica adaptativa que, ao contrário dos algoritmos até aqui apresentados, quantiza cada pixel com base também no valor dos pixels vizinhos. Na Fig. 2.6 é apresentado um diagrama de blocos do algoritmo de difusão de erros. O filtro presente no diagrama de blocos é projetado de acordo com uma análise estatística do pixel de entrada e seus vizinhos, contribuindo para que os pixels sejam arranjandos de forma estocástica.

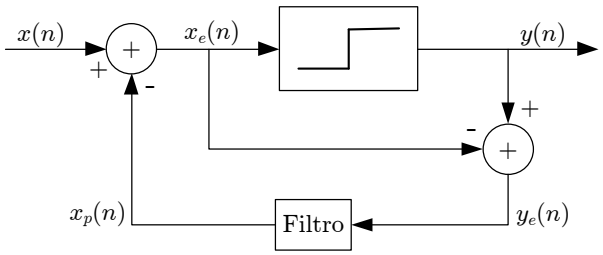


Figura 2.6: Diagrama em bloco do algoritmo de *halftoning*.

A boa aparência de uma imagem, quando processada pelo algoritmo de *halftoning*, pode ser medida através da figura de mérito conhecida como visibilidade, que indica o quão agradável é a imagem para o olho humano. A visibilidade é obtida calculando o MSE (*mean squared error*) entre a imagem em tom-contínuo e a imagem *halftoning* resultante, ambas filtradas pelo sistema visual humano (SVH), como mostrado na Fig. 2.7. O objetivo é minimizar o MSE, de forma que a imagem obtida seja a mais suave e natural possível para o SVH.

Todas as técnicas de *halftoning* digital utilizam, de forma implícita ou explícita, um modelo para a visão humana. Não existe um modelo que descreva o SVH completamente. O SVH possui um comportamento não-linear, uma vez que é capaz de perceber distorções em algumas frequências espaciais mais do que em outras.

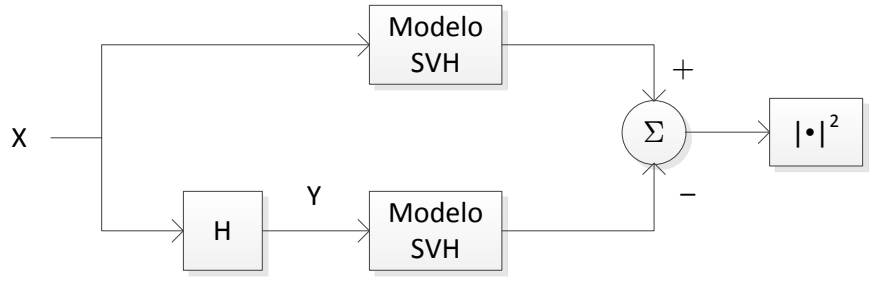


Figura 2.7: Diagrama em bloco do algoritmo para a avaliação de um método de *halftoning*.

Deste modo, o algoritmo de *halftoning* pode ter um melhor desempenho quanto a suavidade, homogeneidade ou visibilidade de texturas e padrões de acordo com o modelo utilizado, mas todos os modelos desenvolvidos concordam ao tratar o SVH como um filtro passa-baixas.

A distribuição dos pixels de forma aleatória realiza o espalhamento dos pixel minoritáros de forma homogênea, formando um arranjo isotrópico e aperiódico, criando componentes de alta frequência espectral, também chamadas de frequências *azuis*, por analogia com o espectro de cores visíveis. O SVH atua então como um filtro passa-baixas, o que torna as imagens visualmente agradáveis. A frequência espacial mais alta, ou "azul", criada por este método faz com que ele seja classificado também como uma técnica de *blue-noise dithering* (espalhamento de ruído azul).

Essa alta frequência espacial advém do não uso de matrizes de erros, permitindo que os pixels sejam quantizados de forma mais livre. O algoritmo de difusão de erros se baseia no fato de que, em imagens naturais, os pixels próximos estão fortemente correlacionados e utiliza essa característica para calcular o valor do pixel.

Como pode ser visto nas Figs. 2.8(a) e 2.8(c), o valor de cada um dos pixels varia bastante ao longo de uma linha da imagem, utilizando grande parte da faixa dinâmica. O mesmo ocorre quando analisamos uma coluna da imagem (Figs. 2.8(b) e 2.8(d)). Mas se analisarmos a diferença entre um pixel e seu vizinho, tanto vertical como horizontalmente, percebemos que a diferença entre eles é baixa (Figs. 2.8(e) e 2.8(f)), mostrando a alta correlação existente. Com base nessa característica das imagens naturais, o algoritmo de difusão de erros calcula a diferença entre o valor do pixel ($x(n)$) e o valor na saída do filtro ($x_p(n)$). O sinal $x_p(n)$ é uma tentativa de predizer o valor do pixel ($x(n)$) e é calculado com base em sinais propagados por pixels vizinhos, que são aplicados ao filtro do algoritmo e recebem um peso w de acordo com sua posição i em relação ao pixel analisado n. A Eq. (2.2) apresenta o cálculo de ($x_p(n)$). Ao realizar a operação $x(n) - x_p(n)$, grande parte da redundância existente no sinal é eliminada e o resultado, um sinal com informação nova, que não pôde ser predita a partir dos pixel vizinhos, é quantizado conforme a Eq. (2.2).

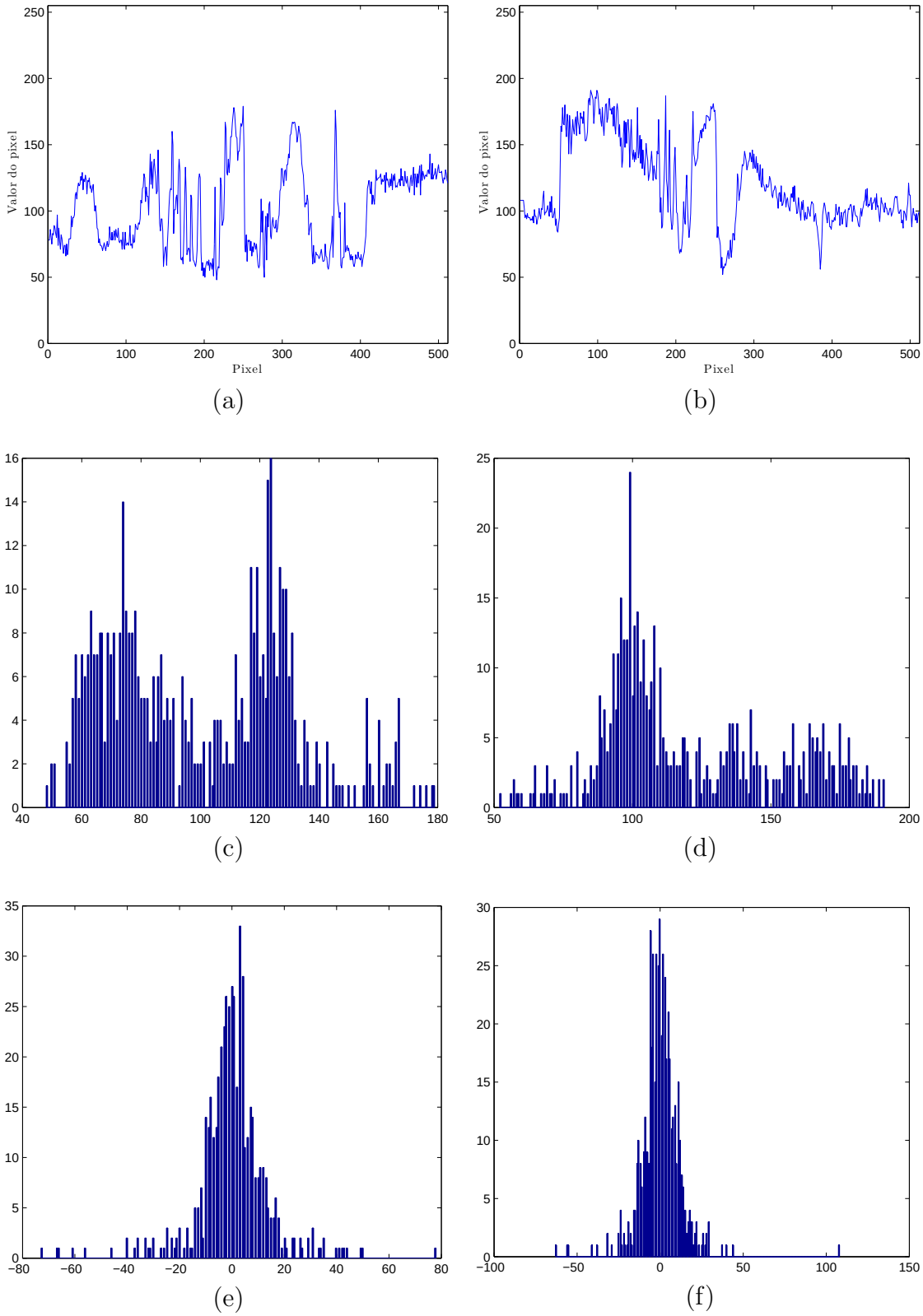


Figura 2.8: Análise da linha e coluna 256 da imagem Lena: (a) valores dos pixels ao longo da linha; (b) valores dos pixels ao longo da coluna; (c) Histograma da linha; (d) histograma da coluna; (e) histograma da diferença entre os pixel da linha; (f) histograma da diferença entre os pixel da coluna.

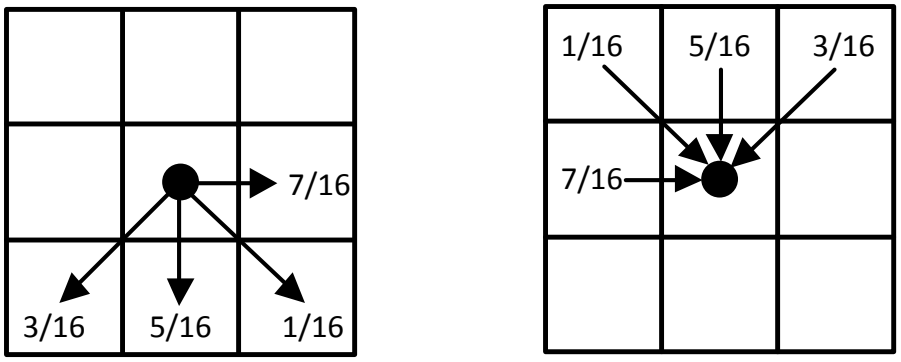
Conhecida a saída do pixel e o valor aplicado na entrada do quantizador, o erro de quantização $y_e(n)$ é calculado (Eq. (2.3)) e propagado para os filtros dos pixels vizinhos, permitindo que os demais pixels sejam quantizados.

$$y(n) = \begin{cases} 1, & \text{se } x(n) - x_p(n) \geqslant 0 \\ 0, & \text{caso contrário} \end{cases}, \tag{2.1}$$

$$x_p(n) = \sum_{i=1}^{M} w_i \cdot y_e(n-i), \tag{2.2}$$

$$y_e(n) = y(n) - (x(n) - x_p(n)). \tag{2.3}$$

Floyd e Steinberg propuseram um filtro [17] com quatro componentes, como mostrado na Fig. 2.9, utilizando uma varredura da esquerda para direita. A Fig. 2.9(a) mostra como os erros são propagados por um pixel para seus vizinhos e a Fig. 2.9(b) mostra como um dado pixel pondera, através do filtro, os erros propagados por seus vizinhos. Algumas perturbações em texturas múltiplas de $\frac{1}{3}$ e $\frac{1}{4}$ dos níveis de cinza (g) [17] são produzidas por este filtro. Em níveis extremos de cinza ($g = 1$ e $g = 0$), estruturas conhecidas como (*worms*) são criadas. Os níveis de cinza g variam de 0 (cor branca) a 1 (cor preta), onde o nível $g = \frac{1}{2}$ significa que a tonalidade de cinza é formada por 50% de pixels pretos e 50% de pixels brancos.



(a) (b)

Figura 2.9: Pesos do filtro de Floyd e Steinberg: (a) maneira como o erro é propagado para o pixel vizinho e (b) maneira como é calculada a predição do bloco utilizando as informações de erro dos pixels vizinhos.

Mais tarde, outros trabalhos foram desenvolvidos com o intuito de reduzir texturas e artefatos indesejáveis produzidos por este filtro. Alguns algoritmos obtiveram

melhor desempenho em níveis de cinza onde Floyd e Steinberg não tiveram sucesso mas, em contrapartida, apresentaram estruturas indesejadas em níveis onde o filtro já apresentado tem bons resultados. Um desses algoritmos, proposto por Jarvis *et al* [18], consiste em um filtro composto por 12 elementos (Fig. 2.10). Este filtro elimina as estrutras indesejáveis presentes nos níveis extremos mas promove um aspecto granulado em tons médios de cinza. O filtro proposto por Stucki [19] tem desempenho similar ao algoritmo proposto por Jarvis, melhorando os níveis extremos e degradando os médios [20]. A estrutura do filtro é apresentada na Fig. 2.11.

		•	7/48	5/48
3/48	5/48	7/48	5/48	3/48
1/48	3/48	5/48	3/48	1/48

Figura 2.10: Filtro proposto por Jarvis, Judice e Ninke. Os pesos indicam como o erro é propagado para os pixels vizinhos.

		•	8/42	4/42
2/42	4/42	8/42	4/42	2/42
1/42	2/42	4/42	2/42	1/42

Figura 2.11: Filtro proposto por Stucki.

			•	8/16
1/16	1/16	2/16	4/16	0/16

Figura 2.12: Filtro proposto por Shiau e Fan.

Shiau e Fan também propuseram um filtro [21] com desempenho similar, mas com dimensões diferentes dos outros dois algoritmos, como observado na Fig. 2.12. As dimensões do filtro influenciam no seu desempenho, sendo que filtros grandes têm um melhor desempenho em tons de cinza próximos a $g = 0$ e $g = 1$, enquanto que filtros menores trabalham melhor com tons médios de cinza e por isso alguns trabalhos propuseram algoritmos com filtros de dimensão variável [22] [23] que se adaptam conforme a tonalidade de cinza processada, permitindo a redução de artefatos ou contornos indesejados.

Algumas imagens foram geradas com esses quatro filtros para demonstrar as diferenças e similaridades entre eles. Na Fig. 2.13 são apresentadas cinco imagens da Lena, sendo uma delas parte da imagem original. As outras quatro são resultados do processamento da imagem através do algoritmo de *halftoning* usando os diferentes

filtros apresentados. Na Fig. 2.13(b) temos o resultado obtido através do algoritmo de Floyd e Steinberg. Os níveis médios de cinza possuem aparência agradável devido à distribuição uniforme dos pontos. Essa percepção fica mais clara quando são observadas as Figs. 2.13(c) e 2.13(d) onde é possível notar um nível maior de texturas nos níveis médios de cinza, dando um aspecto mais granulado à imagem. Esse aspecto granulado se deve a um branqueamento espectral, já observado por Knox [20], produzido pelo filtro e que faz com que a imagem tenha um contraste melhor, pois a inserção das altas frequências permitem uma transição mais abrupta entre os níveis de cinza. Exemplos deste melhor contraste são a borda da aba e as plumas do chapéu. Já na imagem da Fig. 2.13(e), efeitos tipo *minhoca* ficam evidentes por quase toda imagem e o filtro não consegue representar alguns detalhes, como pode ser visto na fita enrolada ao redor do chapéu e também em detalhes de sombra como no nariz.

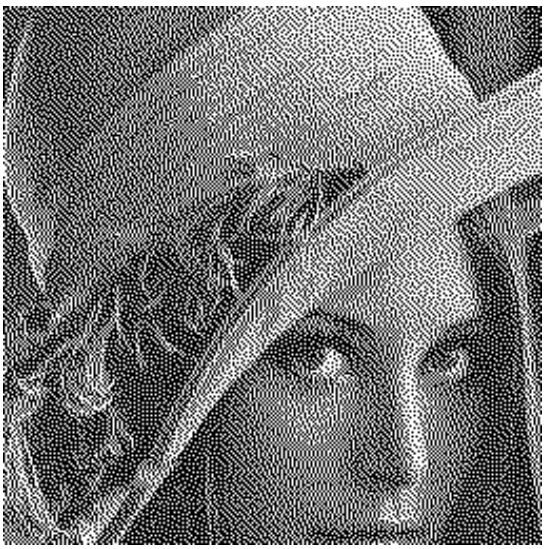
Novamente podemos observar, na Fig. 2.14(b), uma maior qualidade nos níveis médios de cinza como, por exemplo, nos pelos do rosto, ao redor dos olhos e parte escura nariz do babuíno [1]. Na parte mais clara do nariz (abaixo do olho esquerdo) é possível perceber a formação de *worms* que ficam ainda mais visíveis quando comparados às Figs. 2.14(c) e 2.14(d). No olho direito também podemos notar alguns *worms* que desaparecem à medida que ocorre a transição do preto para o cinza. A imagem mostrada na Fig. 2.14(e) novamente é a que apresenta pior desempenho, com forte saturação no branco do nariz do *baboon*.

A última sequência de imagens [24] corrobora as observações feitas em relação às duas sequências anteriores. Na Fig. 2.15(b) os níveis de cinza apresentam qualidade superior ao apresentado nas outras três imagens e na parte acima do nome do jornal a presença de *worms* é bastante evidente. As Figs. 2.15(c) e 2.15(d) apresentam tons médios de cinza mais granulados do que na Fig. 2.15(b) e com transição abrupta próximo ao nome do jornal. Na Fig. 2.15 é evidente a saturação para níveis de cinza claros e a presença de texturas indesejadas por toda imagem.

O melhor desempenho em níveis médios de cinza, que são os níveis predominantes em imagens naturais, nos fez optar pelo filtro proposto por Floyd e Steinberg. Além disso, dentre os filtros aqui analisados, este é o que possui menor dimensão, tornando sua implementação, a nível de circuito elétrico, menor, e portanto, mais simples.

No capítulo a seguir é abordada a implementação do algoritmo através de circuitos eletrônicos. Para um melhor entendimento, não só do circuito, como também do algoritmo, é apresentada a Fig. 2.16 que dá uma visão mais detalhada de como um pixel interage com seus vizinhos e como o algoritmo apresentado neste capítulo trabalha a nível de pixel.

(a)



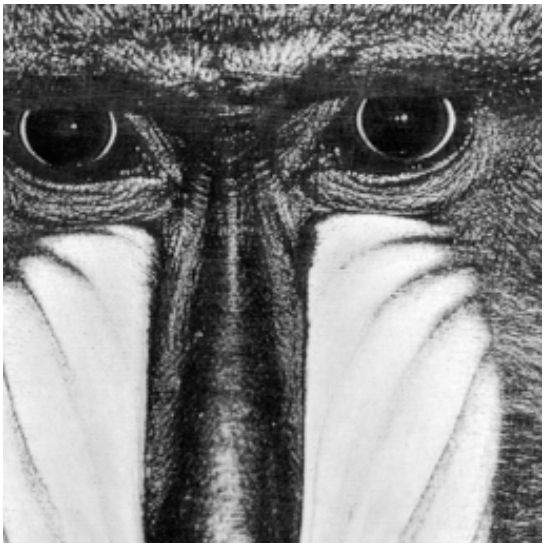
(b)

(c)

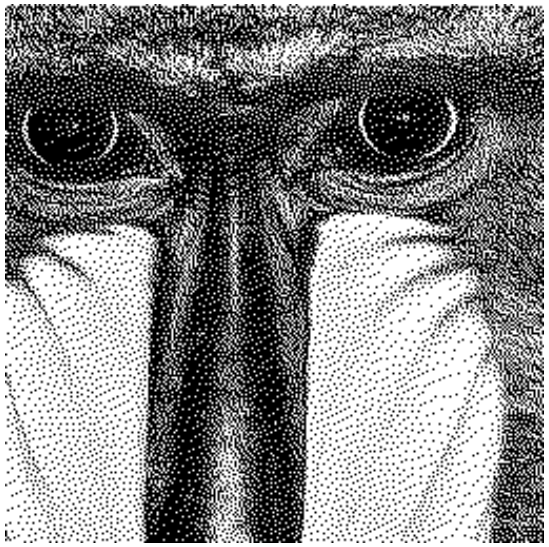
(d)

(e)

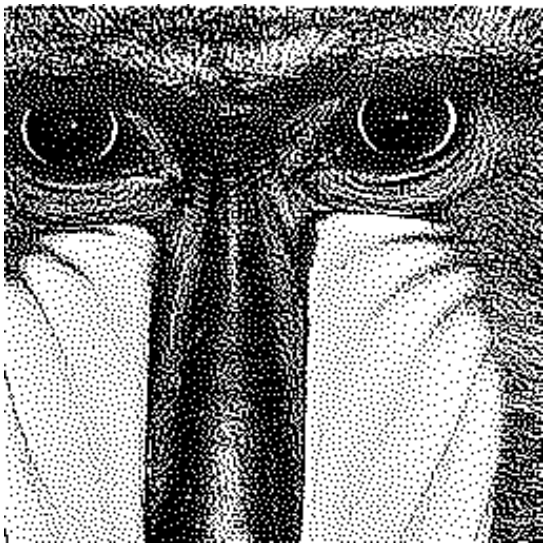
Figura 2.13: Imagens geradas pelo algoritmo de difusão de erros: (a) imagem original; (b) Floyd e Steinberg; (c) Jarvis, Judice e Ninke; (d) Stucki e (e) Shiau e Fan.



