A COMPARATIVE ANALYSIS OF DYNAMIC VISION SENSORS USING 180 nm CMOS TECHNOLOGY

Juan Pablo Girón Ruiz

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
Janeiro de 2017

A COMPARATIVE ANALYSIS OF DYNAMIC VISION SENSORS USING 180 nm CMOS TECHNOLOGY

Juan Pablo Girón Ruiz

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. Mauricio Pamplona Pires, D.Sc.

RIO DE JANEIRO, RJ – BRASIL
JANEIRO DE 2017

Girón Ruiz, Juan Pablo

A comparative analysis of Dynamic Vision Sensors Using 180 nm CMOS Technology/Juan Pablo Girón Ruiz. – Rio de Janeiro: UFRJ/COPPE, 2017.

XVI, 86 p.: il.; 29, 7cm.

Orientador: José Gabriel Rodriguez Carneiro Gomes

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia Elétrica, 2017.

Referências Bibliográficas: p. 66 – 69.

1. Dynamic vision sensors. 2. Frame-free. 3. Temporal contrast changes. 4. Time-based pulse width modulation. 5. Asynchronous delta modulation. 6. Asynchronous logic. 7. Address event representation. I. Gomes, José Gabriel Rodriguez Carneiro. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia Elétrica. III. Título.

"Não to mandei eu? Esforça-te,
e tem bom ânimo; não te
atemorizes, nem te espantes;
porque o Senhor teu Deus está
contigo, por onde quer que
andares."
Josué 1:9

Acknowledgment

Em primeiro lugar, quero lhe agradecer a Deus por ter-me abençoado grandemente nesta experiência na Cidade Maravilhosa. Se falasse todos os milagres que Jesus fez por mim, não alcançaria esta pagina para escrevê-los. Considero que o maior logro que consegui no mestrado foi de reencontrar-me com Cristo Jesus. Aos meus pais, María Esther Ruiz e Luis Eduardo Girón, devo-lhes a minha formação e os valores que hoje levo na minha vida. À minha avó, Fabiola Lopez, por ser a minha segunda mãe incondicional, que Deus te enche de muitos anos de vida. Ao meu avô, Alcides Ruiz, que Deus seja abençoando a tua vida espero vê-lo daqui a pouco. Aos meus irmãos Victor e Andrés, são uns exemplos para mim para ser cada dia melhor pessoa. Aos meus sogros, seu Enrique e Dona Flor, por ser um exemplo para minha vida pela sua coragem para afrontar os problemas. A minha noiva, Ana María Zúñiga, não tenho palavras para lhe agradecer por tudo que tem feito por mim, nunca faltaram palavras de motivação e momentos de muita alegria. Esta experiência não tivesse sido tão enriquecedora sem a sua presença.

Ao meu orientador, Professor José Gabriel, muito obrigado por permitir-me trabalhar com ele e dar-me liberdade para desenvolver esta dissertação de acordo com minha metodologia de estudo. Apesar de seus inumeráveis compromissos com outros estudantes e com a Universidade, sempre separou um espaço em sua agenda para falar sobre este trabalho e fazer que eu pensasse de forma diferente para solucionar as dificuldades. Sem ele este trabalho não teria sido possível.

Aos meus colegas do laboratório de Processamento Analógico e Digital de Sinais (PADS), fico grato por ter-me acolhido nestes dois anos, especialmente a: Thiago Valentin, Fabian Olivera, João, Allan, Fernanda, Pedro e Roberto por dar-me a mão nos momentos que mais precisei. Ao Pastor Tubino por sempre ter uma palavra de vida eterna e de conforto espiritual, sua igreja me permitiu chegar mais perto de Cristo Jesus.

Por último, à CAPES pelo apoio financeiro para a realização de esta dissertação, igualmente para o pessoal administrativo da PEE/COPPE por estar sempre dispostos a orientar-me nos tramites do mestrado.