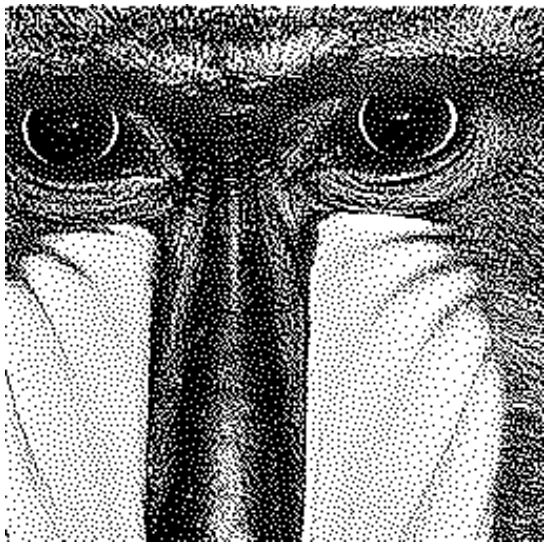
(a)



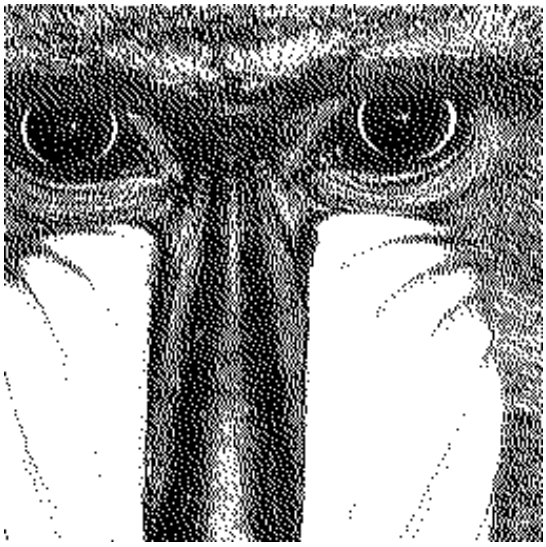
(b)



(c)



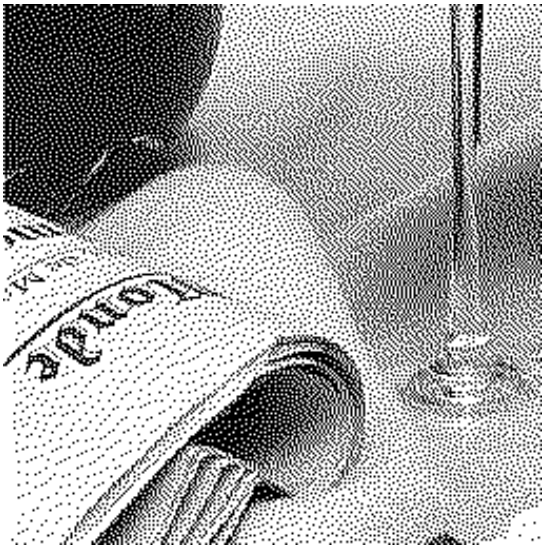
(d)



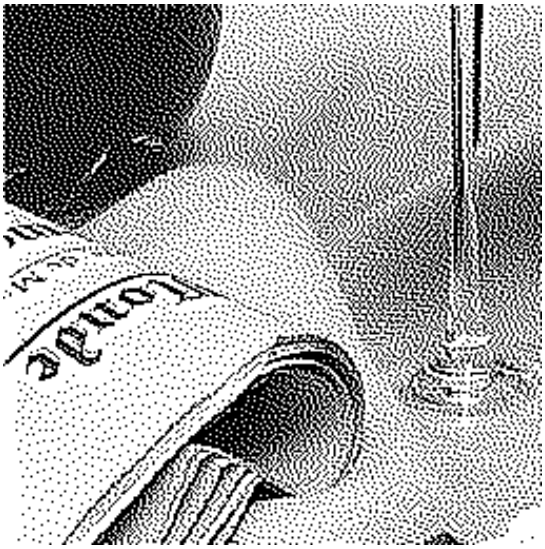
(e)

Figura 2.14: Imagens geradas pelo algoritmo de difusão de erros: (a) Iimagem original; (b) Floyd e Steinberg; (c) Jarvis, Judice e Ninke; (d) Stucki e (e) Shiau e Fan.

(a)

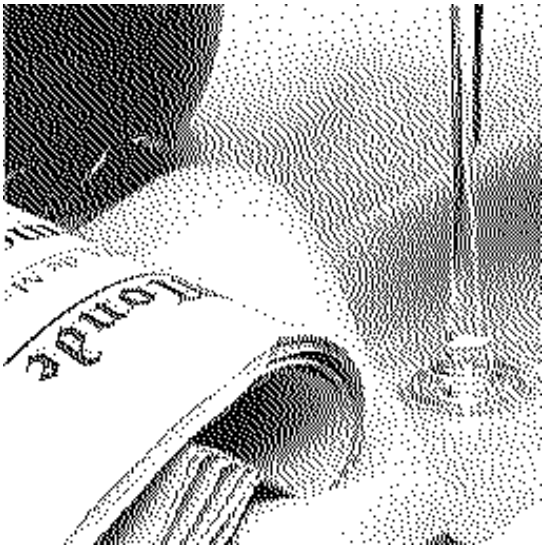


(b)



(c)

(d)



(e)

Figura 2.15: Imagens geradas pelo algoritmo de difusão de erros: (a) imagem original; (b) Floyd e Steinberg; (c) Jarvis, Judice e Ninke; (d) Stucki e (e) Shiau e Fan.

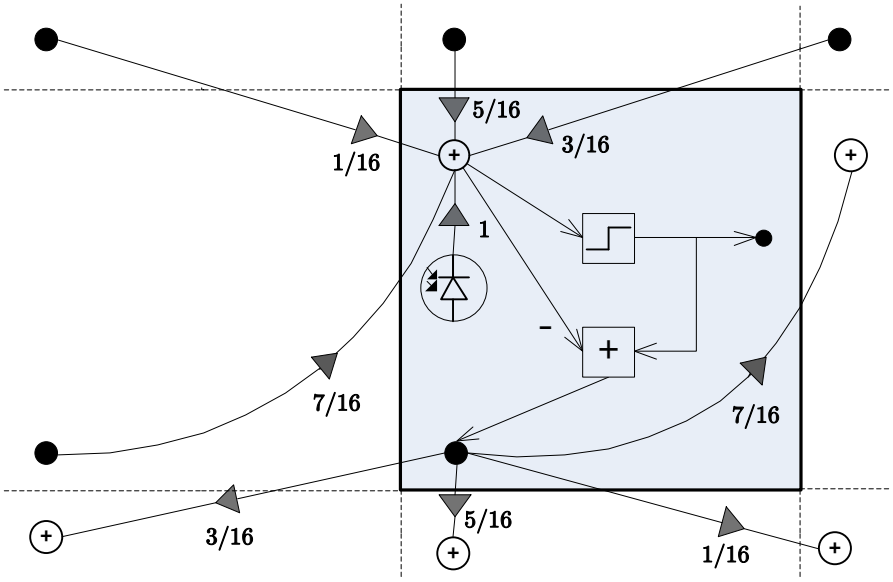


Figura 2.16: Diagrama de interconexão e funcionamento do pixel.

Capítulo 3

Método Proposto

Os circuitos apresentados a seguir foram projetados utilizando o software de simulação *Cadence Spectre* em conjunto com o *Design Kit* da AMS (*austriamicrosystems*) para a tecnologia CMOS de 0,35 μm que possui modelos com parâmetros reais extraídos dessa tecnologia.

O circuito foi desenvolvido para operar em modo de corrente por ser a opção natural para utilização em sensores APS. Os circuitos utilizados para realizar operações básicas nesse tipo de processamento são mais simples e, portanto, geralmente menores. A complexidade, e consequentemente o tamanho, dos circuitos envolvidos pode ser determinante no projeto de um imageador APS, uma vez que o espaço disponível para os circuitos de processamento dentro de um sensor de imagem é extremamente limitado. O modo de corrente também é mais adequado ao escalonamento e permite que fatores multiplicativos sejam ajustados de forma precisa através das dimensões dos transistores. Na Fig. 3.1 novamente é apresentado o diagrama de blocos do algoritmo de *halftoning*, pois é com base nele que serão apresentados os circuitos desenvolvidos, com ênfase na função que cada um deles desempenha no diagrama de blocos.

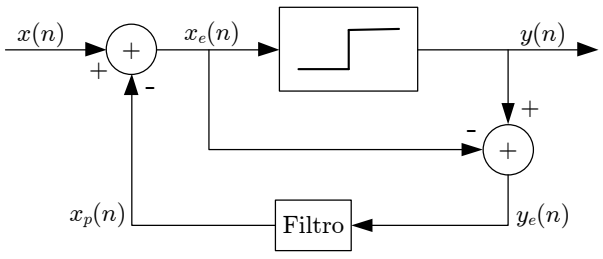


Figura 3.1: Diagrama em bloco do algoritmo de *halftoning*.

Mas antes de iniciar a descrição dos circuitos utilizados no processamento do sinal, é necessário abordar o processo de captura da luz e sua conversão no sinal elétrico $x(n)$ a ser processado.

3.1 Circuito de Leitura

A luz ao incidir no semicondutor gera, devido à energia dos fótons, pares elétron-lacuna que se deslocam através do fotodiodo devido à diferença de potencial existente entre seus terminais. A corrente reversa no diodo, chamada de i_{ph}, pode ser calculada de forma linear e com boa aproximação através da Eq. (3.1) [25]

$$i_{ph} = R_{ph} \cdot L_o \cdot A, \tag{3.1}$$

onde i_{ph} (A) corresponde ao valor da fotocorrente, R_{ph} (A/W) é a responsividade do material do qual é construído o fotodiodo e L_o (W/m^2) é a iluminância que incide sobre a área A (m^2) do fotodiodo.

Como pode ser visto na Eq. (3.1), para projetar o circuito responsável pela conversão luz-corrente é necessário definir algumas especificações do sensor. Utilizando um fotodiodo de dimensões 10 μm $\times$ 10 μm (o que resulta em uma capacitância de junção de aproximadamente 59 fF) e considerando a responsividade R_{ph} do fotodiodo de silício para o comprimento de onda verde (550 nm) aproximadamente igual a 0,3 A/W [25], nos resta definir a faixa dinâmica de operação do circuito. Na Fig. 3.2 [25] é apresentada uma escala relacionando o valor da iluminância (em lux) com diferentes condições de iluminação. Com base nessa escala, definimos a iluminância máxima captada pelo circuito igual a 10.000 lux (que equivale a aproximadamente 14,6 W/m^2). A fotocorrente máxima gerada no fotodiodo com a incidência de 10.000 lux é aproximadamente igual a 480 pA e é com base nessa corrente, chamada de i_{max}, que o circuito apresentado na Fig. 3.3 será projetado.

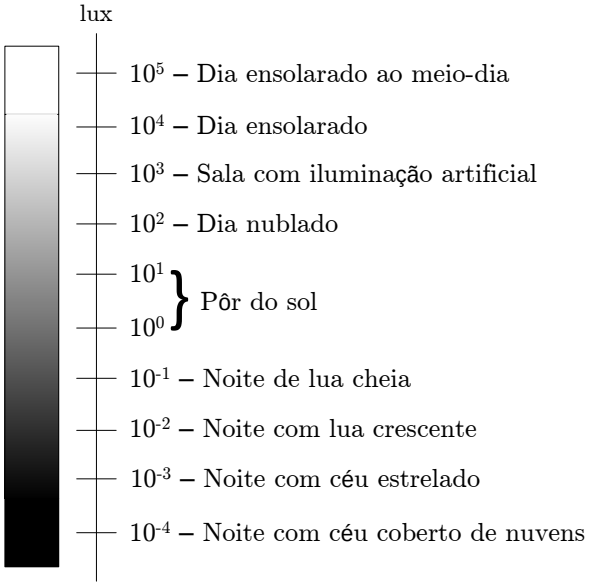


Figura 3.2: Escala de iluminância.

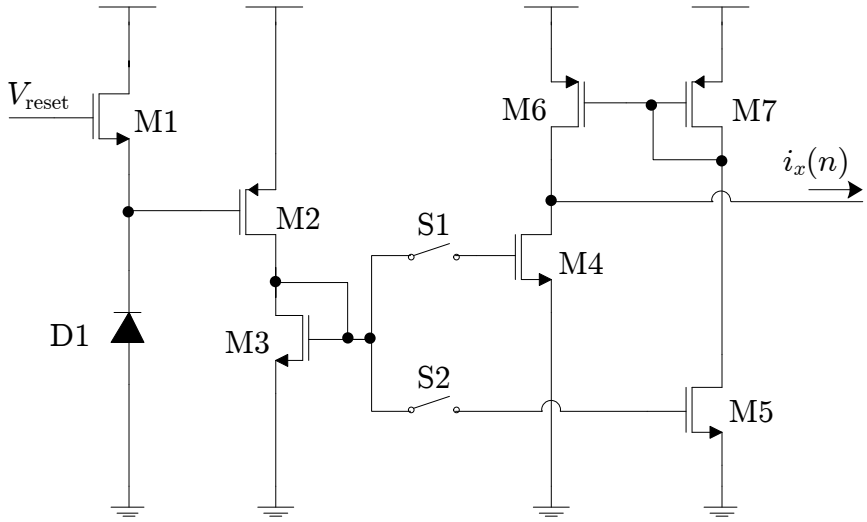


Figura 3.3: Circuito de leitura.

A conversão do sinal luminoso que incide no fotodiodo D1 em um sinal elétrico pronto para o processamento é iniciado com um pulso de tensão (V_{reset}) aplicado no terminal de porta do transistor M1. Enquanto este pulso estiver sendo aplicado o circuito permanecerá inoperante e o capacitor de junção C_j do fotodiodo será carregado com a tensão $V_{dd} - V_{tn}$ do transistor M1. Na Fig. 3.4 podemos observar que somente após o sinal V_{reset} passar de nível lógico um para nível lógico zero é que o processo de fotoconversão é iniciado. A corrente reversa no fotodiodo descarrega o capacitor C_j, alterando a tensão na porta do transistor M2 que tem transcondutância aproximadamente igual a 96 μA/V. Por se tratar de um transistor canal p, à medida que a tensão na porta do transistor M2 diminui, a corrente de dreno aumenta, como visto na Fig. 3.4. Vale notar que o pulso de reset pode ser mais curto do que o apresentado na Fig. 3.4, uma vez que a função do pulso é apenas carregar C_j.

As chaves S1 e S2 em conjunto com os transistores M3 a M7 são resposáveis por realizar o CDS (*correlated double sampling*) com a finalidade de eliminar o *offset* DC do pixel. Logo após o sinal V_{reset} mudar para nível lógico zero dando início ao processo de fotoconversão, a chave S1 fecha e abre durante um curto espaço de tempo, mas tempo sufuciente para que a cargas fiquem aprisionadas na capacitância de porta, mantendo a cópia da corrente $i_{D_{M2}}$ no transistor M4. Transcorrido o tempo necessário para captura do sinal luminoso, a chave S2 é fechada por um curto instante de tempo e logo em seguida aberta e a corrente que circula neste instante por M2 é copiada para M5 ($i_{D_{M5}}$). O espelho de corrente formado pelos transistores M6 e M7 injeta uma corrente igual a $i_{D_{M5}}$ no nó onde está conectado o dreno do transistor M4. Como M4 está drenando uma corrente menor, amostrada no início do processo de fotoconversão, a diferença entre as correntes, chamada de $i_x(n)$, é drenada pelo

circuito seguinte.

O sinal $x(n)$ mostrado na Fig. 3.1 representa a corrente $i_x(n)$ que é drenada pelo circuito de processamento. Mas antes de abordar o circuito de processamento, vejamos alguns detalhes sobre as chaves S1 e S2 e sua influência sobre a corrente $i_{x(n)}$.

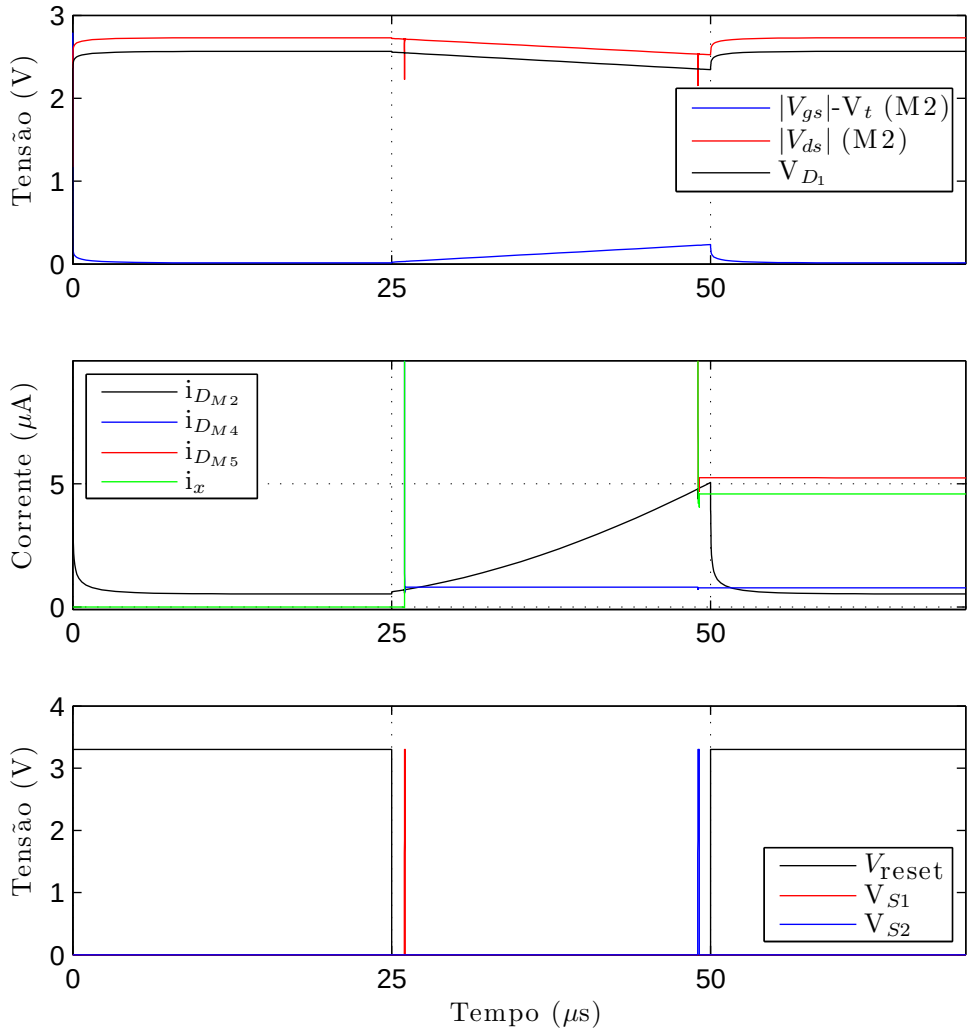


Figura 3.4: Diagrama de tempo do circuito do pixel e CDS.

3.2 Chaves Analógicas

As chaves analógicas são parte do circuito de CDS e é importante que elas sejam bem dimensionadas a fim de evitar que ocorra injeção de carga durante a cópia das correntes, tornando o CDS menos eficiente. No projeto do circuito da Fig. 3.5, inicialmente as chaves e o espelho de corrente formado pelos transistores M3, M4 e M5 foram projetados com as dimensões apresentadas na Tab. 3.1 pensando na questão da área disponível. No entanto os primeiros testes mostraram que o efeito da injeção de carga era bastante alto, como mostrado na Fig. 3.6.

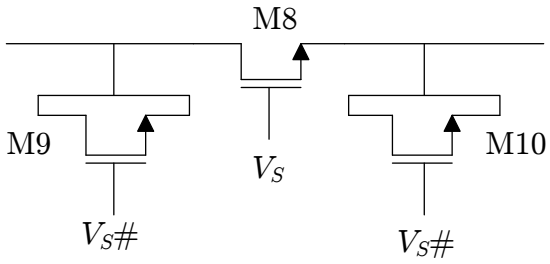


Figura 3.5: Circuito que compõe as chaves analógicas.

Tabela 3.1: Dimensão inicial dos transistores em M3, M4, M5 e transistores que compoẽm as chaves.

Transistor	W (μm)	L (μm)
M3	2	2
M4	2	2
M5	2	2
M8	0.8	0.35
M9	0.4	0.35
M10	0.4	0.35
M11	0.8	0.35
M12	0.4	0.35
M13	0.4	0.35

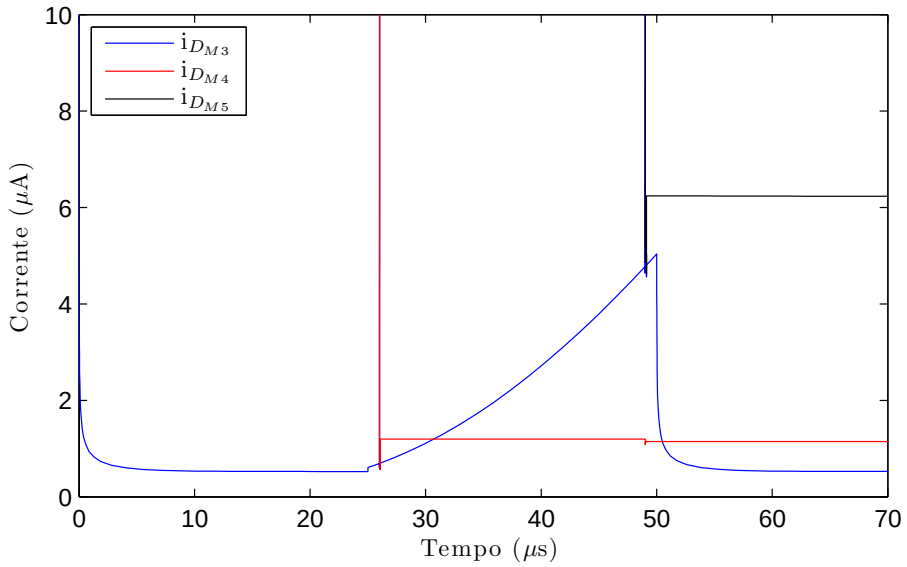


Figura 3.6: A injeção de carga pode ser observada ao copiar a corrente em M3 (azul) para M4 (vermelho) e M5 (preto) aos 27 μs e 48 μs respectivamente. Efeito provocado pelas chaves usando transistores com W = L = 2 μm no espelho de corrente.

Visando manter a cópia da corrente o mais fiel possível, a estrutura da chave foi modificada e chaves complementares foram utilizadas na tentativa de reduzir a injeção de carga nos transistores do espelho. Algumas simulações foram executadas e os resultados (Fig. 3.7 e 3.8) mostram que o uso do par complementar pouco

influencia na injeção de carga. Isso ocorre porque a tensão na porta do transistor M3 sofre pouca variação, excursionando de aproximadamente 0,55 V a 0,75 V, o que faz com que na maior parte do tempo apenas os transistores NMOS da chave estejam funcionando. Uma vez que os transistores não conduzem simultaneamente, o cancelamento de cargas não ocorre, tornando a utilização da chave complementar desnecessária por não apresentar melhora significativa no desempenho do circuito.

Segundo a Eq. (3.2) [26], se metade da carga Q_{ch} da chave for injetada no capacitor de porta (C_g) do transistor, esta introduzirá um erro na tensão (ΔV) por ele armazenada. Com base nessa equação, é possível concluir que ao aumentar C_g, o que significa aumentar as dimensões do transistor, a variação de tensão ΔV diminuirá, para uma mesma quantidade de cargas injetadas. Na Fig. 3.7 notamos que a corrente $i_{D_{M3}}$ é copiada para M4 ($i_{D_{M4}}$) com menor erro (linha pontilhada indica o valor ideal) ao utilizarmos transistores com W e L iguais a 5 μm (curvas vermelha e verde entre os instantes de tempo de 27 μs a 48 μs). O mesmo pode ser observado na Fig. 3.8, onde a partir dos 48 μs fica claro que as curvas obtidas utilizando transistores de 5 μm (curvas vermelha e verde) são as que mais se aproximam da curva ideal (curva pontilhada), apresentando o menor erro na cópia da corrente para o transistor M5. Desta forma podemos concluir que o aumento das dimensões dos transistores reduz significamente o problema causado pela injeção de cargas.

$$\Delta V = \frac{1}{2}\frac{\Delta Q_{ch}}{C} = \frac{WLC_{ox}(V_{DD} - V_{in} - V_{TH})}{2C_g}. \tag{3.2}$$

Os gráficos apresentados nas Figs. 3.7 e 3.8 confirmam a análise feita, mostrando que independente da chave ser ou não complementar, o circuito apresenta um menor erro na cópia da corrente quando os espelhos são construídos com transistores com W e L iguais a 5 μm. As simulações com chaves complementares (em conjunto com os transistores de 5 μm compondo o espelho de corrente) apresentaram resultados ligeiramente melhores na cópia da corrente (as curvas vermelha e verde de ambos os gráficos são as que mais se aproximam do resualtado ideal, representado pela linha preta pontilhada). Porém o fato de uma chave complementar ocupar uma área no mínimo duas vezes maior nos leva, neste primeiro momento, a não utilizar chaves complementares. A solução adotada consiste no uso das chaves simples com transistores *dummy* e dos transistores M3, M4 e M5 com W e L iguais a 5 μm.

3.3 Circuito de Processamento

Nesta seção será abordado o projeto do circuito cujo objetivo, de forma geral, é quantizar o sinal de entrada, gerando um sinal de erro que será propagado para os

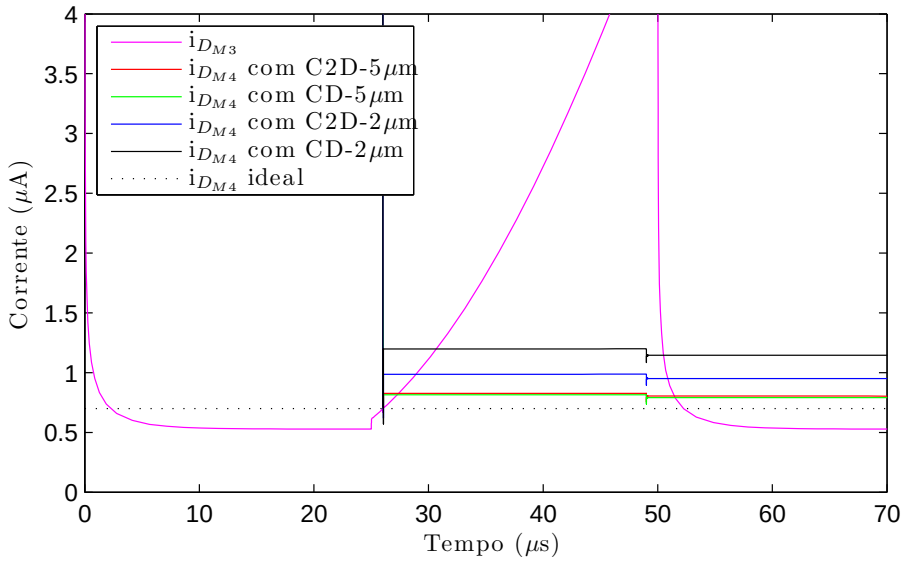


Figura 3.7: Resultado da injeção de carga durante a primeira cópia da corrente, feita através de S1. C2D indica que foram utilizadas chaves complementares com transistores *dummy* e CD chaves simples com transistores *dummy*. As dimensões 2 μm e 5 μm se referem às dimensões W e L dos transistores do espelho de corrente.

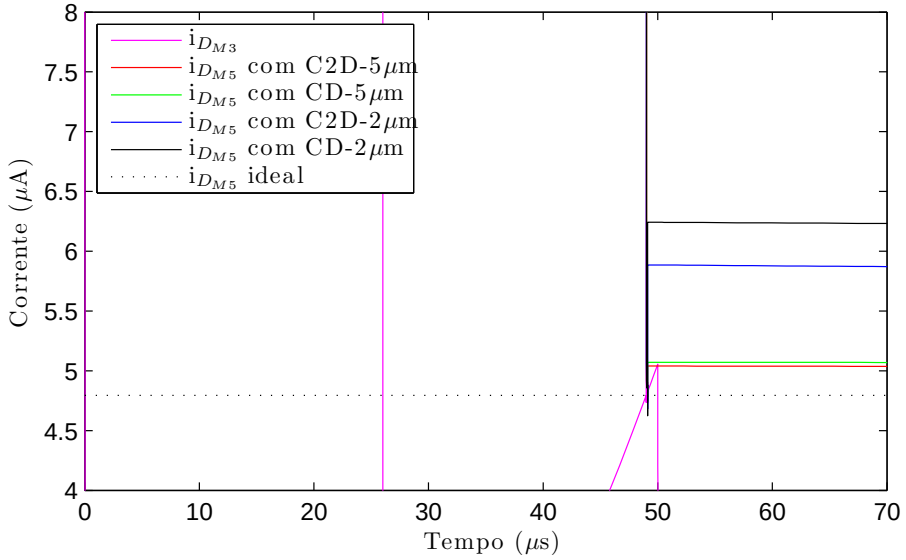


Figura 3.8: Resultado da injeção de carga durante a segunda cópia da corrente, feita através de S2. C2D indica que foram utilizadas chaves complementares com transistores *dummy* e CD chaves simples com transistores *dummy*. As dimensões 2 μm e 5 μm se referem às dimensões W e L dos transistores do espelho de corrente.

pixels vizinhos, que utilizarão esse sinal para processar o sinal capturado por cada um deles. Cada pixel possui um circuito de processamento dedicado e idêntico ao utilizado pelos demais pixels.

A Fig. 3.9 mostra o circuito que realiza a primeira operação sobre o sinal $i_x(n)$. Essa operação consiste em subtrair de $i_x(n)$ o sinal processado pelo filtro de Floyd-

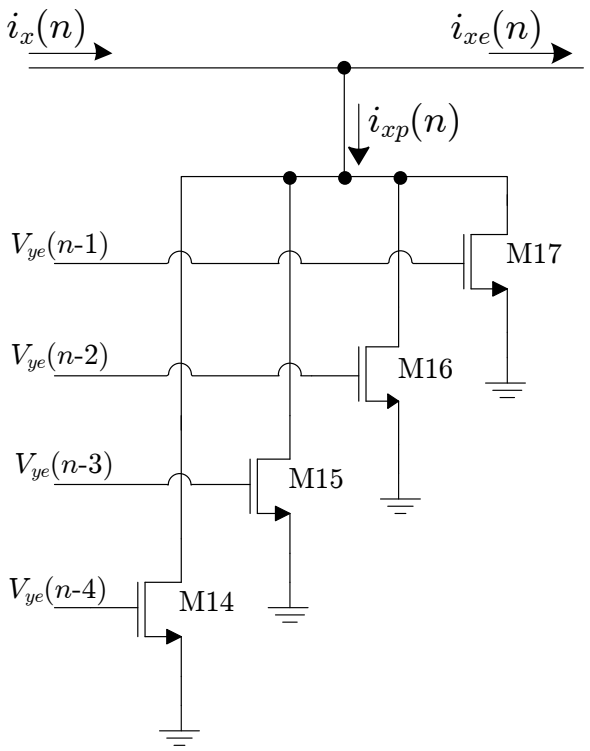


Figura 3.9: Filtro de Floyd-Steinberg.

Steinberg e neste momento iremos nos ater apenas ao fato de que o sinal processado pelo filtro é fornecido pelos pixels vizinhos (conforme mostrado no Capítulo 2). Mais adiante sua origem será explicada.

Como visto no Capítulo 2, o filtro de Floyd-Steinberg tem a função de combinar sinais provenientes de quatro pixels vizinhos, atribuindo pesos para cada um desses sinais de acordo com a posição do pixel que o propaga. Como o processamento do circuito foi todo projetado em modo de corrente, a atribuição dos pesos no circuito elétrico é feita através do ajuste das dimensões físicas dos transistores M14, M15, M16 e M17 que são parte de espelhos de corrente. Esses quatro transistores formam, portanto, o filtro; e o somatório das correntes por eles copiadas forma o sinal $i_{xp}(n)$.

A corrente $i_{xp}(n)$, drenada pelos transistores do filtro, é subtraída da corrente $i_x(n)$, gerando um sinal que representa o erro de predição cometido pelo filtro, a corrente $i_{xe}(n)$, que será processada pelo quantizador. Fazendo um paralelo com o diagrama de blocos mostrado na Fig. 3.1, o circuito até aqui apresentado corresponde aos blocos mais escuros mostrados na Fig. 3.10.

Uma vez conhecida a corrente $i_{xe}(n)$, esse sinal é submetido a um circuito [2] que irá determinar se a predição do sinal $i_x(n)$, calculada pelo filtro, era maior ou menor que o valor real do sinal e a partir daí definir a saída $V_y(n)$ do pixel. O circuito responsável por tal tarefa é apresentado na Fig. 3.11. A corrente $i_{xe}(n)$ pode ser positiva ou negativa, dependendo apenas do módulo das corrente $i_x(n)$ e $i_{xp}(n)$.

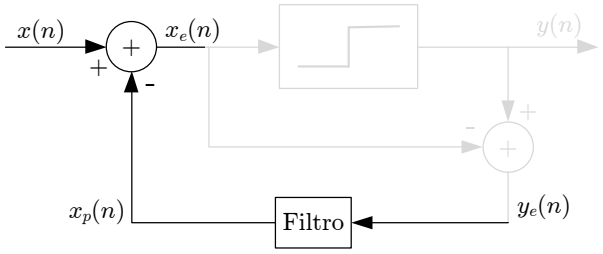


Figura 3.10: O circuito da Fig. 3.9 desempenha as funções dos blocos escuros deste diagrama.

A corrente é considerada positiva quando ela é drenada pelo circuito quantizador e negativa quando ela é drenada do circuito quantizador. A função do circuito apresentado é justamente a de determinar o sentido da corrente, disponibilizando o resultado desta análise na saída $V_y(n)$, de acordo com a Eq. (3.3).

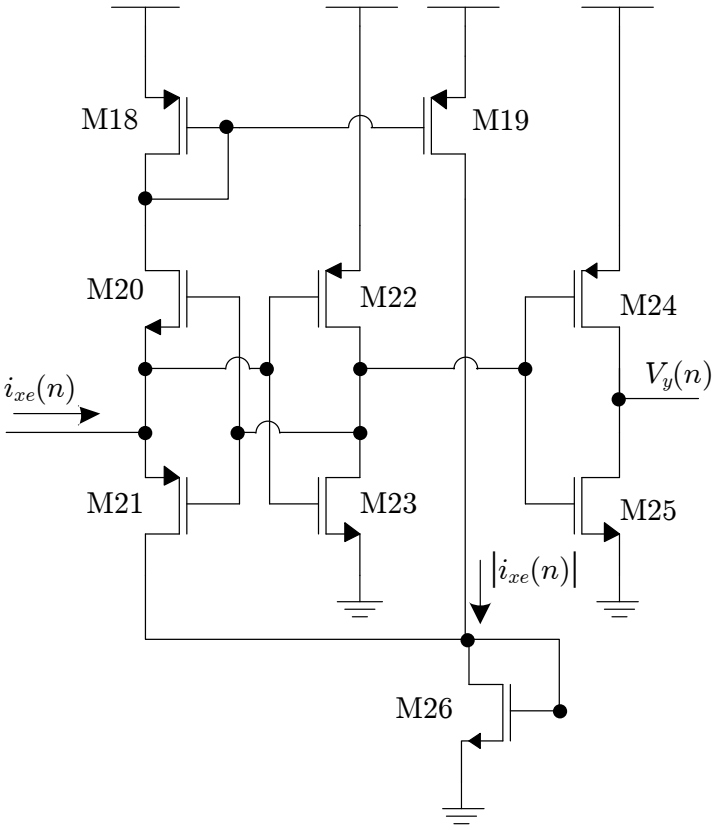


Figura 3.11: Circuito responsável por quantizar o sinal $i_{xe}(n)$ [2].

$$V_y(n) = \begin{cases} 3,3\ V, & \text{se } i_{xe}(n) \geq 0 \\ 0\ V, & \text{caso contrário.} \end{cases} \tag{3.3}$$

No caso da corrente $i_{xe}(n)$ ser positiva, a tensão no nó onde a corrente está sendo injetada também será positiva e como este nó está conectado na porta dos transistores M22 e M23 (que formam uma porta inversora) a tensão no dreno destes transistores terá nível lógico zero. A porta dos transistores M20 e M21, por estes estarem conectados no dreno de M22 e M23, estará também com nível lógico zero, o que fará com que o transistor M21 conduza a corrente $i_{xe}(n)$ para o transistor M26. Se a corrente $i_{xe}(n)$ for negativa a tensão na porta dos transistores M20 e M21 será positiva, fazendo o transistor M20 conduzir a corrente $i_{xe}(n)$, drenada pelo circuito do filtro. Através dos transistores M18 e M19 a corrente $i_{xe}(n)$ é copiada e disponibilizada através do transistor M26. A saída $V_y(n)$ é obtida na saída do circuito inversor formado pelos transistores M24 e M25 que recebem o sinal da porta inversora, formada pelos transistores M22 e M23, que funciona como um *buffer* para o sinal de entrada.

Através do transistor M26, que funciona como parte de um espelho de corrente, esse circuito fornece uma cópia do módulo de $i_{xe}(n)$ que, como veremos adiante, será utilizada em outra parte do circuito de processamento. Na Fig. 3.12 é apresentado, em cor mais escura, o bloco que representa o circuito da Fig. 3.11.

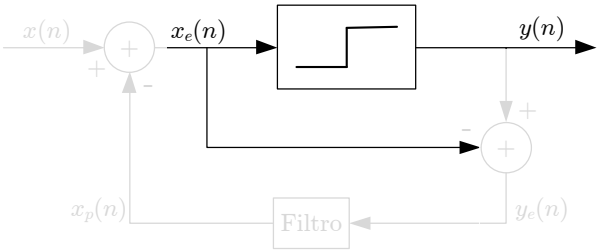


Figura 3.12: O circuito da Fig. 3.11 desempenha as funções dos blocos escuros deste diagrama.

Como podemos observar na Fig. 3.12, resta apresentar apenas o circuito responsável por realizar a subtração dos sinais $y(n)$ e $x_e(n)$ para obter $y_e(n)$. No entanto essa operação não é tão trivial quanto a solução apresentada no início dessa seção porque a saída $y(n)$ é representada pela tensão $V_y(n)$, enquanto x_e é representada pela corrente $i_{xe}(n)$.

Tornou-se, então, necessário converter um dos sinais para a mesma grandeza do outro. Converter a corrente em um sinal de tensão mostrou-se complicado porque seria necessário mapear a corrente de forma linear e com boa precisão em uma escala de tensão variando de 0 V a 3,3 V, realizar a subtração dos sinais e depois disso converter o resultado novamente para o modo de corrente (de forma linear, com boa precisão e em uma escala definida) para que pudesse ser utilizado nos circuitos seguintes. Uma vez que todo o processamento até aqui havia sido realizado em

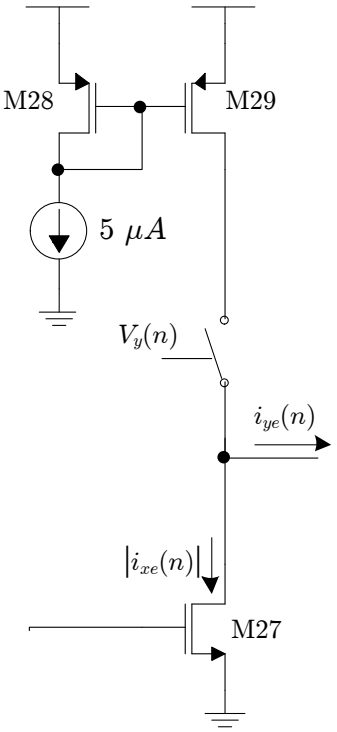


Figura 3.13: Circuito subtrator que calcula a corrente $i_{ye}(n)$.

modo de corrente, a solução mais coerente foi converter sinal $y(n)$ para o modo de corrente. Outra vantagem de realizar essa conversão é que o sinal $y(n)$ tem apenas dois valores válidos, que representam os valores máximo e mínimo do sinal. Definir esses valores não foi complicado, uma vez que a tensão de 0 V foi mapeada em uma corrente de 0 A, e conforme visto na Fig. 3.8, a corrente que representa a máxima intensidade de luz é aproximadamente igual a 5 μA. Esse foi o valor de corrente utilizado para representar a tensão de 3,3 V.

Na Fig. 3.13 é apresentado o circuito responsável por realizar a conversão. Vale notar que o circuito é bem simples e pode ser descrito pela Eq. (3.4), onde podemos ver que se $y(n) = 0$, a corrente no nó de saída será negativa. Para contornar esse problema, mais uma vez foi utilizado o circuito de módulo para fornecer a corrente de forma que possa ser utilizada nos blocos seguintes, através do transistor M39, que como já dito antes, é parte de um espelho de corrente.

$$i_{ye}(n) = \begin{cases} 5\ \mu A - |i_{xe}(n)|\,, & \text{se } V_y(n) = 3,3\ V; \\ -|i_{xe}(n)|\,, & \text{se } V_y(n) = 0\ V. \end{cases} \tag{3.4}$$

O circuito utilizado para realizar essa etapa de subtração é apresentado na Fig. 3.14 e sua representação no diagrama de blocos é dada pela Fig. 3.15. Importante lembrar que os quatro sinais de entrada do filtro de Floyd-Steinberg são tensões V_{ye}

propagadas por outros blocos vizinhos.

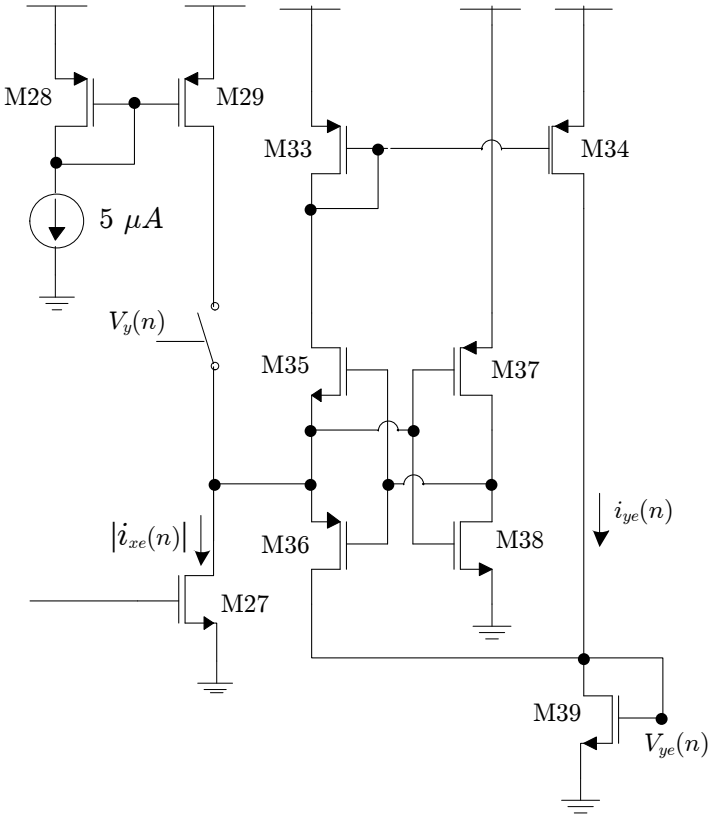


Figura 3.14: Circuito subtrator que calcula a corrente de erro de saída.