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

UMA ANÁLISE COMPARATIVA DE SENSORES DE VISÃO DINÂMICA USANDO TECNOLOGIA CMOS 180 nm

Juan Pablo Girón Ruiz

Janeiro/2017

Orientador: José Gabriel Rodriguez Carneiro Gomes

Programa: Engenharia Elétrica

O desenvolvimento de sensores de visão dinâmicos (DVS) é considerado um dos avanços mais relevantes em termos de processamento de dados no plano focal de câmeras CMOS, por estar fundamentado em processamento neural. O tipo de pixel usado por um DVS é baseado na funcionalidade de um caminho neural conhecido como *magno-cellular pathway*, encontrado na conexão entre a retina biológica e o sistema nervoso central, e caracterizado por responder de forma assíncrona às mudanças temporárias de intensidade de luz e por codificar a informação de entrada por meio de pulsos. Neste trabalho, são projetadas e comparadas três arquiteturas DVS: DVS básico, ATIS (asynchronous time-based image sensor) e ADMDVS (asynchronous delta modulation dynamic vision sensor). Entre estas, somente a arquitetura ATIS implementa um sistema de codificação de intensidade da luz, utilizando modulação por largura de pulso no domínio do tempo. No processo de projeto, a metodologia g_m/I_d é usada como ferramenta adequada para o dimensionamento dos transistores que compõem os pixels. Usando diferentes linguagens de programação, são desenvolvidos vários *scripts* para automatizar as etapas de simulação. O funcionamento correto de cada arquitetura é verificado através da comparação entre o resultado obtido por simulação elétrica e o resultado previsto através de simulação numérica usando um modelo idealizado. Finalmente, conclui-se que a resposta de cada arquitetura, obtida por simulação elétrica, se aproxima bastante da resposta prevista através dos modelos idealizados, o que valida os projetos propostos. Com base nos resultados obtidos, é possível realizar uma comparação entre as diferentes arquiteturas.

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

A COMPARATIVE ANALYSIS OF DYNAMIC VISION SENSORS USING 180 nm CMOS TECHNOLOGY

Juan Pablo Girón Ruiz

January/2017

Advisor: José Gabriel Rodriguez Carneiro Gomes

Department: Electrical Engineering

The development of dynamic vision sensors (DVS) is regarded as one of the most relevant advances in CMOS camera focal-plane signal processing, because it is based on neural processing. The type of pixel that is used in a DVS mimicks the functionality of a neural pathway known as *magno-cellular pathway*, which is responsible for part of the communication between the biological retina and the central nervous system. The magno-cellular pathway responds in asynchronous fashion to light intensity temporal variations, and it encodes such variations by means of neural spike sequences. In this work, we designed and compared three DVS architectures: basic DVS, ATIS (asynchronous time-based image sensor)and ADMDVS (asynchronous delta modulation dynamic vision sensor). Among these architectures, only ATIS implements a light intensity encoding system, using time-based pulse-width modulation. In the design process, g_m/I_D methodology is used as a suitable tool for pixel design. Using different programming languages, several scripts are developed for making the simulation stages automatic. To verify the correct operation of each architecture, we compare electrical simulation results to numerical simulation predictions that were previously obtained using ideal pixel models. We finally conclude that the behavior of each architecture, which was obtained by electrical simulation, approximates rather well the behavior that was predicted using ideal models, which validates the proposed pixel design for all three sensor types. Based on these results, the basic DVS, ATIS, and ADMDVS architectures may be compared.

Contents

List of Figures

List of Tables

List of Abbreviations

ADMDVS	Asynchronous Delta Modulation DVS
ADM	Asynchronous Delta Modulation
AER	Address-Event Representation
AMS	austriamicrosystems.com
APS	Active-Pixel Sensor
ATIS	Asynchronous Time-based Image Sensor
CA	Column Acknowledgment
CCD	Charge-Coupled Device
CDS	Correlated Double Sampling
CSV	Comma-Separated Value
DC	Direct Current
DI	Delay-Insensitive
DVS	Dynamic Vision Sensor
EKV	Enz, Krummenacher, and Vittoz
EM	Exposure Measurement
FPN	Fixed-Pattern Noise
GBW	Unity-Gain Bandwidth
KTC	Thermal noise, which is defined by Boltzmann constant k, temperature T, and capacitance C
L	Transistor channel length
ME	Mutual Exclusion

MOS	Metal-Oxide Semiconductor
NDP	Non-Dominant Pole
Op Amp	Operational Amplifier
PWM	Pulse Width Modulation
QDI	Quasi-Delay-Insensitive
Q	Quality Factor
RA	Row Acknowledgment
SF	Source Follower
SR	Reset Switch
SS	Signal Switch
TCS	Temporal Contrast Sensitivity
TIA	Transimpedance Amplifier
W	Transistor channel width

Chapter 1

Introduction

Nowadays people try to build systems with performance akin to their biological counterparts, but their objectives are in general still far from the reality. Small insects still outperfom powerful computing machines in executing several tasks involving a balanced combination of real-time data processing, control systems, sensory systems, and the optimization of bandwidth and power consumption [1].

Carver Mead, from the California Institute of Technology, introduced in the 80's the concept of *neuromorphic* systems [2], to refer to systems which try to mimic some of the properties of biological neural architectures. Systems based on the neuromorphic approach are, in some aspects, more efficient than conventional designs. A vision sensor inspired on biological retinas, for example, only generates outputs when it detects spatial or temporal changes in its inputs, whereas conventional video sensors are continuously generating output data regardless of changes occurring or not within their fields of view.