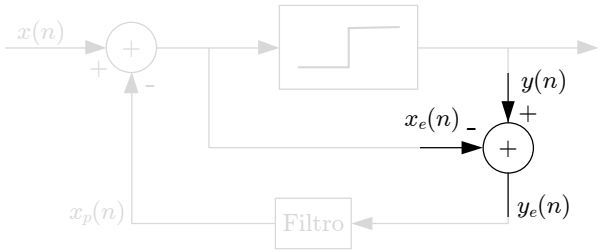


Figura 3.15: O circuito da Fig. 3.14 desempenha as funções dos blocos escuros deste diagrama.

Na Fig. 3.16 é apresentado o circuito de um pixel completo, incluindo tanto o circuito de foto-conversão quanto o circuito de processamento. As dimensões dos transistores utilizados estão disponíveis na Tabela 3.2

[1]O asterisco ao lado de alguns transistores da Tabela 3.2 indicam que estes transistores compõem as chaves analógicas.

Tabela 3.2: Dimensões dos transistores utilizados no circuito de cada pixel.

Transistor [1]	W (μm)	L (μm)
M1	1	0,35
M2	1	0,35
M3	5	5
M4	5	5
M5	5	5
M6	1	2
M7	1	2
M8*	0,8	0,35
M9*	0,4	0,35
M10*	0,4	0,35
M11*	0,8	0,35
M12*	0,4	0,35
M13*	0,4	0,35
M14	0,5	1
M15	1,5	1
M16	2,5	1
M17	3,5	1
M18	1	2
M19	1	2
M20	1,4	1
M21	4	1
M22	3	1
M23	1	1
M24	3	1
M25	1	1
M26	1	2
M27	1	2
M28	1	3.4
M29	1	3.4
M30*	0.8	0.35
M31*	0.4	0.35
M32*	0.4	0.35
M33	1	2
M34	1	2
M35	1,4	1
M36	4	1
M37	3	1
M38	1	1
M39	8	1

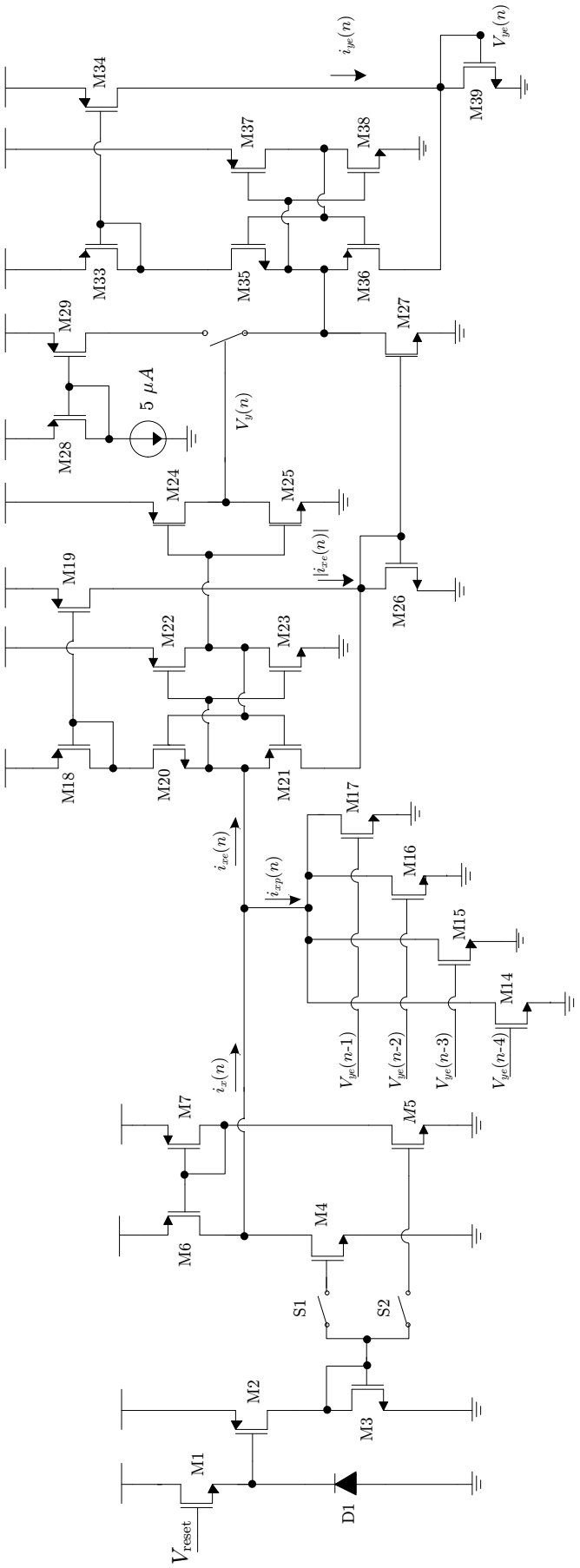


Figura 3.16: Circuito completo do pixel.

Capítulo 4

Resultados e Discussões

Com o propósito de validar o projeto apresentado neste trabalho, algumas simulações elétricas foram realizadas utilizando, como entrada para o circuito, sinais obtidos a partir dos pixels de uma região da imagem Lena.

Em uma imagem colorida codificada em RGB (*red, green and blue*), como é o caso da imagem utilizada, existem três camadas de cores: vermelha, verde e azul. As imagens monocromáticas são formadas, em geral, por uma combinação dessas três cores, conforme mostrado nas Eqs. (4.1) [27] e (4.2)[28]. A Eq. (4.1) representa o cálculo da luminância definido pelo padrão NTSC (*national television system committee*) e a Eq. (4.2) para o padrão utilizado pelo HDTV (*high definition television*). Em ambos os casos a camada verde do sistema RGB é a camada que mais contribui para a composição da imagem de luminância e por isso, apenas para efeito de simplificação, essa camada foi utilizada como imagem de referência.

$$Y = 0,2999R + 0,5870G + 0,1140B. \tag{4.1}$$

$$Y = 0,2126R + 0,7152G + 0,0722B. \tag{4.2}$$

Definida a imagem a ser utilizada, é necessário converter a intensidade de cada um dos pixels para valores de corrente que serão utilizados para simular a incidência de luz sobre os fotodiodos. Para isso é necessário normalizar a imagem (considerando o valor máximo, para um pixel qualquer da imagem, igual a 255) e multiplicar essa matriz resultante pelo fator $I_{max} = 480 \times 10^{-12}$ A (Seção 3.1) para obter a matriz $\mathbf{M}_{ph}$ com os valores das correntes reversas geradas nos fotodiodos quando expostos à imagem da Lena, conforme mostrado na Eq. (4.3).

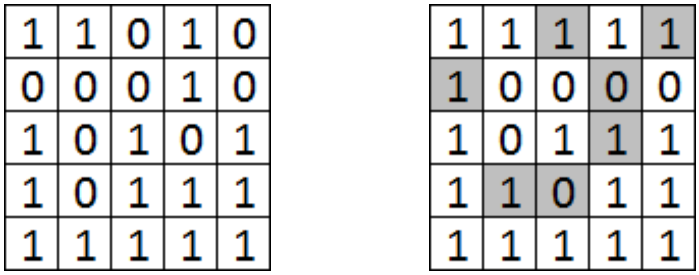
$$M_{ph}(m,n) = \frac{Y(m,n)}{255} i_{max}. \tag{4.3}$$

A matriz $\mathbf{M}_{ph}$ representa, portanto, o sinal de entrada do circuito, e as correntes são inseridas no circuito através de fontes de corrente ideais colocadas em paralelo com os fotodiodos (apenas para efeitos de simulação) com a função de carregar o capacitor de junção dos mesmos.

Para todas as simulações elétricas realizadas uma simulação equivalente utilizando o algoritmo ideal de Floyd-Steinberg foi feita no MATLAB, permitindo comparar os resultados obtidos. Em ambas as simulações foram respeitadas as mesmas condições de contorno. Os pixels localizados nas bordas superior e esquerda tiveram as entradas destinadas à conexão com pixels vizinhos inexistentes conectadas ao terra no caso da simulação elétrica. O valor zero foi associado às referidas entradas no caso da execução do algoritmo ideal.

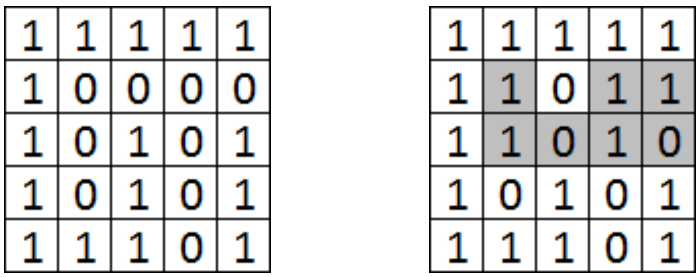
Figura 4.1: Blocos 5×5 utilizados na simulação.

A primeira simulação foi feita utilizando um circuito composto por 25 pixels, organizados como uma matriz quadrada de dimensão 5×5, com o intuito de avaliar o desempenho do circuito como um todo, principalmente em relação à propagação do erro entre os pixels vizinhos. Três diferentes partes da imagem Lena foram utilizadas nesta simulação e são apresentadas na Fig. 4.1. Um dos blocos 5×5 foi retirado propositalmente do olho da Lena para testar o comportamento do circuito em uma transição abrupta entre claro e escuro (alta frequência), enquanto que o bloco retirado da parede ao fundo foi utilizado para testar o circuito em uma condição oposta, quando praticamente não há variação no tom de cinza (baixa frequência) e o terceiro bloco, retirado do chapéu, para testar o circuito em uma condição



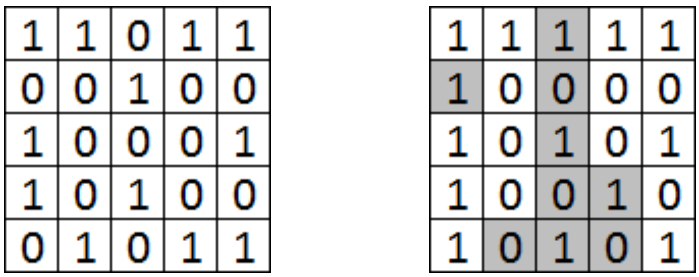
(a) (b)

Figura 4.2: Resultado da simulação utizando o bloco do olho da Lena: (a) algoritmo ideal no MATLAB e (b) simulação elétrica no SPICE.



(a) (b)

Figura 4.3: Resultado da simulação utizando o bloco do chapéu da Lena: (a) algoritmo ideal no MATLAB e (b) simulação elétrica no SPICE.



(a) (b)

Figura 4.4: Resultado da simulação utizando o bloco da parede: (a) algoritmo ideal no MATLAB e (b) simulação elétrica no SPICE.

intermediária. Nas Figs. 4.2, 4.3 e 4.4 são apresentados os resultados obtidos nessa simulação, onde os erros observados foram de 28%, 28% e 40% respectivamente, mas estes erros não importam muito. À primeira vista essas taxas de erros podem ser consideradas demasiadamente elevadas, mas essa suposição não é necessariamente correta no caso do processamento por *halftoning*, uma vez que a exata localização dos pixels claros e escuros não tem muito significado com respeito à representação da imagem como um todo. O que realmente importa em uma imagem de *halftoning* é a densidade de pixels claros e escuros presentes em uma determinada região, formando os diferentes tons de cinza que irão compor a imagem final.

Uma segunda simulação foi realizada para verificar se em uma imagem de maior dimensão as densidades de pixels claros e escuros se manteriam constantes nas diferentes regiões da imagem, formando os tons de cinza desejados. A imagem de dimensão 10×10 mostrada na Fig. 4.5 foi utilizada para essa simulação por apresentar dois tons de cinza bem distintos, o que permitiria ver, caso o circuito operasse conforme desejado, uma maior densidade de pixels escuros no canto superior esquerdo e uma menor densidade no canto inferior direito.

Para simular o circuito 10×10 utilizando a interface gráfica seria necessário inserir no projeto 100 pixels, 100 fontes de corrente ideais e fazer manualmente todas as conexões entre os blocos. Todo esse trabalho demandaria muito tempo para circuitos como esse ou maiores. Para realizar essa simulação foi, portanto, necessário criar um programa para geração automática do *netlist*. O código deste programa pode ser visualizado no Apêndice A.

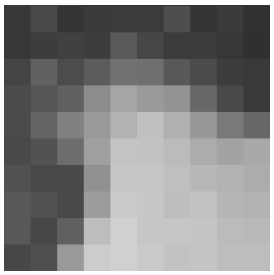
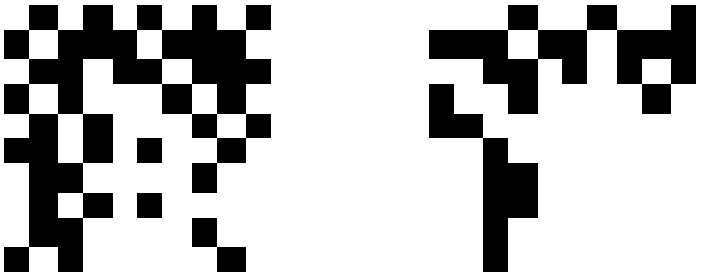


Figura 4.5: Bloco 10×10 extraído da imagem Lena.

Voltando à discussão anterior, os resultados do processamento da imagem utilizando o algoritmo ideal e o algoritmo implementado em *hardware* são apresentados na Fig. 4.6. Estas duas imagens nos permitem observar que os resultados estão correlacionados. No entanto a imagem resultante da simulação elétrica nitidamente possui uma densidade de pixels pretos menor do que a imagem obtida através do algoritmo ideal. O circuito se comporta como se estivesse sendo submetido a uma intensidade luminosa bastante alta, possivelmente acima da qual ele foi projetado para suportar. A luminosidade muito alta poderia causar limitação de tensão em alguns nós do circuito elétrico, explicando o mau funcionamento.

Com base na suposição feita, a solução consiste basicamente em reduzir o tempo de exposição do sensor, o que pode ser feito alterando a temporização das chaves do circuito de CDS, o que irá reduzir o valor da corrente processada pelo banco de transistores que compõem o filtro de Floyd-Steinberg. Podemos, alternativamente, alterar as especificações do sensor, reduzindo a iluminância máxima admitida pelo sensor. Em ambos os casos, essas alterações levam a um resultado comum que é a redução no valor da corrente de saída do circuito de CDS, o que mostra que essa suposição atribui esse comportamento (aparente saturação em níveis claros de cinza)



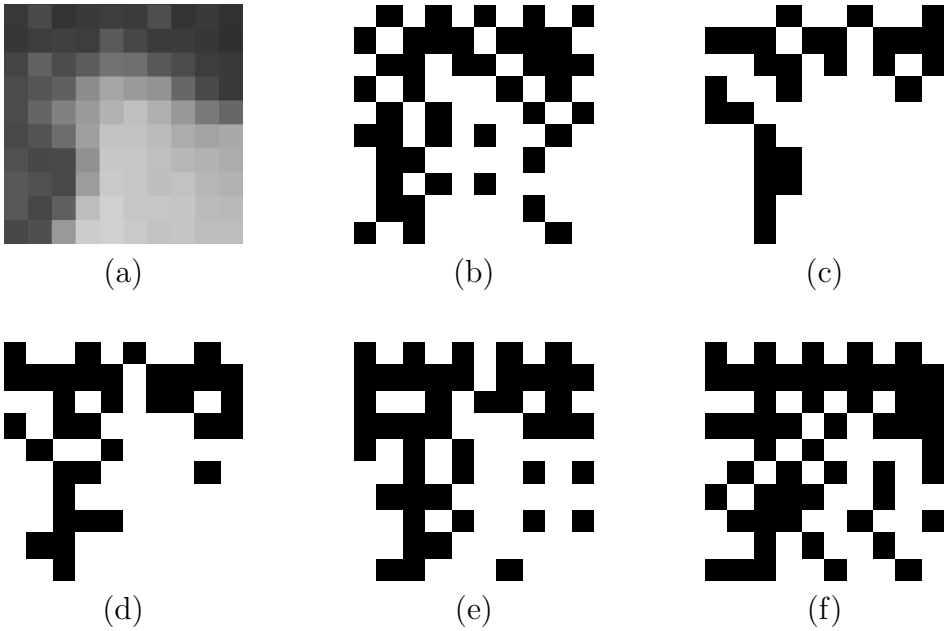
(a) (b)

Figura 4.6: Simulação do bloco 10×10: (a) algoritmo ideal no MATLAB e (b) simulação elétrica no SPICE.

ao circuito de leitura e não ao circuito que processa o sinal.

Para testar essa hipótese, foram feitas simulações variando o valor da corrente i_{max}, obtido através da Eq. (4.3). Na Fig. 4.7 são apresentados os resultados dessa simulação, onde podemos ver que a imagem obtida utilizando $i_{max} = 350 \cdot 10^{-12}$ A é a que possui a densidade de pixels mais próxima da imagem ideal gerada pelo MATLAB.

Esse resultado, ainda que de forma parcial, mostra que a suposição feita se mostrou correta: os fotodiodos estavam saturando. Isso nos permite concluir que o circuito de processamento, a princípio, funciona como esperado. Algo que deve ser observado é que ao corrigir a corrente i_{max} para o valor de $350 \cdot 10^{-12}$ A, a



(a) (b) (c)

(d) (e) (f)

Figura 4.7: Simulação do bloco 10×10: (a) parte original da imagem; (b) imagem processada através do algoritmo ideal no MATLAB; (c) simulação elétrica com $i_{max} = 480 \times 10^{-12}$ A; (d) simulação elétrica com $i_{max} = 400 \times 10^{-12}$ A; (e) simulação elétrica com $i_{max} = 350 \times 10^{-12}$ A e (f) simulação elétrica com $i_{max} = 300 \times 10^{-12}$ A.

corrente I_{ref}, calculada a partir do valor projetado de i_{max}, não foi modificada. O que se fez foi buscar o valor máximo de corrente na entrada que permitisse o circuito operar sem saturação, mantendo todos os demais parâmentros do circuito inalterados. A escolha do valor de $350 \cdot 10^{-12}$ A foi feita de forma informal e um estudo mais aprofundado, com um número maior de imagens, seria importante para definir formalmente o valor de i_{max}.

Para verificar como o circuito se comporta com o ajuste de i_{max}, foi executada outra simulação, mas dessa vez com uma imagem maior, onde fosse possível observar visualmente a qualidade da imagem formada. Da imagem Lena foi selecionado um bloco de tamanho 64×64, apresentado na Fig. 4.8.

Figura 4.8: Bloco da imagem Lena tamanho 64×64.

Antes de apresentar os resultados dessa nova simulação, uma análise sobre a dinâmica do circuto se faz necessária. Como visto nos Capítulos 2 e 3, o processamento realizado por um dado pixel depende diretamente do sinal propagado por seus vizinhos. O modo como os pixels estão interconectados faz com que a estabilização do sinal na saída de cada pixel se dê como uma onda que se propaga da esquerda para a direita, de cima para baixo. Essa propagação do sinal através do sensor de imagem faz com que haja um atraso entre o processamento do sinal do pixel superior esquerdo e do pixel inferior direito. Com isso, à medida que a imagem aumenta, o tempo para simular o seu processamento também aumenta.

As simulações transientes de 90 μs dos blocos 10×10 levaram, cada uma, em média quatro minutos para serem concluídas, mas as simulações do bloco 64×64 estavam consumindo um tempo muito maior, em torno de três semanas para atingir cerca de 60% da simulação. Além de todo esse tempo de simulação, o volume de dados gerados é muito grande. Isso fez com que a simulação fosse abortada antes do tempo estipulado pois o arquivo de saída, gerado pela versão utilizada do simulador *Spectre*, está limitado ao tamanho máximo de 2 GB. Com isso as primeiras simulações rodadas não apresentaram dados conclusivos. Os dados obtidos mostravam o estado do circuito muito antes que o mesmo atingisse o seu estado final.