Image sensors have evolved greatly since the 70's. The first charge-coupled imaging devices (CCDs) were built around 1969 by Willard Boyle and George Smith at the AT&T Labs [3]. A CCD is composed by an adjacent association of charge-coupled capacitors which are sensitive to light. The implementation usually adopts one of three different architectures [4]: full-frame transfer, frame transfer, and interline transfer. From one architecture to another, the shuttering approach changes. The active-pixel sensor (APS) came out some years later, around the middle 80's [5], [6]. The APS is a metal-oxide semiconductor (MOS) image sensor. The APS pixel typically includes a photodiode, which generates a current proportional to the incident light intensity, and at least three transistors: i) a reset transistor, which is used for clearing charge that has been integrated at the photodiode cathode terminal, ii) an amplifying transistor usually in source follower configuration, and iii) a switch that either blocks or allows pixel read-out. Contrary to CCD the MOS imagers (i.e. APS image sensor) do not use charge transfer to convey the data information to external (off-chip) system. Both the CCD and the APS have been widely used in

many applications: scientific surveys, astronomy, satellite imaging, consumer video, cell phones, webcams, and so forth.

In frame-based image sensors, either CCD or APS, the integration time is set according to a global shutter . Many vision sensors follow the frame-based approach. Differently from frame-based vision sensors, the biological retinas do not use the 'frame' concept. They generate output data asynchronously and their data output depends on specific features detected in the sensor field of view. To inform what type of event is being captured, specific structures, for example parvo-cellular and magno-cellular pathways, exist in the biological retina [2].

As neuromorphic engineering expands quickly, some properties of biological vision systems are successfully mimicked today. According to the biological structure examples mentioned in the previous paragraph, vision sensors inspired on biological vision systems are roughly divided into two categories: i) *Spatial Contrast* sensors, which reduce spatial redundancy by replicating aspects of the parvo-cellular pathway [7], [8]; and ii) *Temporal Contrast* sensors, which reduce temporal redundancy by replicating aspects of the magno-cellular pathway [7], [8].

One of the most important vision sensors in the neuromorphic engineering field is the Dynamic Vision Sensor (DVS). The DVS arose from the CAVIAR Project[1], where the CAVIAR acronym stands for *Convolution Address-Event Representation* (AER) *Vision Architecture for Real Time*. The DVS was the first event-based commercial vision sensor to respond asynchronously to temporal changes. Its output information is encoded by short voltage pulses informally denoted as *spikes*, and the spikes are conveyed to subsequent systems via AER protocols [9]. Temporal waveforms containing spike sequences are informally denoted as *spike trains*. The DVS approach mimicks its biological counterpart, and this approach is usually referred to as the *frame-free* vision sensor.

The DVS has some advantages with respect to frame-based vision sensors. As the DVS pixels only respond to temporal changes, asynchronously and through spike trains, they reduce bandwidth and power consumption, whereas the pixels in frame-based sensors have their values transmitted even if no change occurs within the field of view. As the DVS does not have a global shutter, each pixel independently defines its own operation point. Since the DVS invention, research has focused on including additional features that are similar to those found in biological retinas. Pixels with larger temporal contrast sensitivity have been proposed [10], [11]. To efficiently encode light intensity temporal variations, other designs focus on asynchronous time-based modulation [12], [13].

[1]http://www2.imse-cnm.csic.es/caviar/introduction.htm

1.1 Objectives

This work aims at studying a few different DVS architectures and comparing them. In particular, this work focuses on three architectures: basic DVS, asynchronous time-based image sensor (ATIS), and asynchronous delta modulation DVS (AD-MDVS). Available architectures are analyzed and comparison methods are proposed. To accomplish that task, specific goals were defined:

- Studying bibliography references about free-frame vision sensors (Chapter 2);
- Modelling selected architectures at the system level, in a numerical environment (Chapter 2);
- Studying g_m/I_D methodology[2] [14] for DVS design (Chapter 2);
- Designing and simulating a single instance for each pixel (Chapters 3 and 4);
- Building input stimulus files that are suitable for pixel model validation (Chapter 4);
- Designing and simulating a small pixel array (i.e. an array with a few pixels) in order to validate pixel operation when two or more pixels simultaneously detect light intensity temporal change (Chapter 4);
- Using different programming languages, to make the test phase automatic (appendices)

1.2 Text Organization

In Chapter 2 we will explain basic DVS concepts, including models for each pixel that is studied in this dissertation. In Chapter 3, pixel design details are presented, including a suitable application of g_m/I_D methodology to pixel design. Chapter 3 also contains an innovative delay comparator design, not found in the papers we studied, that affects the exposure measurement logic in the ATIS architecture. Comparisons among pixel models, based on electrical simulation results, are presented in Chapter 4. The main conclusions and some future research topics are presented in Chapter 5. Some algorithms required for the simulation of a small ATIS camera are included in the appendices.