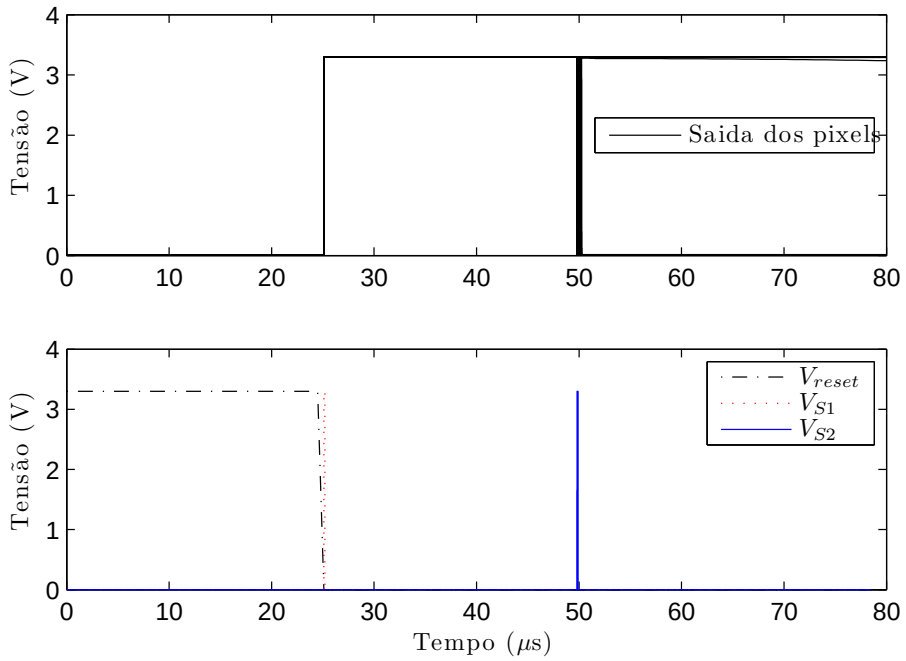
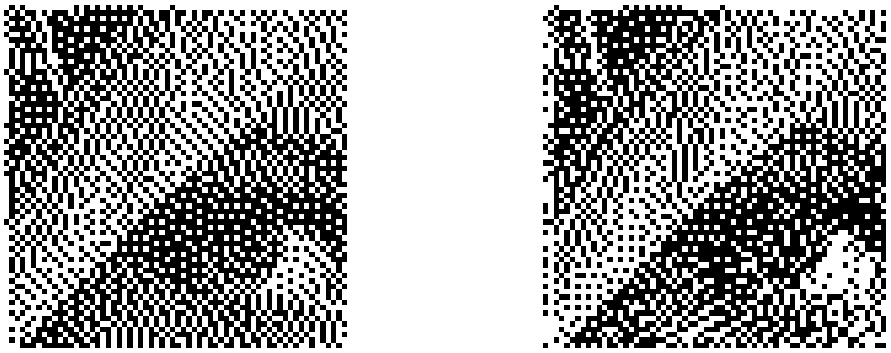


Figura 4.9: Dinâmica do circuito durante o processamento do bloco 10×10.

Para contornar estes problemas alguns parâmetros foram modificados. Por exemplo, o modo de simulação foi mudado de *conservative* para *liberal*, o que acelerou muito o processo, permitindo que a simulação atingisse os mesmos 60% em duas semanas, mas o volume de dados gerados não sofreu redução e a simulação continuou sendo abortada toda vez que o arquivo de saída atingia o seu limite.

Analisando o diagrama de tempo da simulação do bloco 10×10 na Fig. 4.9, vimos que o circuito atinge o estado final logo após ter sido completado o CDS, uma vez que a saída de todos os pixels se estabilizam rapidamente após o CDS (em 50 μs). No caso desta simulação e também da simulação dos blocos 5×5, os valores dos pixel foram lidos em 75 μs, o que garante que todos os resultados de simulações elétricas apresentados eram resultados do processamento completo dos blocos. Na simulação do bloco 64×64, ocorre extensa onda de propagação de sinais dentro do sensor, o que faz com que o circuito demore um tempo muito maior para atingir a estabilidade.

As primeiras simulações dos blocos 64×64 foram inconclusivas, pois os únicos dados obtidos eram relativos somente aos instantes posteriores ao chaveamento (circuito CDS), devido ao grande volume de dados gerados. Por esse motivo a simulação posteriormente foi configurada para salvar os dados somente a partir de 80 μs e o resultado é apresentado na Fig. 4.10. É possível notar que a imagem obtida na simulação elétrica possui irregularidades que não estão presentes na imagem gerada pelo MATLAB.



(a) (b)

Figura 4.10: Simulação do bloco 64 × 64: (a) algoritmo ideal no MATLAB e (b) simulação elétrica no SPICE.

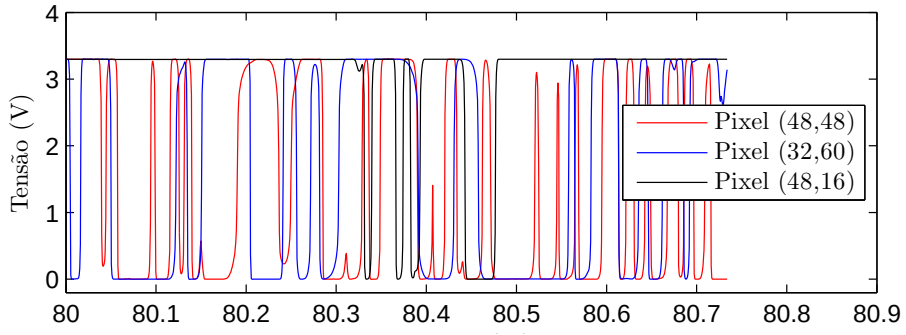


Figura 4.11: Comportamento dos pixels localizados no canto inferior-direito do bloco 64 × 64.

Observando as imagens, as irregularidades observadas aparentemente se concentram no canto inferior direito. Observando a dinâmica desses pixels (Fig. 4.11), constata-se que estes, mesmo após 30 μs de processamento (uma vez que o processamento começa aos 50 μs, após a realização do CDS), ainda não tiveram suas saídas estabilizadas, enquanto que os pixels localizados mais próximos do canto superior esquerdo da imagem já têm suas saídas estáveis (Fig. 4.12).

Como podemos ver pela Fig. 4.11, as saídas do circuito de processamento de cada pixel alteram seu estado de forma muito intensa ao longo do tempo. Isso se deve ao fato de que a saída de cada um desses circuitos é influenciada por quatro outros circuitos vizinhos, que são, por sua vez, influenciados, cada um, por quatro outros e assim sucessivamente. A Fig. 4.13 mostra o estado da saída de cada um dos 4096 pixels com relação à convergência dos mesmos, onde a região preta da figura indica que os pixels apresentavam saídas constantes após 80 μs, enquanto que os pixels da região branca não. Os dados utilizados para gerar esta figura correspondem a um tempo total de simulação inferior a 1 μs. Esse curto tempo se deve ao fato de o

volume de dados gerados ser muito grande. Mesmo iniciando a gravação dos dados apoś 80 μs de simulação, o arquivo de saída atinge 2 GB antes mesmo da simulação chegar a 81 μs.

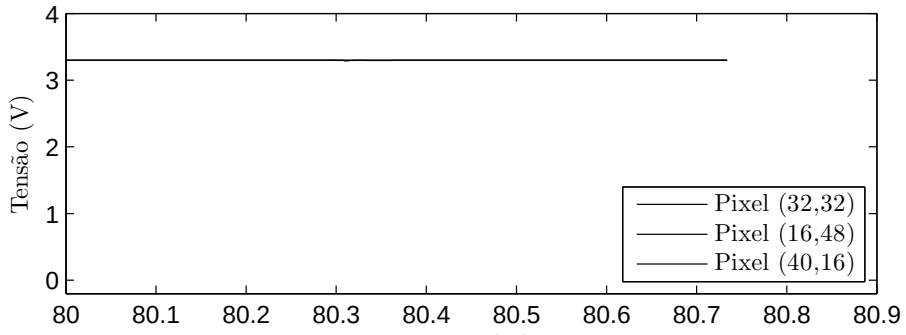


Figura 4.12: Comportamento dos pixels localizados no canto superior-esquerdo do bloco 64×64.

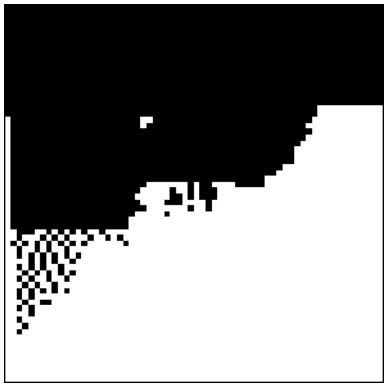


Figura 4.13: Os pixels na região escura estão estáveis após 80 μs de simulação e os na região clara ainda sofrem transições.

Uma análise um pouco mais detalhada da estabilidade da saída de cada pixel foi desenvolvida com o objetivo de tentar determinar por quanto tempo a simulação transiente deveria ser executada para que uma imagem estável pudesse ser obtida. No entanto, resultados inesperados não permitiram fazer qualquer estimativa sobre a duração da simulação para que este objetivo fosse alcançado. Os resultado parciais obtidos estão disponíveis no Apêndice B.

4.1 Análise de Monte Carlo

Executar uma simulação de Monte Carlo no simulador Spectre sobre o circuito proposto não fornecerá dados muito conclusivos, uma vez que nem todas as saídas

dos pixels alcançaram um estado estável e desta forma suas saídas ainda estarão variando no tempo ao fim das simulações. A fim de avaliar, mesmo que de forma grosseira, a robustez do circuito, foram realizadas simulações no MATLAB criando perturbações em alguns pontos do circuito, como uma tentativa de reproduzir efeitos de variação de parâmetros e processos. O código utilizado pode ser encontrado no Apêndice C.

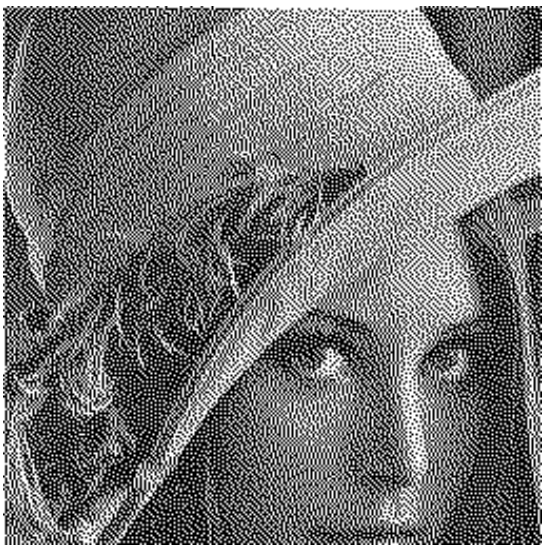
Como uma tentativa de simular o FPN entre pixels, cada multiplicador do filtro de Floyd-Steinberg foi perturbado por um erro distinto. Todos os erros são gaussianos, com média zero e com desvio padrão variando de 0.001 a 0.2. As Figs. 4.14 a 4.19 apresentam os resultados das simulações. Nas Figs. 4.17, 4.18 e 4.19 observamos que algumas imagens apresentam áreas escuras. O modo como o erro se manifestou nessas imagens parece ser o resultado de algum tipo de saturação aliado a algum tipo de comportamento caótico. É provável que este tipo de resultado não seja observado em uma simulação de Monte Carlo no Spectre.

As imagens apresentadas na Fig. 4.20 além das perturbações nos coeficientes dos filtros, têm perturbações também nos sinais de entrada $x(n)$, como uma tentativa de simular os erros cometidos pelo circuito de CDS, devido à injeção de carga provocada pelas chaves S1 e S2. Como podemos observar, as perturbações provocadas nos sinais de entrada em conjunto com as perturbações dos coeficientes dos filtros levaram as saídas dos pixels a um tipo de saturação, resultando em imagens finais muito diferentes da imagem desejada. Outras simulações, com erros maiores, não foram realizadas porque os resultados da Fig. 4.20 já apresentam resultados que indicam que o circuito é bastante sensível às perturbações provocadas.

(a)

(b)

(c)

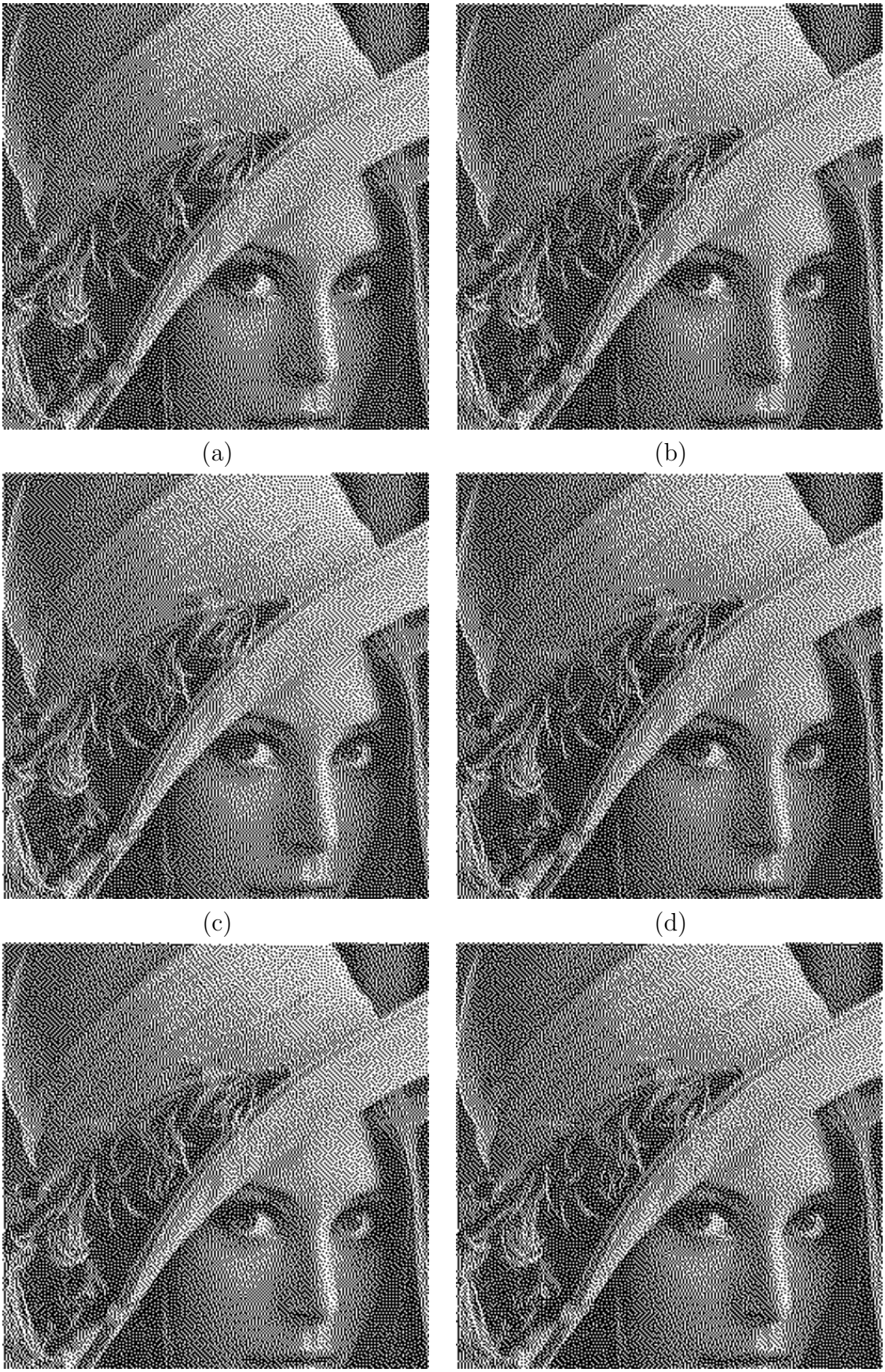


(d)

(e)

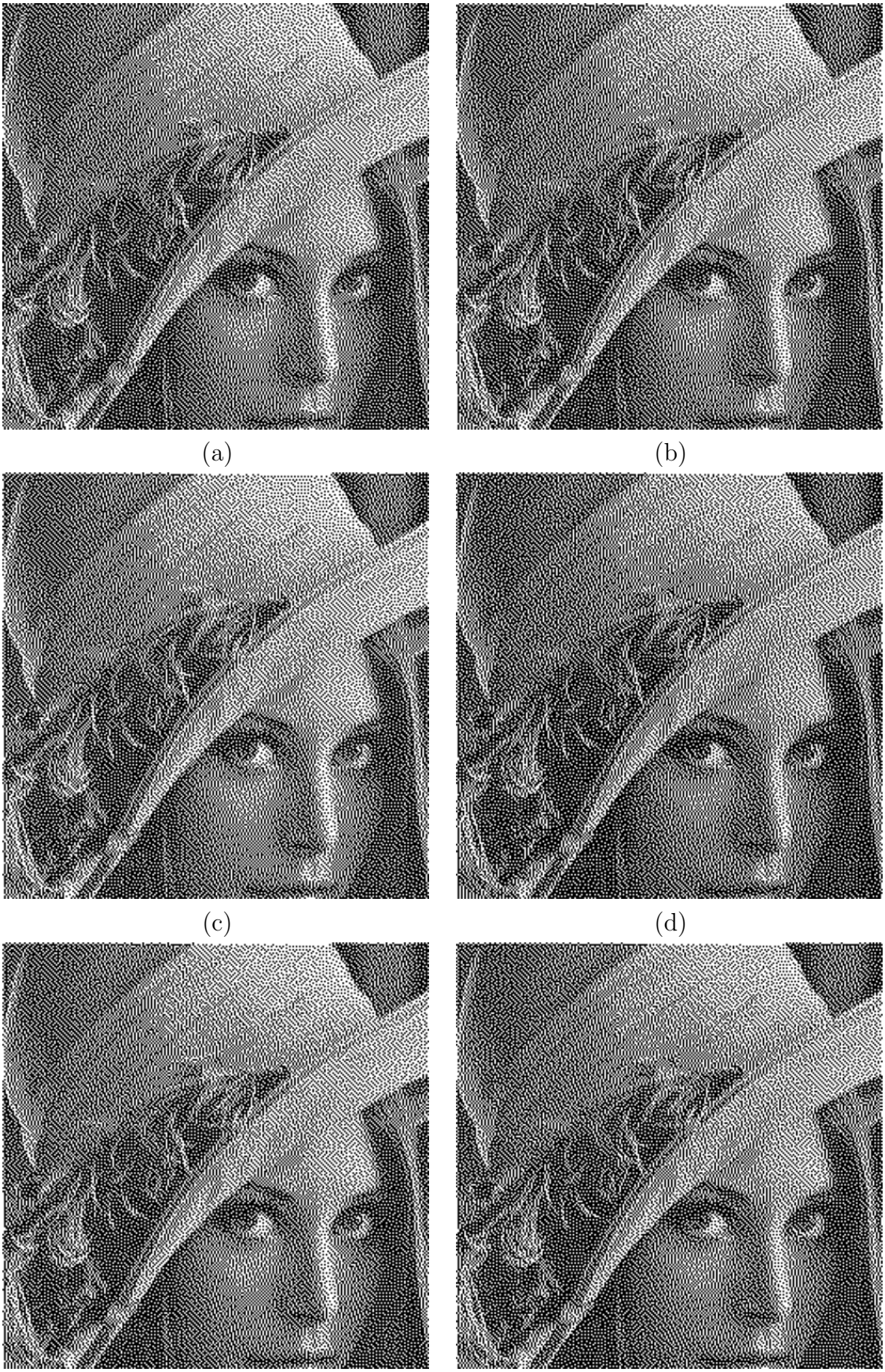
(f)

Figura 4.14: Resultado da simulação no MATLAB com perturbações, usando desvio padrão igual a 0.001, onde: (a) imagem original e as demais são o resultado das perturbações produzidas nos coeficientes dos filtros.



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

Figura 4.15: Resultado da simulação no MATLAB com perturbações, usando desvio padrão igual a 0.005, onde: (a) imagem original e as demais são o resultado das perturbações produzidas nos coeficientes dos filtros.



(a) (b)

(c) (d)

(e) (f)

Figura 4.16: Resultado da simulação no MATLAB com perturbações, usando desvio padrão igual a 0.01, onde: (a) imagem original e as demais são o resultado das perturbações produzidas nos coeficientes dos filtros.