[2] g_m/I_D is a design method which is used to design integrated circuit based on the efficiency of the transistor transconductance (Section 2.6). We use EKV (Enz, Krummenacher, and Vittoz) model for modelling DVS sensor (Section 2.1.1). The g_m/I_D ratio can be obtained using transistor models (i.e. EKV model). The g_m/I_D method was used for designing the operational amplifier (Section 3.2) and voltage comparator with hysteresis (Section 3.3).

Chapter 2

Theory

In this chapter, we address fundamental concepts about the DVS options studied in this work. The pixel models are described. An introduction to asynchronous circuits, delay models, and AER systems is provided. These explanations lead to an understanding of how a reduced number of bits represents, in the frame-free vision sensor, light intensity values from all pixels in the array. Additional pixel operation details are provided in subsequent chapters.

2.1 Dynamic Vision Sensor

The DVS pixel responds to temporal contrast changes, instead of the absolute light intensity sampled at the pixel location. The DVS pixel behavior is inspired by the behavior of neurons in the magno-cellular pathway of a biological vision system [2]. They are primarily sensitive to temporal changes in incoming light intensity (sampled by photoreceptors at the retina), and they tend to ignore constant light intensity values. The temporal contrast in DVS sensors is defined by Equation (2.1) [15]:

$$\text{TCON} = \frac{1}{I(t)}\frac{dI(t)}{dt} = \frac{d(\ln(I(t)))}{dt}, \tag{2.1}$$

where $I(t)$ is the photocurrent at the local photodiode. Autonomously, the DVS pixel transmits temporal change information through two asynchronous communications channels, also found in the biological vision system, which are denoted as *on* and *off* channels [2]. Figure 2.1 indicates that the DVS pixel uses a logarithmic photoreceptor, a circuit for the computation of temporal derivatives with gain equal to C_1/C_2, two voltage comparators, and a logical circuit for AER communication. Each block is described next.

The original DVS schematic diagram [4], [15] is simpler than the one shown in Figure 2.1. In the original schematic diagram, the temporal derivative circuit

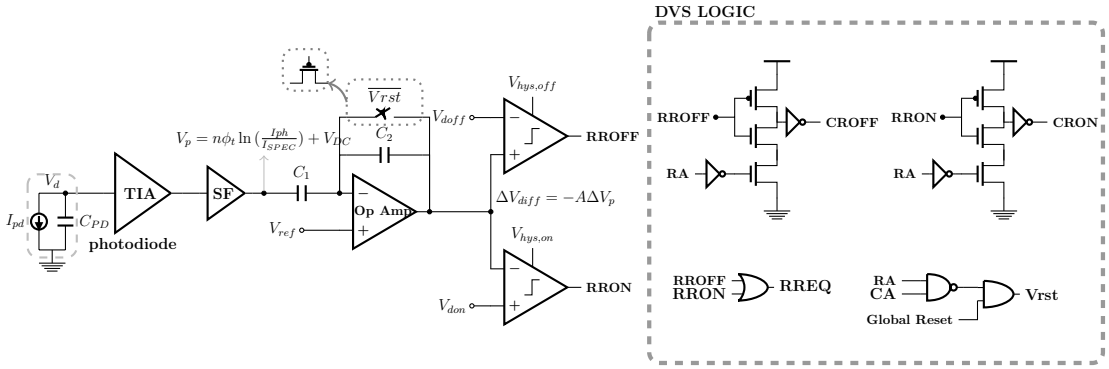


Figure 2.1: Pixel schematic diagram.

Table 2.1: DVS pixel input and output signals.

Object Name	**I/O**	**Object Description**
V_{ref}	In	Starting voltage level (between V_{doff} and V_{don}) for DVS pixel
V_{doff}	In	*Off* event minimum absolute voltage threshold
V_{don}	In	*On* event minimum absolute voltage threshold
Global Reset	In	Global command (active at low level) for resetting pixel value to V_{ref}
$V_{hys,off}$	In	Voltage reference for hysteresis loop in *off* comparator
$V_{hys,on}$	In	Voltage reference for hysteresis loop in *on* comparator
I_{pd}	In	Photodiode current that is due to the photoelectric effect
V_d	-	Photodiode cathode terminal
RREQ	Out	Data bus pixel request (active at high level)
CRON	Out	Positive temporal contrast change indication (active at high level)
CROFF	Out	Negative temporal contrast change indication (active at high level)
RA	Out	Row AER (Y-AER) acknowledgment, which enables row data transmission
CA	Out	Column AER (X-AER) acknowledgment

is implemented by a single common-source amplifier, and the voltage comparators are implemented by logical inverters with independent inputs. In the present work, the common-source amplifier was replaced by a conventional two-stage operational amplifier, and the logical inverters were replaced by two-stage voltage comparators with hysteresis [16]. We decided to replace those parts of the original circuit by these more complex ones for two reasons: i) in the common-source configuration, after an event has occurred, its output is taken to the operation point V_{ref} because of a short circuit between the gate and drain voltages. The temporal derivative circuit is thus converted into a diode-connected circuit with an active load[1]. For power consumption savings, we would like the analog circuits to operate in the subthreshold regime, and so the bias voltage at the p-channel transistor gate is at least 1.5 V (assuming that the supply voltage is 1.8 V and the transistor threshold voltage is around 0.3 V). To make operation point adjustment easier, we use an

[1]The common-source amplifier uses one p-channel transistor (input) and one n-channel transistor (current source). When a reset takes place, the gate and drain terminals of the p-channel transistor are connected, which turns the p-channel transistor into a diode forward biased by a current source (which is the n-channel transistor). The common-source amplifier output is taken to V_{ref}.

operational amplifier. The operational amplifier has two inputs, while the common-source amplifer has only one. The V_{ref} voltage in the common-source amplifier is defined by the bias current, while in the operational amplifier the V_{ref} voltage is defined at its non-inverting input, and the bias current does not depend on V_{ref}; ii) using logical inverters as voltage comparators leads to large MOS transistors, and it is not possible to obtain hysteresis with them. To make comparator operation robust to input noise, using comparators with hysteresis is desirable. To reduce area and power consumption, we use a classical two-stage voltage comparator with hysteresis.

The original DVS has several advantages over conventional frame-based cameras. It only responds to input changes, thus leading to reduced bandwidth data transmission. Because of AER, a few bits represent any pixel in the sensor array. Although the data redundancy is reduced, the timing information is accurately preserved. As the DVS pixel response is independently composed by asynchronous spikes, power consumption is reduced as well. Depending on temporal contrast, each input event is simply classified as either an *on* or an *off* event. As the pixel reset signal (to establish the pixel operating point) is self-generated, an external clock is not required.

The DVS pixel (Figure 2.1) is basically composed by five building blocks, which are described next. For clarity, the symbols representing input and output signals in Figure 2.1 are described in Table 2.1.

1. *Logarithmic Transimpedance Amplifier (TIA)*: this amplifier logarithmically converts a photocurrent signal into a voltage-mode signal. The circuit is designed to rapidly respond to temporal changes. As the V_d node is kept at virtual ground, the transimpedance amplifier bandwidth is increased;

2. *Source Follower (SF)*: this amplifier drives the large capacitive load that is present at the input of the circuit for temporal derivative computation;

3. *Temporal Derivative Circuit:* this circuit amplifies, with gain equal to C_1/C_2, relative changes in log intensity that take place after the latest reset [15]. The DC signal component is cancelled. If the pixel detects a temporal contrast change event, and if this event is acknowledged by an off-chip acknowledgment system[2], then the temporal derivative circuit output is reset to the starting voltage level (V_{ref}, according to Table 2.1). After that, the temporal contrast change detection process restarts. The differencing circuit output keeps increasing (negative temporal contrast change) or decreasing (positive temporal

[2]The AER systems (row and column) generate acknowledgment signals. The simplest implementation of an acknowledgment system involves delays (logical inverter pairs). The output of the highest level arbiter generates a delayed version of RREQ, which returns to the ACK input (Figure 3.10) for self-acknowledgment.

contrast change). It returns to V_{ref} after a request (threshold crossing) is acknowledged.

The DC signal component is cancelled. If the pixel detects a temporal contrast change event, and if this event is acknowledged by an off-chip acknowledgment system, then the temporal derivative circuit output is reset to the starting voltage level (V_{ref}, according to Table 2.1). After that, the temporal contrast change detection process restarts;

4. *Comparators*: to decide whether the temporal contrast change was large enough to generate an event, the DVS pixel uses two voltage comparators. The arbitrary thresholds that are set by both comparators define the pixel temporal contrast sensitivity (TCS). Comparator outputs are connected to a logical interface. The comparators also feature hysteresis, and the hysteresis voltage range may be defined by means of a control voltage, which is denoted as V_{hyst} in Figure 2.1. The RROFF signal indicates that the temporal contrast change event was negative, and the RRON signal indicates that the temporal contrast change event was positive;

5. *Logical Interface*: to establish a handshaking protocol with an AER circuit, a logical interface generates a reset signal, which is denoted as V_{rst} in Figure 2.1, after the temporal contrast change event was acknowledged by an external (off-chip) system.