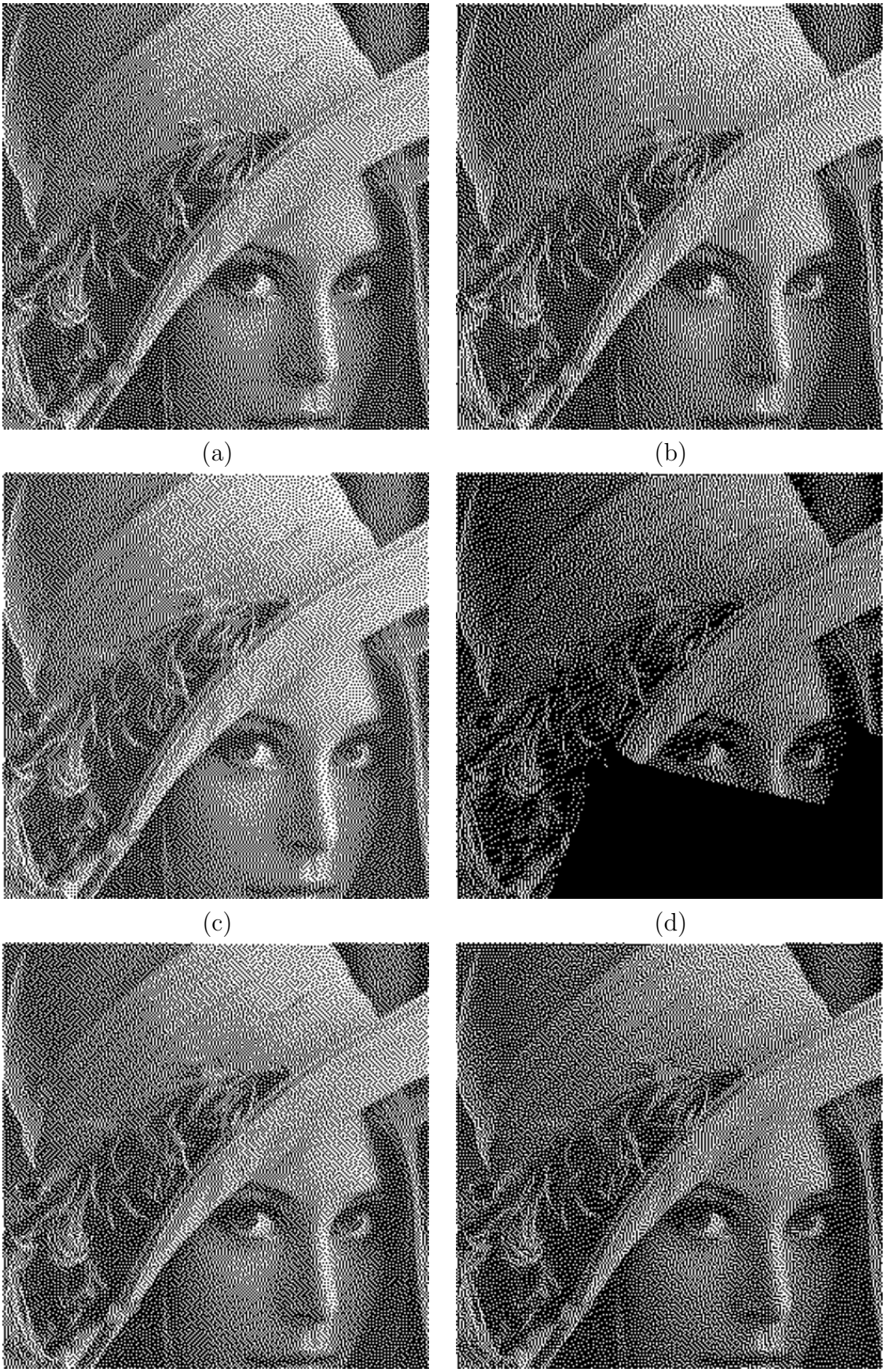
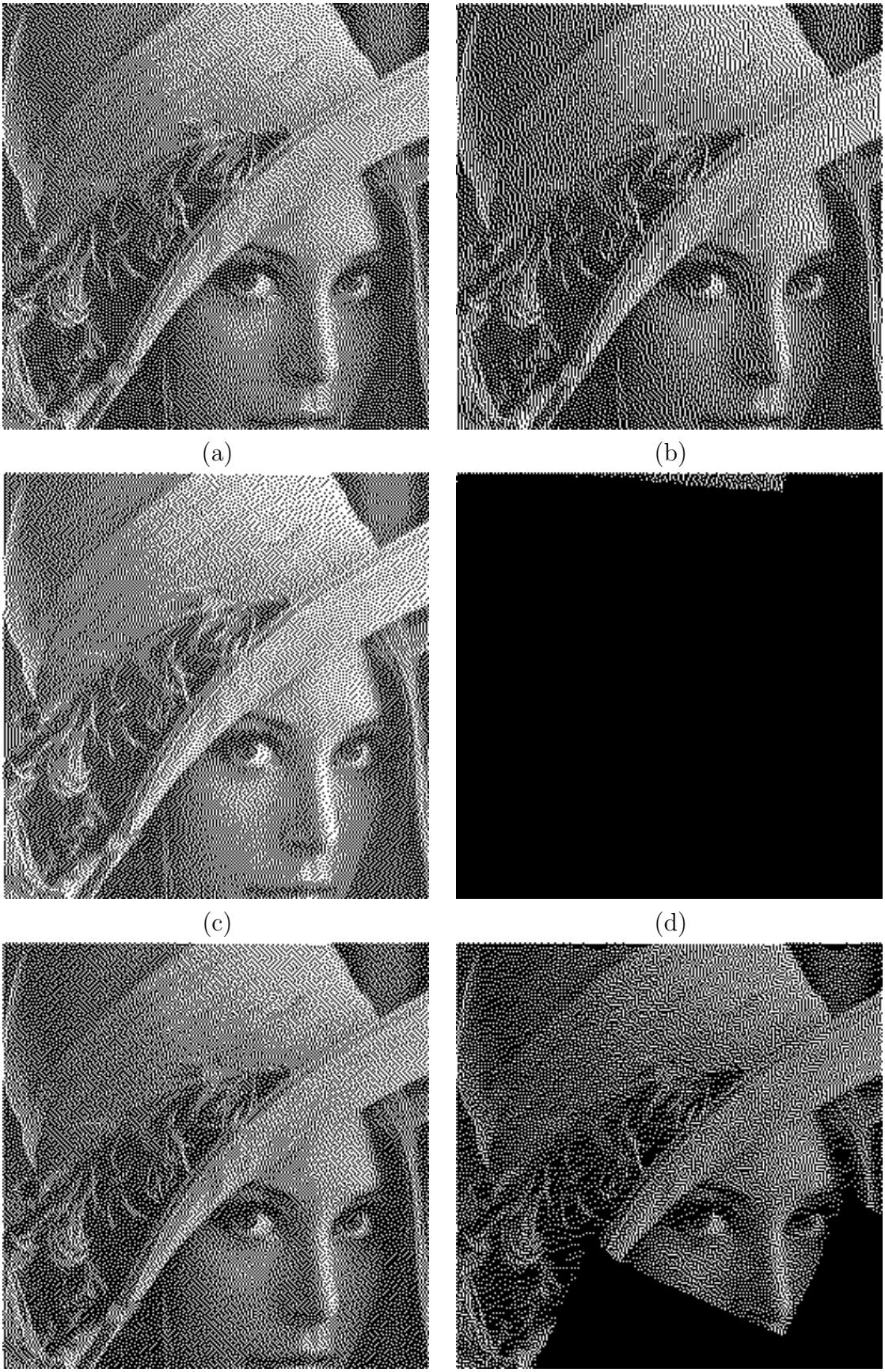


Figura 4.17: Resultado da simulação no MATLAB com perturbações, usando desvio padrão igual a 0.05, onde: (a) imagem original e as demais são o resultado das perturbações produzidas nos coeficientes dos filtros.



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

Figura 4.18: Resultado da simulação no MATLAB com perturbações, usando desvio padrão igual a 0.1, onde: (a) imagem original e as demais são o resultado das perturbações produzidas nos coeficientes dos filtros.

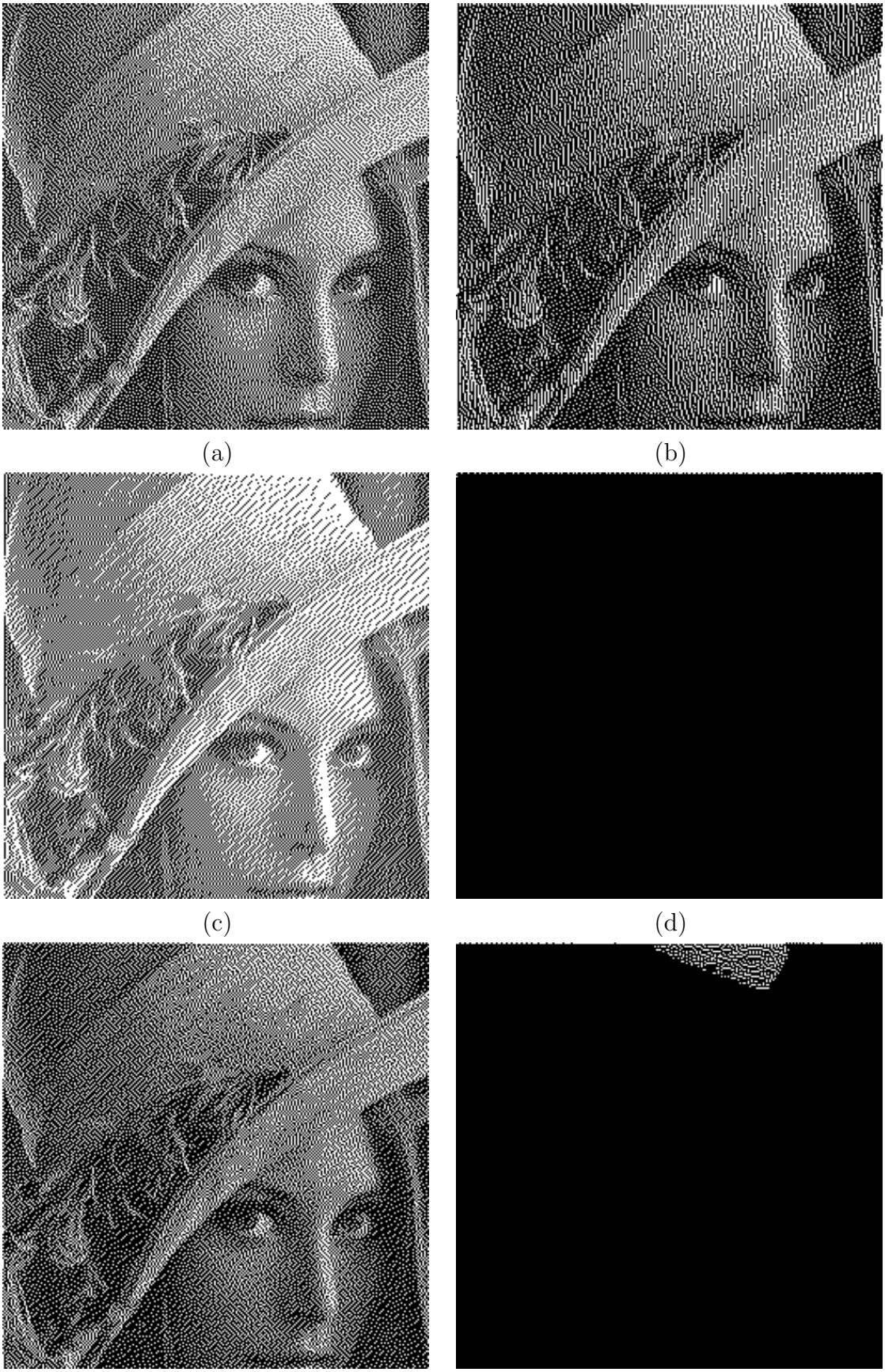


Figura 4.19: Resultado da simulação no MATLAB com perturbações, usando desvio padrão igual a 0.2, onde: (a) imagem original e as demais são o resultado das perturbações produzidas nos coeficientes dos filtros.

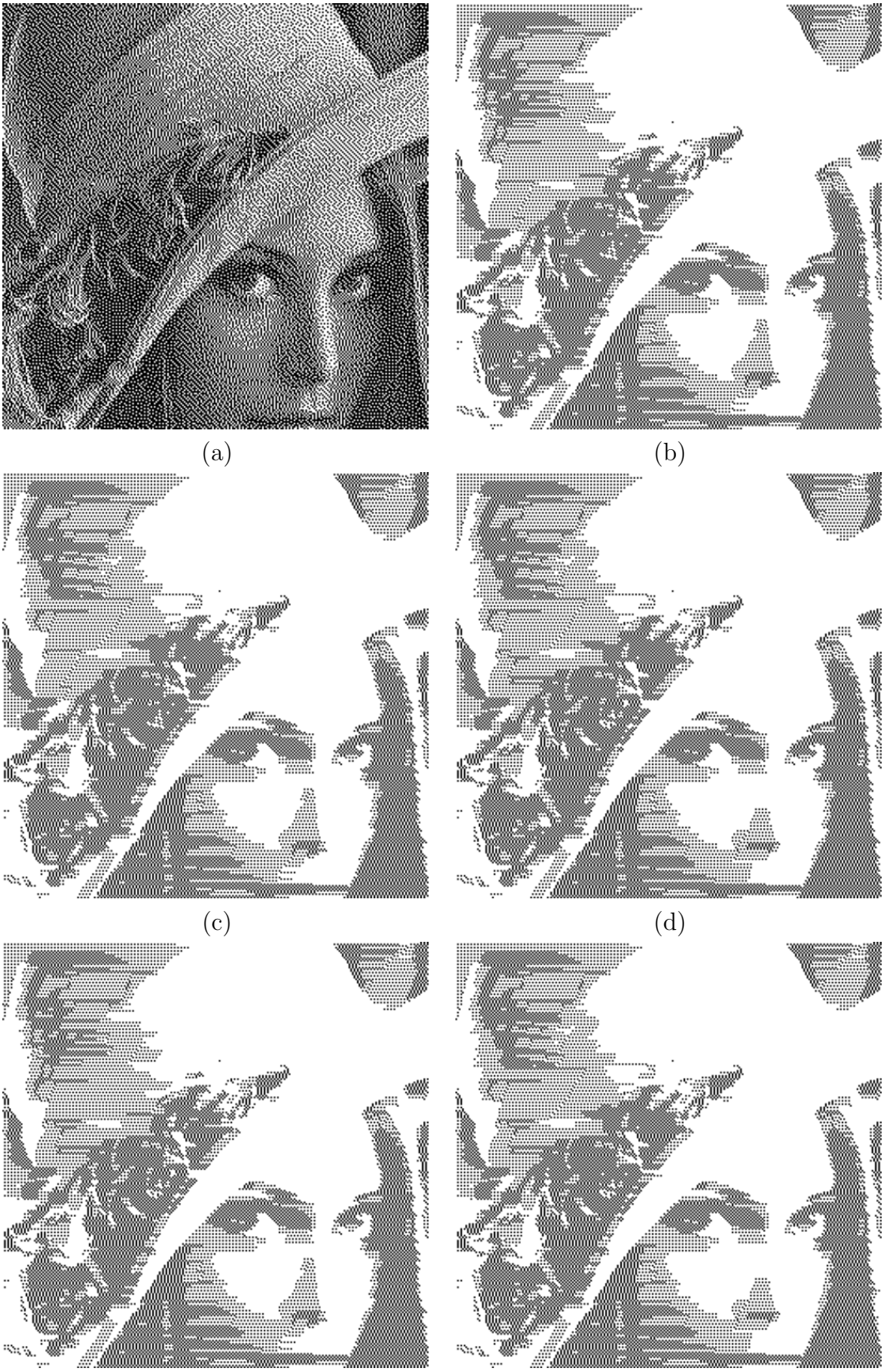


Figura 4.20: Resultado da simulação no MATLAB com perturbações, usando desvio padrão igual a 0.0001, onde: (a) imagem original e as demais são o resultado das perturbações produzidas nos coeficientes dos filtros.

Capítulo 5

Conclusões

Um circuito APS capaz de processar uma imagem no plano focal de sensores de imagem CMOS utilizando o algoritmo de *halftoning* foi projetado utilizando apenas circuitos que operam em modo de corrente, resultando em um circuito com 39 transistores por pixel.

A partir das simulações elétricas realizadas, pode-se observar a saturação dos pixels com as configurações propostas inicialmente e por esse motivo um ajuste foi realizado, reduzindo a corrente reversa máxima no fotodiodo, o que significa uma redução na faixa dinâmica de operação do circuito proposta inicialmente.

Os efeitos produzidos pelo modo como os pixel são interconectados se confirmaram. Os primeiros pixels a terem suas saídas estabilizadas são aqueles localizados próximos ao canto superior esquerdo da imagem e com o decorrer do tempo as saídas dos pixels se estabilizam em uma trajetória que remete a uma onda percorrendo a imagem em sentido diagonal, da esquerda para direita, de cima para baixo. Também é possível observar um atraso entre o processamento do pixel localizado no canto inferior direito em relação ao pixel localizado no canto superior esquerdo da imagem. Este atraso se mostrou maior que o esperado, o que não permitiu que imagens estáveis pudessem ser obtidas.

5.1 Trabalhos Futuros

Os resultados mostraram que uma investigação a respeito do tempo de simulação do circuito é necessária. É possível que essa enorme demanda de tempo para simular o circuito seja devida a alguma dificuldade de convergência do algoritmo de Newton-Raphson e não uma característica do circuito. Algumas análises, estudos e simulações que devem ser realizadas futuramente com o intuito de completar e melhorar este trabalho são:

- Executar a simulação do CDS separadamente, fazendo uma simulação para

calcular a corrente na saída do circuito de CDS de cada um dos pixels do circuito. Uma segunda simulação utilizaria o valor das correntes calculadas na primeira simulação como dados de entrada. É possível que dividir a simulação nessas duas etapas, o que consiste em menos cálculos por simulação, reduza o tempo utilizado para simular o circuito.

- Outra possibilidade consiste em seccionar o circuito em pedaços independentes, em blocos de tamanho 4×4 pixels por exemplo, armazenando os sinais a serem propagados para os pixels vizinhos para que eles sejam utilizados nas simulações dos blocos seguintes.

- Utilizar modelos simplificados para simular o fotodiodo e sua corrente reversa. Este modelo utiliza um capacitor externo para simular a capacitância de junção do fotodiodo. Essa simplificação pode acelerar a simulação.

- Analisar problemas relacionados a *Fixed Pattern Noise* (FPN), uma vez que cada sinal possui um circuito de processamento independente.

- Estudar a implementação de circuitos comuns entre pixels, isto é, verificar se é possível utilizar, por exemplo, um circuito de processamento em comum para todos os pixels de uma mesma coluna. Além de ajudar com o FPN, o circuito dedicado a cada pixel seria também minimizado, permitindo uma maior economia de área.

- Avaliar uma possível modificação nos pesos do filtro de Floyd-Steinberg de forma a torná-los mais adequados para implementação nos espelhos de corrente.

- Fazer uma simulação de Monte Carlo no simulador elétrico para avaliar o comportamento do circuito com relação a variações de parâmetros e de processos.

- Fazer o layout do circuito.

Referências Bibliográficas

[1] Imagem acessada em 31 de Março de 2012. Disponível em: <`http://sipi.usc.edu/database/`>.

[2] 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.

[3] ANASTASSIOU, D. "Error Diffusion Coding for A/D Conversion", *IEEE Transactions on Circuits and Systems*, v. 36, n. 9, pp. 1175–1186, Setembro de 1989.

[4] LEÓN-SALAS, W., 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] 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.

[6] 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 de 2005.

[7] LIN, Z., HOFFMAN, M., 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.

[8] CHI, Y., 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.

[9] 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.

[10] 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.

[11] 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.

[12] OLIVEIRA, F., HAAS, H., GOMES, J., et al. "A Circuit for Focal-Plane Image Compression using Vector Quantization", *10th International Symposium on Signals, Circuits and Systems*, pp. 1–4, Junho de 2011.

[13] EID, E.-S., FOSSUM, E. "Design of a CCD Focal-Plane Image Half-Toner". In: Blouke, M. (Ed.), *Proceedings SPIE 1242*, pp. 126–132, 1990.

[14] CROUNSE, K. R., ROSKA, T., CHUA, L. "Some Methods for Pratical Halftoning on the CNN Universal Machine", *1998 Fifth IEEE International Workshop on Cellular Neural Networks and Their Applications Proceedings*, pp. 337–342, Abril de 1998.

[15] BERNARD, T., GARDA, P., REICHART, A., et al. "Design of a Half-toning Integrated Circuit Based on Analog Quadratic Minimization by Non Linear Multistage Switched Capacitor Network", *IEEE International Symposium on Circuits and Systems*, v. 2, pp. 1217–1220, Junho de 1988.

[16] BAYER, B. E. "An Optimum Method for Two Level Rendition of Continuous-tone Pictures", *IEEE International Conference on Communications, Conference Record*, v. 1, n. 26, pp. 11–15, Junho de 1973.

[17] FLOYD, R. W., STEINBERG, L. "An adaptive algorithm for spatial gray-scale". In: *Proceedings Society Information Display*, v. 17(2), pp. 75–78, 1976.

[18] JARVIS, J., JUDICE, C., NINKE, W. "A Survey of Techniques for the Display of Continuous Tone Pictures on Bilevel Displays", *Computer Graphics and Image Processing*, v. 5, n. 1, pp. 13–40, Março de 1976.

[19] STUCKI, P. *MECCA-A Multiple-Error Correcting Computation Algorithm for Bilevel Image Hardcopy Reproduction.* Relatório Técnico RZ1060, Laboratório de Pesquisas da IBM, Zurich, Switzerland, 1981.

[20] KNOX, K. T. "Introduction to Digital Halftones". In: Eschbach, R. (Ed.), *Recent Progress in Digital Halftoning*, pp. 30–33. IS&T, 1994.

[21] SHIAU, J., FAN, Z. "A Set of Easily Implementable Coefficients in Error Diffusion with Reduced Worm Artifacts". In: *Proceedings SPIE*, v. 2658, pp. 222–225, 1996.

[22] WONG, P. W. "Error Diffusion with Dynamically Adjusted Kernel". In: Eschbach, R. (Ed.), *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, v. 5, pp. 113–116, Adelaide, Australia, 1994.

[23] ESCHBACH, R. "Reduction of Artifacts in Error Diffusion by Means of Input-Dependent Weights", *Journal of Eletronic Imaging*, v. 2, n. 4, pp. 352–358, Outubro de 1993.

[24] Imagem acessada em 31 de Março de 2012. Disponível em: <`http://www.csee.wvu.edu/~xinl/database.html`>.

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

[26] RAZAVI, B. *Design of Analog CMOS Integrated Circuits*. Primeira ed. New York, USA, McGraw-Hill, 2000.

[27] ITU-R. *Recommendation ITU-R BT.601-7*. International Telecommunication Union, Março de 2011. Disponível em: <`http://www.itu.int/rec/R-REC-BT.601/en`>.

[28] ITU-R. *Recommendation ITU-R BT.709-5*. International Telecommunication Union, Abril de 2002. Disponível em: <`http://www.itu.int/rec/R-REC-BT.709-5-200204-I/en`>.

Apêndice A

Simulação de Circuitos em Linha de Comando Utilizando o Cadence Spectre

Simular grandes circuitos utilizando a interface gráfica dos simuladores é uma tarefa bastante trabalhosa, em alguns casos, impraticável, principalmente em casos como o do circuto desenvolvido neste trabalho, que possui uma matriz com 4096 pixels, onde cada um deles está conectado a outros quatro.

Analisando o modo como o simulador Cadence Spectre realiza as simulações foi possível desenvolver um método alternativo de simulação que não envolve a interface gráfica do mesmo. A grosso modo, o que a interface gráfica do simulador faz quando pressionamos o botão para iniciar a simulação é gerar o *netlist* do circuito "desenhado"pelo usuário e criar um arquivo contendo o comando completo para simulação. Em seguida o processo de simulação é iniciado com a execução do comando via terminal.