Figure 2.2a illustrates the DVS pixel as a building block. This building block has the same input and output signals that were already described in Table 2.1. The DVS pixel operation is illustrated by the diagram in Figure 2.2b. In Figure 2.2b, the solid curve at the top refers to the photodiode cathode voltage, after amplification by the transimpedance amplifier and the source follower blocks, which is denoted as V_p. The dotted line corresponds to an approximate V_p reconstruction, which is obtainable from the pulses shown at the bottom plot. This bottom plot of Figure 2.2b shows the pulses generated at the temporal derivative circuit output, which is denoted as V_{diff}. The V_{diff} signal is obtained by integrating the ΔV_{diff} signal in Figure 2.1. The comparators continuously verify whether V_{diff} exceeds *on* or *off* threshold levels, which are denoted as V_{don} and V_{doff} in Figure 2.1 and Table 2.1. If the V_{diff} absolute value corresponds to sufficiently high temporal contrast change, then one of the comparator outputs changes into 'active' (e.g. high voltage level), which means that the pixel requests data bus access in order to transmit that high temporal contrast change event. This activity starts communication with an external AER circuit. After the pixel request is acknowledged by the AER circuit, the AER circuit sends a reset signal back to the pixel. This reset signal brings the

operation point (i.e. the output V_{diff} voltage level) back to V_{ref}, which means that the pixel is ready to detect new temporal contrast change events.

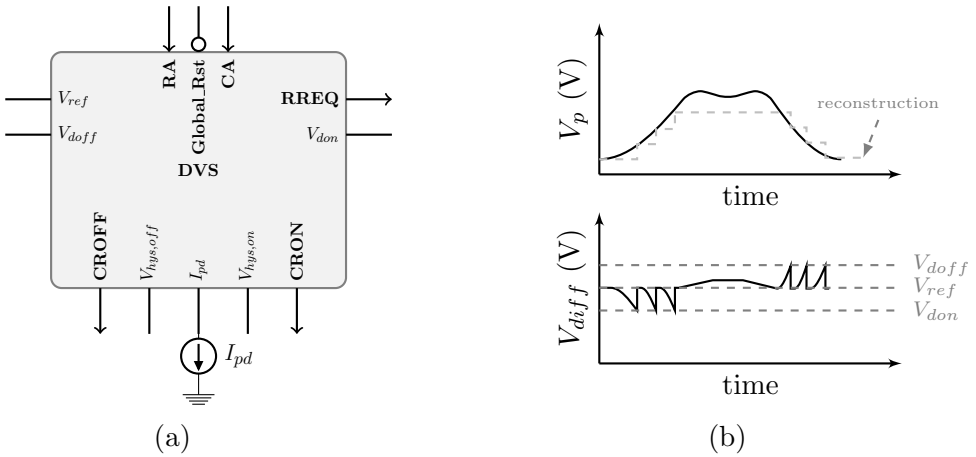


Figure 2.2: DVS pixel instance. (a) DVS pixel as a system building block, and (b) DVS pixel operation fundamentals, adapted from [15].

Because of transistor mismatch (threshold voltage and geometrical aspect variation) at the photoreceptor level, the DVS pixel array suffers from a high mismatch among the pixels. That mismatch is partially compensated by using temporal derivative circuits with an accurate gain, which is obtained by carefully matching the capacitors that realize the C_1/C_2 ratio [4], [10], [15].

2.1.1 DVS Pixel Model

The following analysis is based on [10]. Using the EKV (Enz, Krummenacher, and Vittoz) model for subthreshold regime operation [17], the V_{diff} signal can be modelled by Equation (2.4). To obtain the V_{diff} signal we need to find the source follower output as next:

$$V_p = n\phi_t \ln\left(\frac{I_{pd}}{I_s}\right) + V_{DC} \tag{2.2}$$

where n is the subthreshold slope factor, ϕ_t is the thermal voltage, $I_s = 2n\phi_t^2 KpW/L$ is the subthreshold current factor of a MOS transistor, $V_{DC} = V_d + V_T$, W is transistor channel width and L is transistor channel length. Because of V_p feeds the differencing circuit then we have that:

$$
\begin{aligned}
dV_{diff} &= -AdV_p \\
&= -\frac{C_1}{C_2} n\phi_t d \ln\left(\frac{I_{pd}}{I_{spec}}\right) \\
&= -\frac{C_1}{C_2} n\phi_t d\frac{I_{pd}}{I_{pd}}
\end{aligned} \tag{2.3}
$$