O primeiro passo para realizar a simulação via terminal é a geração do *netlist*. Um programa em C++ foi desenvolvido para criar o *netlist* desejado, utilizando como base um *netlist* de um circuito 5×5, gerado pelo Cadence Spectre com o auxílio da interface gráfica. Este *netlist*, chamado *input.scs* é utilizado apenas como referência para a sintaxe e estrutura que o simulador utiliza. Outro arquivo de entrada utilizado pelo programa é um arquivo, chamado *correntesEntrada.txt*, contendo o valor (um valor por linha) das fontes de correntes (aquelas posicionadas em paralelo com o fotodiodo e que representam a luminosidade incidente em cada pixel) que são os dados de entrada do circuito. Portanto se o circuito tiver 100 pixels, este arquivo deverá conter 100 correntes, uma em cada linha para que o programa, da forma como foi escrito, possa criar o netlist corretamente.

Gerado o netlist basta executar em linha de comando a sintaxe abaixo para que

o circuito descrito no *netlist sensorAPS.scs* seja simulado.

```
spectre -env artist5.1.0 +escchars +log ./psf64x64/spectre.
    out -format psfascii -raw ./psf64x64 +lqtimeout 900 +
    param /cds/Opus/user/a370/spectre/ams_range.lmts
    sensorAPS.scs
```

É possível ainda criar um arquivo de lote (*batch*) para que sejam realizadas várias simulações, ou incluir simulações. Os dois arquivos de entrada e o código fonte do programa desenvolvido em C++ *criarNetlist.cpp* são apresentados a seguir.

`input.scs`

```
// Generated for: spectre
// Generated on: Mar  4 14:25:07 2011
// Design library name: COE810
// Design cell name: APStest
// Design view name: schematic

simulator lang=spectre
global 0 vdd!
include "/usr/local/Opus/user/tools/dfII/samples/artist/
    ahdlLib/quantity.spectre"
include "/cds/Opus/user/a370/spectre/c35/mcparams.scs"
include "/cds/Opus/user/a370/spectre/c35/cmos53.scs" section
    =cmostm
include "/cds/Opus/user/a370/spectre/c35/res.scs" section=
    restm
include "/cds/Opus/user/a370/spectre/c35/cap.scs" section=
    captm
include "/cds/Opus/user/a370/spectre/c35/bip.scs" section=
    biptm
include "/cds/Opus/user/a370/spectre/c35/ind.scs" section=
    indtm

// Library name: COE810
// Cell name: analogKey
// View name: schematic
subckt analogKey In Out S S\#
    MN3 (In S\# In 0) modn w=0.4u l=0.35u as=3.4e-13 ad=3.4e
        -13 ps=2.1u \
```

```
        pd=2.1u nrd=1.25 nrs=1.25 ng=1
    MN2 (Out S\# Out 0) modn w=0.4u l=0.35u as=3.4e-13 ad
        =3.4e-13 ps=2.1u \
        pd=2.1u nrd=1.25 nrs=1.25 ng=1
    MN0 (In S Out 0) modn w=0.8u l=0.35u as=6.8e-13 ad=6.8e
        -13 ps=2.5u \
        pd=2.5u nrd=0.625 nrs=0.625 ng=1
ends analogKey
// End of subcircuit definition.

// Library name: COE810
// Cell name: PixelCell
// View name: schematic
subckt PixelCell _1_16 _3_16 _5_16 _7_16 CDS_1 CDS_1\# CDS_2
    CDS_2\# Gnd \
        Iph V_diff Vdd Vreset Y
    D0 (Gnd Iph) nd area=100p perimeter=40u m=1
    I22 (net70 net113 Y Y\#) analogKey
    I16 (net74 Gnd) isource dc=5u type=dc
    MN23 (net0235 CDS_1 net0216 Gnd) modn w=0.8u l=0.35u as
        =6.8e-13 \
        ad=6.8e-13 ps=2.5u pd=2.5u nrd=0.625 nrs=0.625 ng=1
    MN24 (net0235 CDS_1\# net0235 Gnd) modn w=0.4u l=0.35u
        as=3.4e-13 \
        ad=3.4e-13 ps=2.1u pd=2.1u nrd=1.25 nrs=1.25 ng=1
    MN20 (net0235 CDS_2 net0220 Gnd) modn w=0.8u l=0.35u as
        =6.8e-13 \
        ad=6.8e-13 ps=2.5u pd=2.5u nrd=0.625 nrs=0.625 ng=1
    MN22 (net0235 CDS_2\# net0235 Gnd) modn w=0.4u l=0.35u
        as=3.4e-13 \
        ad=3.4e-13 ps=2.1u pd=2.1u nrd=1.25 nrs=1.25 ng=1
    MN25 (net0216 CDS_1\# net0216 Gnd) modn w=0.4u l=0.35u
        as=3.4e-13 \
        ad=3.4e-13 ps=2.1u pd=2.1u nrd=1.25 nrs=1.25 ng=1
    MN21 (net0220 CDS_2\# net0220 Gnd) modn w=0.4u l=0.35u
        as=3.4e-13 \
        ad=3.4e-13 ps=2.1u pd=2.1u nrd=1.25 nrs=1.25 ng=1
    MN13 (net141 net0216 Gnd Gnd) modn w=5u l=5u as=4.25e-12
        ad=4.25e-12 \
```

```
    ps=6.7u pd=6.7u nrd=0.1 nrs=0.1 ng=1
MN14 (net0305 net0220 Gnd Gnd) modn w=5u l=5u as=4.25e
   -12 ad=4.25e-12 \
    ps=6.7u pd=6.7u nrd=0.1 nrs=0.1 ng=1
MN15 (net0235 net0235 Gnd Gnd) modn w=5u l=5u as=4.25e
   -12 ad=4.25e-12 \
    ps=6.7u pd=6.7u nrd=0.1 nrs=0.1 ng=1
MN16 (Vdd Vreset Iph Gnd) modn w=1u l=0.35u as=8.5e-13
   ad=8.5e-13 \
    ps=2.7u pd=2.7u nrd=0.5 nrs=0.5 ng=1
MN8 (net141 _1_16 Gnd Gnd) modn w=0.5u l=1u as=4.25e-13
   ad=4.25e-13 \
    ps=2.2u pd=2.2u nrd=1 nrs=1 ng=1
MN9 (net141 _3_16 Gnd Gnd) modn w=1.5u l=1u as=1.275e-12
    ad=1.275e-12 \
    ps=3.2u pd=3.2u nrd=0.333333 nrs=0.333333 ng=1
MN10 (net141 _5_16 Gnd Gnd) modn w=2.5u l=1u as=2.125e
   -12 ad=2.125e-12 \
    ps=4.2u pd=4.2u nrd=0.2 nrs=0.2 ng=1
MN11 (net141 _7_16 Gnd Gnd) modn w=3.5u l=1u as=2.975e
   -12 ad=2.975e-12 \
    ps=5.2u pd=5.2u nrd=0.142857 nrs=0.142857 ng=1
MN6 (net85 net112 net113 Gnd) modn w=1.4u l=1u as=1.19e
   -12 ad=1.19e-12 \
    ps=3.1u pd=3.1u nrd=0.357143 nrs=0.357143 ng=1
MN7 (net112 net113 Gnd Gnd) modn w=1u l=1u as=8.5e-13 ad
   =8.5e-13 \
    ps=2.7u pd=2.7u nrd=0.5 nrs=0.5 ng=1
MN5 (V_diff V_diff Gnd Gnd) modn w=8u l=1u as=6.8e-12 ad
   =6.8e-12 \
    ps=9.7u pd=9.7u nrd=0.0625 nrs=0.0625 ng=1
MN4 (net113 net142 Gnd Gnd) modn w=1u l=2u as=8.5e-13 ad
   =8.5e-13 \
    ps=2.7u pd=2.7u nrd=0.5 nrs=0.5 ng=1
MN3 (net142 net142 Gnd Gnd) modn w=1u l=2u as=8.5e-13 ad
   =8.5e-13 \
    ps=2.7u pd=2.7u nrd=0.5 nrs=0.5 ng=1
MN2 (net97 Y\# net141 Gnd) modn w=1.4u l=1u as=1.19e-12
   ad=1.19e-12 \
```

```
     ps=3.1u pd=3.1u nrd=0.357143 nrs=0.357143 ng=1
MN1 (Y Y\# Gnd Gnd) modn w=1u l=1u as=8.5e-13 ad=8.5e-13
     ps=2.7u \
     pd=2.7u nrd=0.5 nrs=0.5 ng=1
MN0 (Y\# net141 Gnd Gnd) modn w=1u l=1u as=8.5e-13 ad
    =8.5e-13 ps=2.7u \
     pd=2.7u nrd=0.5 nrs=0.5 ng=1
MP12 (net0305 net0305 Vdd Vdd) modp w=1u l=2u as=8.5e-13
     ad=8.5e-13 \
     ps=2.7u pd=2.7u nrd=0.5 nrs=0.5 ng=1
MP13 (net141 net0305 Vdd Vdd) modp w=1u l=2u as=8.5e-13
    ad=8.5e-13 \
     ps=2.7u pd=2.7u nrd=0.5 nrs=0.5 ng=1
MP14 (net0235 Iph Vdd Vdd) modp w=1u l=0.35u as=8.5e-13
    ad=8.5e-13 \
     ps=2.7u pd=2.7u nrd=0.5 nrs=0.5 ng=1
MP9 (net112 net113 Vdd Vdd) modp w=3u l=1u as=2.55e-12
    ad=2.55e-12 \
     ps=4.7u pd=4.7u nrd=0.166667 nrs=0.166667 ng=1
MP10 (V_diff net112 net113 Vdd) modp w=4u l=1u as=3.4e
    -12 ad=3.4e-12 \
     ps=5.7u pd=5.7u nrd=0.125 nrs=0.125 ng=1
MP8 (V_diff net85 Vdd Vdd) modp w=1u l=2u as=8.5e-13 ad
    =8.5e-13 \
     ps=2.7u pd=2.7u nrd=0.5 nrs=0.5 ng=1
MP7 (net85 net85 Vdd Vdd) modp w=1u l=2u as=8.5e-13 ad
    =8.5e-13 ps=2.7u \
     pd=2.7u nrd=0.5 nrs=0.5 ng=1
MP6 (net74 net74 Vdd Vdd) modp w=1u l=3.4u as=8.5e-13 ad
    =8.5e-13 \
     ps=2.7u pd=2.7u nrd=0.5 nrs=0.5 ng=1
MP5 (net70 net74 Vdd Vdd) modp w=1u l=3.4u as=8.5e-13 ad
    =8.5e-13 \
     ps=2.7u pd=2.7u nrd=0.5 nrs=0.5 ng=1
MP4 (Y Y\# Vdd Vdd) modp w=3u l=1u as=2.55e-12 ad=2.55e
    -12 ps=4.7u \
     pd=4.7u nrd=0.166667 nrs=0.166667 ng=1
MP3 (Y\# net141 Vdd Vdd) modp w=3u l=1u as=2.55e-12 ad
    =2.55e-12 \
```

```
        ps=4.7u pd=4.7u nrd=0.166667 nrs=0.166667 ng=1
    MP2 (net142 Y\# net141 Vdd) modp w=4u l=1u as=3.4e-12 ad
        =3.4e-12 \
        ps=5.7u pd=5.7u nrd=0.125 nrs=0.125 ng=1
    MP1 (net97 net97 Vdd Vdd) modp w=1u l=2u as=8.5e-13 ad
        =8.5e-13 ps=2.7u \
        pd=2.7u nrd=0.5 nrs=0.5 ng=1
    MP0 (net142 net97 Vdd Vdd) modp w=1u l=2u as=8.5e-13 ad
        =8.5e-13 \
        ps=2.7u pd=2.7u nrd=0.5 nrs=0.5 ng=1
ends PixelCell
// End of subcircuit definition.

// Library name: COE810
// Cell name: APStest
// View name: schematic
V3 (CDS_1\# 0) vsource type=pulse val0=3.3 val1=0 period=1
   delay=25.09u \
        rise=10n fall=10n width=100n
V4 (CDS_2 0) vsource type=pulse val0=0 val1=3.3 period=1
   delay=49.79u \
        rise=10n fall=10n width=100n
V1 (Vreset 0) vsource type=pulse val0=0 val1=3.3 period=1
   delay=0 rise=10n \
        fall=10n width=24.99n
V2 (CDS_1 0) vsource type=pulse val0=0 val1=3.3 period=1
   delay=25.09u \
        rise=10n fall=10n width=100n
V5 (CDS_2\# 0) vsource type=pulse val0=3.3 val1=0 period=1
   delay=49.79u \
        rise=10n fall=10n width=100n
V0 (vdd! 0) vsource dc=3.3 type=dc
I29 (net243 0) isource dc=Iph_22 type=dc
I31 (net221 0) isource dc=Iph_13 type=dc
I35 (net177 0) isource dc=Iph_24 type=dc
I26 (net265 0) isource dc=Iph_11 type=dc
I45 (net34 0) isource dc=Iph_41 type=dc
I30 (net210 0) isource dc=Iph_23 type=dc
I25 (net254 0) isource dc=Iph_12 type=dc
```

```
I39 (net155 0) isource dc=Iph_34 type=dc
I27 (net232 0) isource dc=Iph_21 type=dc
I43 (net23 0) isource dc=Iph_43 type=dc
I41 (net12 0) isource dc=Iph_45 type=dc
I48 (net100 0) isource dc=Iph_53 type=dc
I42 (net1 0) isource dc=Iph_44 type=dc
I40 (net122 0) isource dc=Iph_35 type=dc
I49 (net144 0) isource dc=Iph_54 type=dc
I38 (net111 0) isource dc=Iph_33 type=dc
I46 (net89 0) isource dc=Iph_51 type=dc
I36 (net56 0) isource dc=Iph_31 type=dc
I34 (net188 0) isource dc=Iph_25 type=dc
I50 (net133 0) isource dc=Iph_55 type=dc
I47 (net78 0) isource dc=Iph_52 type=dc
I44 (net45 0) isource dc=Iph_42 type=dc
I37 (net67 0) isource dc=Iph_32 type=dc
I33 (net199 0) isource dc=Iph_15 type=dc
I32 (net166 0) isource dc=Iph_14 type=dc
I24 (Iout_33 Iout_35 Iout_34 Iout_43 CDS_1 CDS_1\# CDS_2
   CDS_2\# 0 net1 \
        Iout_44 vdd! Vreset Y_44) PixelCell
I23 (Iout_34 Gnd Iout_35 Iout_44 CDS_1 CDS_1\# CDS_2 CDS_2\#
    0 net12 \
        Iout_45 vdd! Vreset Y_45) PixelCell
I22 (Iout_32 Iout_34 Iout_33 Iout_42 CDS_1 CDS_1\# CDS_2
   CDS_2\# 0 net23 \
        Iout_43 vdd! Vreset Y_43) PixelCell
I21 (Gnd Iout_32 Iout_31 Gnd CDS_1 CDS_1\# CDS_2 CDS_2\# 0
   net34 Iout_41 \
        vdd! Vreset Y_41) PixelCell
I20 (Iout_31 Iout_33 Iout_32 Iout_41 CDS_1 CDS_1\# CDS_2
   CDS_2\# 0 net45 \
        Iout_42 vdd! Vreset Y_42) PixelCell
I19 (Gnd Iout_22 Iout_21 Gnd CDS_1 CDS_1\# CDS_2 CDS_2\# 0
   net56 Iout_31 \
        vdd! Vreset Y_31) PixelCell
I18 (Iout_21 Iout_23 Iout_22 Iout_31 CDS_1 CDS_1\# CDS_2
   CDS_2\# 0 net67 \
        Iout_32 vdd! Vreset Y_32) PixelCell
```

```
I17 (Iout_41 Iout_43 Iout_42 Iout_51 CDS_1 CDS_1\# CDS_2
   CDS_2\# 0 net78 \
        Iout_52 vdd! Vreset Y_52) PixelCell
I16 (Gnd Iout_42 Iout_41 Gnd CDS_1 CDS_1\# CDS_2 CDS_2\# 0
   net89 Iout_51 \
        vdd! Vreset Y_51) PixelCell
I15 (Iout_42 Iout_44 Iout_43 Iout_52 CDS_1 CDS_1\# CDS_2
   CDS_2\# 0 net100 \
        Iout_53 vdd! Vreset Y_53) PixelCell
I14 (Iout_22 Iout_24 Iout_23 Iout_32 CDS_1 CDS_1\# CDS_2
   CDS_2\# 0 net111 \
        Iout_33 vdd! Vreset Y_33) PixelCell
I13 (Iout_24 Gnd Iout_25 Iout_34 CDS_1 CDS_1\# CDS_2 CDS_2\#
    0 net122 \
        Iout_35 vdd! Vreset Y_35) PixelCell
I12 (Iout_44 Gnd Iout_45 Iout_54 CDS_1 CDS_1\# CDS_2 CDS_2\#
    0 net133 \
        Iout_55 vdd! Vreset Y_55) PixelCell
I11 (Iout_43 Iout_45 Iout_44 Iout_53 CDS_1 CDS_1\# CDS_2
   CDS_2\# 0 net144 \
        Iout_54 vdd! Vreset Y_54) PixelCell
I10 (Iout_23 Iout_25 Iout_24 Iout_33 CDS_1 CDS_1\# CDS_2
   CDS_2\# 0 net155 \
        Iout_34 vdd! Vreset Y_34) PixelCell
I9 (Gnd Gnd Gnd Iout_13 CDS_1 CDS_1\# CDS_2 CDS_2\# 0 net166
    Iout_14 vdd! \
        Vreset Y_14) PixelCell
I8 (Iout_13 Iout_15 Iout_14 Iout_23 CDS_1 CDS_1\# CDS_2
   CDS_2\# 0 net177 \
        Iout_24 vdd! Vreset Y_24) PixelCell
I7 (Iout_14 Gnd Iout_15 Iout_24 CDS_1 CDS_1\# CDS_2 CDS_2\#
   0 net188 \
        Iout_25 vdd! Vreset Y_25) PixelCell
I6 (Gnd Gnd Gnd Iout_14 CDS_1 CDS_1\# CDS_2 CDS_2\# 0 net199
    Iout_15 vdd! \
        Vreset Y_15) PixelCell
I5 (Iout_12 Iout_14 Iout_13 Iout_22 CDS_1 CDS_1\# CDS_2
   CDS_2\# 0 net210 \
        Iout_23 vdd! Vreset Y_23) PixelCell
```

```
I4 (Gnd Gnd Gnd Iout_12 CDS_1 CDS_1\# CDS_2 CDS_2\# 0 net221
    Iout_13 vdd! \
        Vreset Y_13) PixelCell
I3 (Gnd Iout_12 Iout_11 Gnd CDS_1 CDS_1\# CDS_2 CDS_2\# 0
   net232 Iout_21 \
        vdd! Vreset Y_21) PixelCell
I2 (Iout_11 Iout_13 Iout_12 Iout_21 CDS_1 CDS_1\# CDS_2
   CDS_2\# 0 net243 \
        Iout_22 vdd! Vreset Y_22) PixelCell
I1 (Gnd Gnd Gnd Iout_11 CDS_1 CDS_1\# CDS_2 CDS_2\# 0 net254
    Iout_12 vdd! \
        Vreset Y_12) PixelCell
I0 (Gnd Gnd Gnd Gnd CDS_1 CDS_1\# CDS_2 CDS_2\# 0 net265
   Iout_11 vdd! \
        Vreset Y_11) PixelCell

simulatorOptions options reltol=100e-6 vabstol=1e-6 iabstol
   =1e-12 temp=27 \
    tnom=27 homotopy=all limit=delta scalem=1.0 scale=1.0 \
    compatible=spice2 gmin=1e-12 rforce=1 maxnotes=5
       maxwarns=5 digits=5 \
    cols=80 pivrel=1e-3 ckptclock=1800 sensfile="../psf/sens
       .output" \
    checklimitdest=psf
tran tran stop=100u errpreset=conservative write="spectre.ic
   " \
    writefinal="spectre.fc" annotate=status maxiters=5
finalTimeOP info what=oppoint where=rawfile
designParamVals info what=parameters where=rawfile
primitives info what=primitives where=rawfile
subckts info what=subckts  where=rawfile

save I0.MP14:s
saveOptions options save=allpub
```

correntesEntrada.txt

```
// Este arquivo deve conter apenas os numeros, e por ser um
   arquivo muito simples apenas algumas linhas sao
   apresentadas a titulo de demonstracao.
```

```
8.09803921568627e-11
1.00196078431373e-10
1.22156862745098e-10
1.01568627450980e-10
1.61960784313725e-10

 ...
```