Integrating Equation (2.3) we have V_{diff} equal to:

$$
V_{diff} = -\frac{C_1}{C_2} n\phi_t \ln\left(I_{pd}\right) \tag{2.4}
$$

To define whether temporal contrast change is positive or negative, the V_{diff} signal is compared to two threshold voltages, which are denoted as $V_{diff,on}$ and $V_{diff,off}$ in Equations (2.5) and (2.6). The $V_{diff,on}$ threshold voltage is negative, and the $V_{diff,off}$ threshold voltage is positive. To define $V_{diff,on}$, we compute the difference between V_{don} and V_{ref}, taking into account the fact that V_{don} sets the threshold for a minimum voltage (relative to V_{ref}) that is required for firing an *on* event. The $V_{don} - V_{ref}$ voltage difference is compensated by $V_{os,comp} + V_{os,opamp}$. The $V_{os,comp}$ voltage represents the offset voltage at the comparators, and $V_{os,opamp}$ represents the offset voltage at the temporal derivative circuit. The definition of $V_{diff,off}$ is similar to the definition of $V_{diff,on}$, but it is based on V_{doff}, which sets the threshold for a maximum voltage (relative to V_{ref}) that is required for firing an *off* event.

$$
V_{diff,on} = (V_{don} - V_{ref}) + (V_{os,comp} + V_{os,opamp}) < 0 \tag{2.5}
$$

$$
V_{diff,off} = (V_{doff} - V_{ref}) + (V_{os,comp} + V_{os,opamp}) > 0 \tag{2.6}
$$

To compute the minimum temporal contrast inputs that generate events (either *on* or *off* events), we define θ^+ as the minimum temporal contrast that generates an *on* event. Similarly, θ^- is the minimum temporal constrast that generates an *off* event. Integrating Equation (2.3) yields

$$\theta^{+} = \left|\ln\frac{I_{bright}}{I_{dark}}\right| = \left|\frac{V_{diff,on}C_2}{C_1 n\phi_t}\right|$$
$$= \left|\frac{C_2((V_{don} - V_{ref}) + (V_{os,comp} + V_{os,opamp}))}{C_1 n\phi_t}\right| \quad (2.7)$$
$$\theta^{-} = \left|\ln\frac{I_{dark}}{I_{bright}}\right| = \left|\frac{V_{diff,off}C_2}{C_1 n\phi_t}\right|$$
$$= \left|\frac{C_2((V_{doff} - V_{ref}) + (V_{os,comp} + V_{os,opamp}))}{C_1 n\phi_t}\right| \quad (2.8)$$

2.2 ATIS Pixel

Improving the dynamic range of vision cameras is currently an active research topic. Using time-based encoding in CMOS technology improves pixel performance with respect to dynamic range [16], [18]. Each pixel may choose its own integration time, as it responds to input variations in autonomous and asynchronous fashion.

Like the DVS, the ATIS pixel is time-based. It was conceived in an attempt to mimic the magno-cellular and parvo-cellular pathways of biological vision systems. Roughly speaking, the magno-cellular pathway uses a spatially coarse representation system, which allows for quick detection of a new incoming event. The parvo-cellular pathway focuses on the event detailed description, thus allowing for a definition of the object that generated the incoming event [2]. As Figure 2.3 shows, the ATIS pixel is composed by one DVS pixel, which works as a temporal change detector, and one exposure measurement (EM) circuit, which encodes the light intensity associated with the event generated by the DVS pixel. The DVS pixel roughly plays the role of a magno-cellular pathway input, whereas the exposure measurement circuit plays the role of a parvo-cellular pathway input.

Similarly to its biological counterparts, the ATIS pixel presents voltage spike trains at its outputs. The spikes are generated in autonomous and asynchronous fashion by the ATIS pixel, and they simultaneously encode local brightness (luminance) information and temporal contrast change information. The ATIS pixel implements a particular correlated double sampling (CDS) [18] technique for KTC noise[3] and FPN (fixed pattern noise) removal [19], which is called True CDS. The True CDS technique is explained in Sections 2.2.1 and 2.2.2. To simultaneously detect temporal contrast change and measure light intensity, the ATIS pixel uses two photodiodes separately. As the brightness measurement cycle only starts after an event was generated, the ATIS temporal contrast sensitivity depends on the DVS pixel anyway. The ATIS camera thus uses two AER systems. The first one, which

[3]Thermal noise on circuit capacitors is kT/C, where k is the Boltzmann constant, T is temperature, and C is node capacitance.