criarNetlist.cpp

```
/* Programa desenvolvido para gerar um netlist com uma
   matriz de pixels utilizada para simular o circuito APS
   com processamento por halftoning */


#include <stdlib.h>
#include <string.h>
#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,nlines,ncolumns;
  char saux[256];

  // Teste do numero de parametros da linha de comando.
  if (argc < 6)
  {
    cout<< argv[0] << " <netlist modelo> <netlist saida> <
       arquivo correntes> <n linhas> <n colunas>"<< endl;
    return (0);
  }
```

```
// Abre o netlist modelo. A partir desse arquivo sera
   criado o novo netlist, alterando apenas os parametros
   especificados.
ifile.open(argv[1],ios::in);
if (!ifile.is_open())
{
  cout<< "Arquivo "<< argv[1] <<" nao pode ser aberto" <<
     endl;
  return(0);
}

// Cria o novo arquivo netlist com os parametros desejados
   .
ofile.open(argv[2],ios::in);
if (!ofile.is_open())
{
  cout<< "O netlist " << arg[2] << " nao pode ser criado"
     << endl;
  return(0);
}

// Abre o arquivo com as correntes de entrada dos pixels.
input.open(argv[2],ios::in);
if ( !input.is_open() )
{
  cout<< "O arquivo " << arg[3] << " nao pode ser aberto"
     << endl;
  return(0);
}

//O loop abaixo percorre o arquivo ate encontrar a linha
   contendo a string "// Cell name: APStest". A partir
   desse ponto no netlist serao feitas modificacoes.
n = 0;
while (ifile.getline(saux,256) && (s.compare("// Cell name
   : APStest")))
{
  s = saux;
  ofile << s << endl;
```

```
    s = s.substr(0,21); //Necessario fazer isso porque
        estava sendo inserido um caracter oculto no final da
        linha. Assim limito a string apenas ao texto desejado
        . O numero "21" determina a quantidade de caracteres
        da string procurada.
}
s=saux;

// Adicao das fontes de alimentacao e de sinal desejadas
ofile<<"// View name: schematic"<<endl;
ofile<<"V1 (CDS_1\\# 0) vsource type=pulse val0=3.3 val1=0
     period=1 delay=25.09u rise=10n fall=10n width=100n"<<
    endl;
ofile<<"V2 (CDS_2 0) vsource type=pulse val0=0 val1=3.3
    period=1 delay=49.79u rise=10n fall=10n width=100n"<<
    endl;
ofile<<"V3 (Vreset 0) vsource type=pulse val0=0 val1=3.3
    period=1 delay=0 rise=10n fall=10n width=24.99n"<<endl;
ofile<<"V4 (CDS_1 0) vsource type=pulse val0=0 val1=3.3
    period=1 delay=25.09u rise=10n fall=10n width=100n"<<
    endl;
ofile<<"V5 (CDS_2\\# 0) vsource type=pulse val0=3.3 val1=0
     period=1 delay=49.79u rise=10n fall=10n width=100n"<<
    endl;
ofile<<"V6 (vdd! 0) vsource dc=3.3 type=dc"<<endl<<endl;

// Numero de linhas e colunas, lidos dos parametros na
    linha de comando. Utilizados para criar a matriz de
    pixel com a dimensao desejada.
nlines = atoi(argv[2]);
ncolumns = atoi(argv[3]);

// Faz a leitura do arquivo de correntes de entrada dos
    pixels, e adiciona fontes de corrente com esses valores
     no netlist segundo o modelo : "I29 (net243 0) isource
    dc=Iph_22 type=dc".
s="";
n = 1;
for (i=1; i<=nlines; i++)
```

```
{
  for (j=1;j<=ncolumns;j++)
  {
    input>>s;
    ofile <<"I_"<<n<<" (I_"<<i<<"_"<<j<<" 0) isource dc="
      <<s<<" type=dc"<<endl;
    n++;
  }
}
ofile<<endl;

// Adiciona um pixel (definido no netlist modelo e ja
   copiado para esse novo netlist durante a busca pela
   string "// Cell name: APStest".
// Modelo: I24 (Iout_33 Iout_35 Iout_34 Iout_43 CDS_1
   CDS_1\# CDS_2 CDS_2\# 0 net1 Iout_44 vdd! Vreset Y_44)
   PixelCell
// Os testes realizados tem a funcao de analisar a posicao
    do pixel dentro da matriz e a partir dai determinar
   quais correntes serao utilizadas pelo filtro do
   algoritmo de Floyd-Steinberg.
for (i=1; i<=nlines; i++)
{
  for (j=1;j<=ncolumns;j++)
  {
    if (j==1)
    {
      if (i==1)
      {
        ofile <<"P_"<<i<<"_"<<j<<" (0 0 0 0 CDS_1 CDS_1\\#
            CDS_2 CDS_2\\# 0 I_"<<i<<"_"<<j<<" Iout_"<<i<<
          "_"<<j<<" vdd! Vreset Y_"<<i<<"_"<<j<<")
           PixelCell"<<endl;
      }
      else
      {
        ofile <<"P_"<<i<<"_"<<j<<" (0 Iout_"<<i-1<<"_"<<j
          +1<<" Iout_"<<i-1<<"_"<<j<<" 0 CDS_1 CDS_1\\#
          CDS_2 CDS_2\\# 0 I_"<<i<<"_"<<j<<" Iout_"<<i<<"
```

```
            _"<<j<<" vdd! Vreset Y_"<<i<<"_"<<j<<")
            PixelCell"<<endl;
        }
      }
      else if (j == ncolumns)
      {
        if (i==1)
        {
          ofile <<"P_"<<i<<"_"<<j<<" (0 0 0 Iout_"<<i<<"_"<<
            j-1<<" CDS_1 CDS_1\\# CDS_2 CDS_2\\# 0 I_"<<i<<
            "_"<<j<<" Iout_"<<i<<"_"<<j<<" vdd! Vreset Y_"
            <<i<<"_"<<j<<") PixelCell"<<endl;
        }
        else
        {
          ofile <<"P_"<<i<<"_"<<j<<" (Iout_"<<i-1<<"_"<<j
            -1<<" 0 Iout_"<<i-1<<"_"<<j<<" Iout_"<<i<<"_"<<
            j-1<<" CDS_1 CDS_1\\# CDS_2 CDS_2\\# 0 I_"<<i<<
            "_"<<j<<" Iout_"<<i<<"_"<<j<<" vdd! Vreset Y_"
            <<i<<"_"<<j<<") PixelCell"<<endl;
        }
      }
      else if (i==1)
      {
        ofile <<"P_"<<i<<"_"<<j<<" (0 0 0 Iout_"<<i<<"_"<<j
          -1<<" CDS_1 CDS_1\\# CDS_2 CDS_2\\# 0 I_"<<i<<"_"
          <<j<<" Iout_"<<i<<"_"<<j<<" vdd! Vreset Y_"<<i<<"
          _"<<j<<") PixelCell"<<endl;
      }
      else
      {
        ofile <<"P_"<<i<<"_"<<j<<" (Iout_"<<i-1<<"_"<<j-1<<"
            Iout_"<<i-1<<"_"<<j+1<<" Iout_"<<i-1<<"_"<<j<<"
          Iout_"<<i<<"_"<<j-1<<" CDS_1 CDS_1\\# CDS_2 CDS_2
          \\# 0 I_"<<i<<"_"<<j<<" Iout_"<<i<<"_"<<j<<" vdd!
            Vreset Y_"<<i<<"_"<<j<<") PixelCell"<<endl;
      }
    }
  }
```

```
  // Substitui os parametros de simulacao do netlist modelo
     pelos indicados abaixo
  while (ifile.getline(saux,256) && ((s.substr(0,75)).
     compare("simulatorOptions options reltol=100e-6 vabstol
     =1e-6 iabstol=1e-12 temp=27 \\")))
  {
    s=saux;
  }
  ofile << s << endl;
  ofile << saux << endl;

  n = 1;
  while (ifile.getline(saux,256) && n)
  {
    s = saux;
    if ((s.substr(0,14)).compare("save I0.MP14:s"))
      ofile << s << endl;
    else
      n = 0 ;
  }

  ofile << "save ";
  for (i=1; i<=nlines; i++)
  {
    for (j=1;j<=ncolumns;j++)
    {
      ofile << "Y_"<<i<<"_"<<j<<" ";
    }
  }
  ofile << endl;
  ofile<<"saveOptions options save=selected"<<endl;
  ifile.close();
  ofile.close();
  input.close();

  return (0);
}
```

Apêndice B

Análise de Estabilidade da Saída dos Pixels

O resultado da análise da estabilidade das saídas dos pixels, apresentada no Capítulo 4, mostrou como se dá a estabilização das saídas dos pixels e motivou a realização de algumas simulações com o objetivo de definir ou ao menos estimar o tempo necessário para que os pixels da matriz 64×64 atinjam um estado estável.

Com esse objetivo foram feitas quatro outras simulações transientes. Em cada uma delas o momento em que os dados são salvos é diferente. É necessário realizar uma simulação para cada instante de tempo, pois a simulação é interrompida quando o arquivo de saída atinge 2 GB. As informações salvas são referentes aos instantes de tempo de 70 μs, 90 μs, 100 μs e 110 μs. Os dados referentes ao instante de tempo de 80 μs são apresentados no Capítulo 4, mas serão novamente apresentados para comparação.

Para gerar as imagens apresentadas na Fig. B.1, foram analisados todos os dados do arquivo de saída. Através de um programa escrito em C++, cada um desses arquivos de 2 GB foi dividido em outros 64 arquivos, sendo um arquivo por linha da imagem. Cada arquivo possui todas as saídas salvas pelo simulador para os 64 pixels que compõem a linha. Cada um dos 64 arquivos gerados foi analisado em busca de ao menos uma transição na saída de cada um do pixels integrantes da linha. É importante lembrar que cada arquivo possui cerca de apenas 1 μs de tempo de simulação transiente.

Na Fig. B.1, podemos ver que pixels que apresentavam saídas estáveis em 90 μs, apresentam comportamento contrário na simulação em 100 μs. Este comportamento relatado para as Fig. B.1(c) e B.1(d) é observado também em relação a outros instantes de tempo como nas Fig. B.1(b) e B.1(c) e nas Fig. B.1(b) e B.1(e).

Uma explicação para este comportamento está no fato de que há um grande intervalo de tempo entre cada uma das simulacoes realizadas. Para cada simulação realizada, com dados obtidos a cada de 10 μs, há menos de 1 μs de informação

Tabela B.1: Tempo utilizado para realizar as simulações apresentadas neste apêndice.

Simulação	Tempo (h)
Transiente 70 μs	290
Transiente 80 μs	409
Transiente 90 μs	526
Transiente 100 μs	641
Transiente 110 μs	753

e mais de 9 μs sem dados. Esse grande intervalo de tempo sem informação pode estar mascarando os resultados e essas perturbações podem, na verdade, ser mais constantes do que podemos ver.

As mudanças de estado nas saídas dos pixels são tão intensas, que o simulador mantém um passo bastante pequeno durante toda a simulação. Este comportamento além de gerar um volume de dados muito grande em um pequeno intervalo de tempo da simulação, faz também com que a simulação leve um tempo muito grande para ser executada. Na Tabela B.1 são apresentados os tempos necessários para que as simulações fosse concluídas.

A Fig. B.2 apresenta as imagens obtidas das simulações apresentadas nesse apêndice. Essas imagens foram extraídas dos últimos tempos salvos em cada uma das simulações.

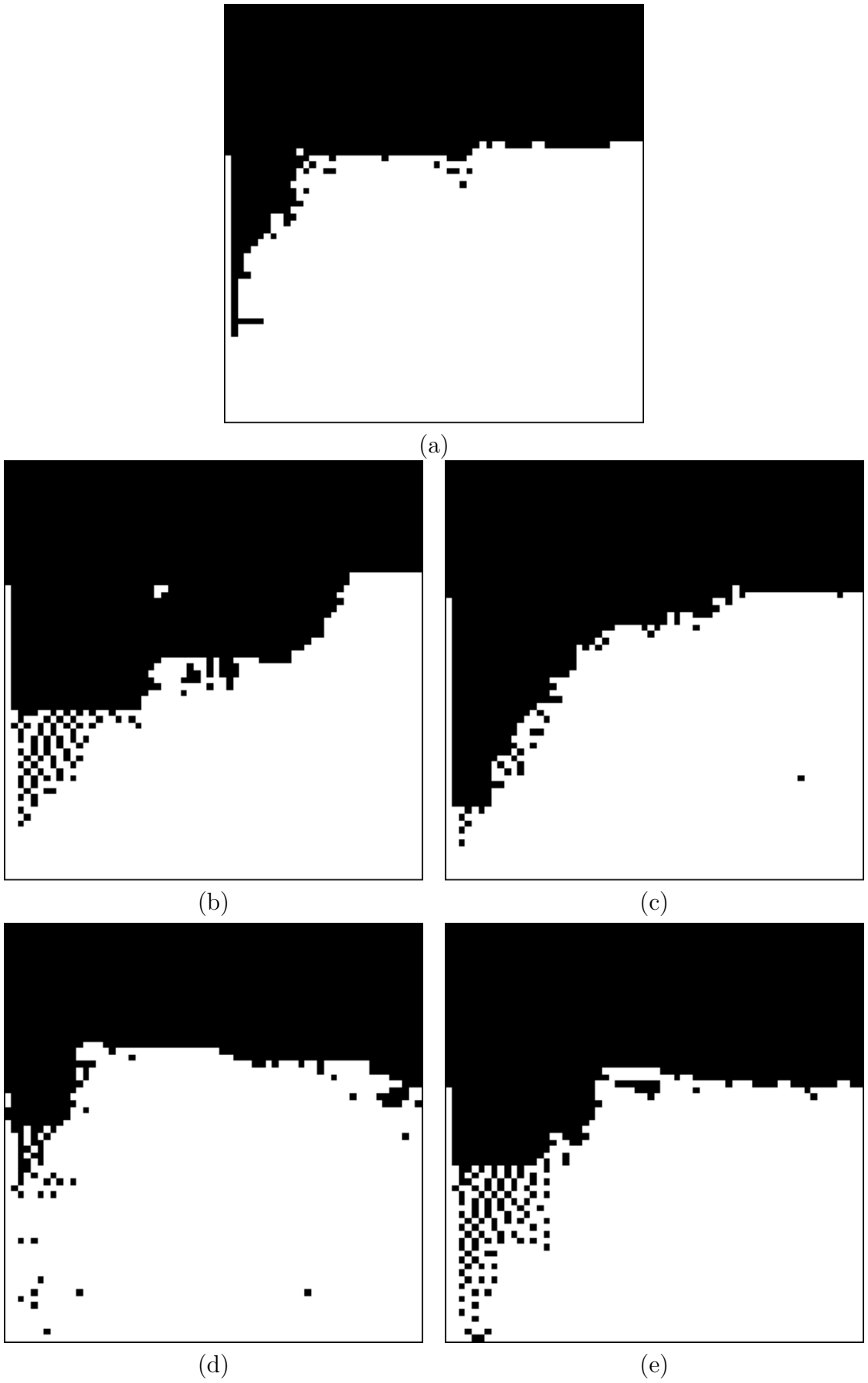


Figura B.1: Resultado das simulações transientes. Os pixels na região escura estão estáveis após: (a) 70 μs; (b) 80 μs; (c) 90 μs; (d) 100 μs e (e) 110 μs.

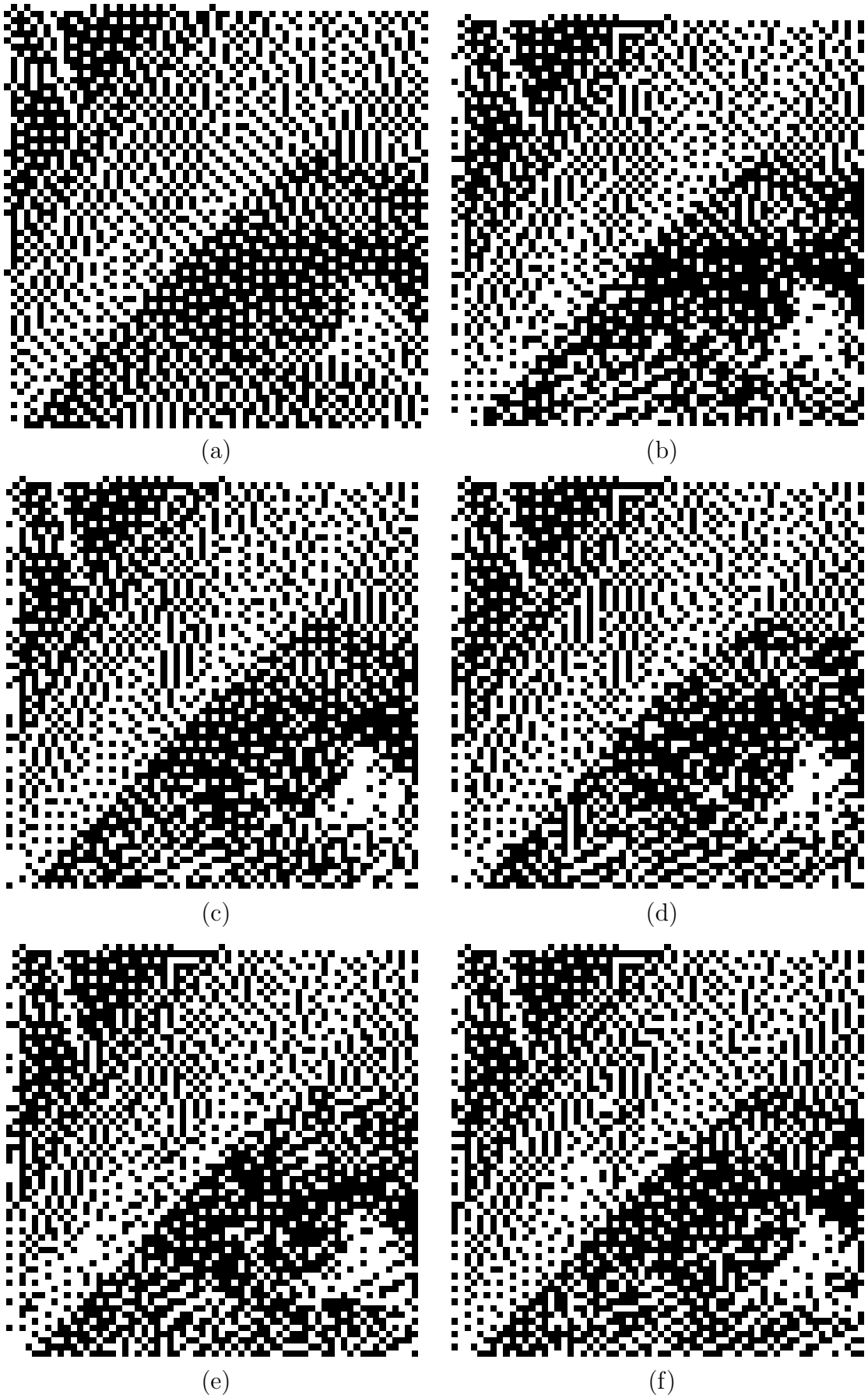


Figura B.2: Imagens obtidas das simulações: (a) do algoritmo ideal; e simulação elétrica transiente após: (b) 70 μs; (c) 80 μs; (d) 90 μs; (e) 100 μs e (f) 110 μs.

Apêndice C

Código Matlab para Avaliar Robustez do Algoritmo

O código utilizado para produzir perturbações no algoritmo foi escrito com base no diagrama de blocos apresentado na Fig. 2.6. As equações descrevem exatamente o algoritmo de difusão de erros utilizando o filtro de Floyd-Steinberg mas com alguns parâmetros a mais para introduzir as perturbações desejadas.

SimMonteCarlo.m

```
clc; close all;
addpath ('.\matlab');
addpath ('.\matlab\dados');

load lena.mat;
x = double(F(100:355,120:375,3))/255;

%AO INICIAR O MATLAB, EXECUTAR OS 2 COMANDOS ABAIXO E DEPOIS
    NAO MUDAR MAIS
%O VALOR DA VARIAVEL savedState.
% defaultStream = RandStream.getDefaultStream;
% savedState = defaultStream.State;

%se eCkt = 1 simula erros em todo o circuito, caso contrario
   , apenas no
%filtro
eCkt = 1;

std = [0.0001 0.001 0.005 0.01 0.05 0.1 0.2];
```

```
for d = 1:size(std,2) %define o desvio padrao a ser
    utilizado na simulacao.
    defaultStream.State = savedState;

    for mc = 1:5
        y = zeros(size(x));
        ye = zeros(size(x));
        xe = zeros(size(x));
        e = 0 + std(d).*randn(1,6);
        for i = 1:size(x,1)
            for j = 1:size(x,2)
                y(i,j) = (sign(((eCkt*e(5))+x(i,j)) - xe(i,j
                    ))+1)/2;
                ye(i,j) = (((eCkt*e(6))+y(i,j)) - ((eCkt*3e
                    -6*e(5))+x(i,j)) - xe(i,j));

                %FLOYD'S & STEINBERG'S FILTER
                if (j<size(x,2))%se nao for um elemento da
                    ultima coluna
                    xe(i,j+1) = xe(i,j+1)+(e(1)+7/16)*ye(i,j
                        );
                end
                if ((j>1) && (i<size(x,1)))%se nao for da
                    primeira coluna nem da ultima linha
                    xe(i+1,j-1) = xe(i+1,j-1)+ (e(2)+3/16)*
                        ye(i,j);
                end
                if (i<size(x,1))%se nao for da ultima linha
                    xe(i+1,j) = xe(i+1,j)+ (e(3)+5/16)*ye(i,
                        j);
                end
                if ((j<size(x,2)) && (i<size(x,1)))%se nao
                    for da ultima coluna nem da ultima linha
                    xe(i+1,j+1) = xe(i+1,j+1)+ (e(4)+1/16)*
                        ye(i,j);
                end
            end
        end
        figure;
```

```
        imshow(y);

        set(gcf,'PaperUnits','centimeters');
        set(gcf,'PaperPosition',[0.0 0.0 20 20]);
        if eCkt == 0
            nome = ['.\MC_filtroFS_Lena_' num2str(std(d)) '_
                ' num2str(mc) '.eps'];
        else
            nome = ['.\MC_Ckt_Lena_' num2str(std(d)) '_'
                num2str(mc) '.eps'];
        end
        print ('-depsc2',nome);
    end
    close all;

end
```