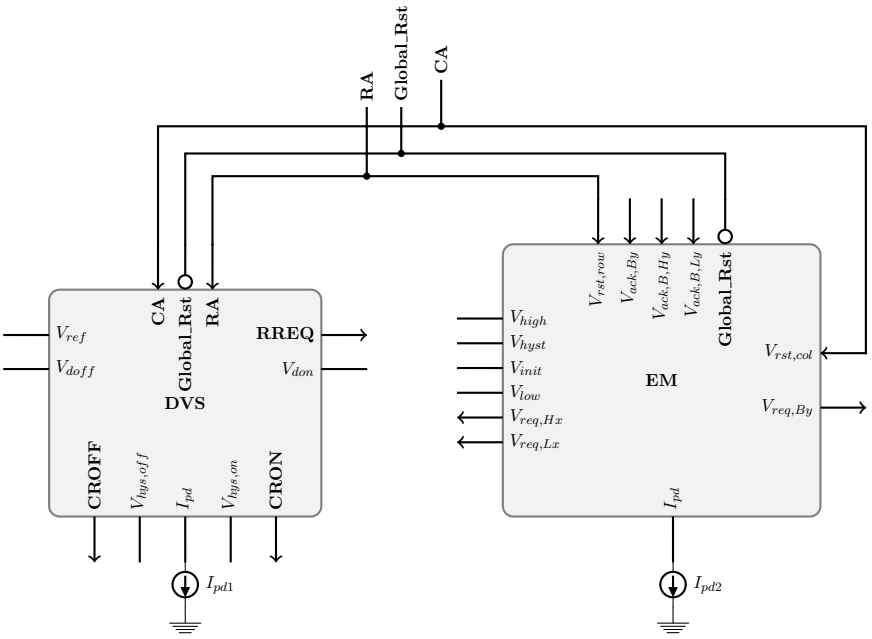


Figure 2.3: ATIS pixel instance, which is composed by a DVS pixel (left) and an exposure measurement circuit (right).

Table 2.2: ATIS pixel input and output signals.

Object Name	**I/O**	**Object Description**
V_{high}	In	Voltage reference to start the brightness encoding process
V_{low}	In	Voltage reference to complete the brightness encoding process
V_{hyst}	In	Voltage reference for hysteresis loop in comparator
V_{init}	In	Global command to bring the logic to valid state, it is used just once in all operation
Global Reset	In	Global command (active at low level) for resetting pixel value to V_{ref}
$V_{rst,row}$	In	Row acknowledgment from DVS (Y-AER), which enables row data transmission
$V_{rst,col}$	In	Column acknowledgment from DVS (X-AER)
$V_{ack,By}$	In	Row acknowledgment from ATIS (Y-AER), which enables row data transmission
$V_{ack,B,Hy}$	In	Column acknowledgment from ATIS (X-AER)
$V_{ack,B,Ly}$	In	Column acknowledgment from ATIS (X-AER)
$V_{req,By}$	Out	Data bus pixel request to transmit its pixel encoding status
$V_{req,Hx}$	Out	Indicates that the pixel started the brightness encoding process
$V_{req,Lx}$	Out	Indicates that the pixel finished the brightness encoding process

is based on the DVS pixel, encodes temporal contrast change events, and the second one encodes local brightness (local light intensity).

As it was previously mentioned in Chapter 1, time-based vision sensors such as DVS, ATIS, and so on, have some advantages over frame-based imagers (e.g. APS camera): reduced transmission bandwidth, as less data are generated by the pixel array; lower buffer storage requirements, as information pre-processing leads to less raw information; and simpler post-processing for image decoding. ATIS presents two particular advantages in addition to those ones: it encodes light intensity through pulse-width modulation (PWM), which is a very simple data transmission method,

and it improves signal-to-noise ratio, by means of True CDS.

In Sections 2.2.1 and 2.2.2, we will briefly discuss CDS concepts, in order to explain how CDS leads to better image quality in conventional image sensors. We will also address True CDS as a technique for enhancing the output signal in time-based vision sensors.

2.2.1 Correlated Double Sampling (CDS)

To obtain high signal-to-noise ratio, which translates into good image quality, the pixel *readout noise* shall be kept low. Pixel readout noise is a generic expression that includes several types of noise [20]: the intrinsic noise present in pixel devices (dark signal non-uniformity and pixel response non-uniformity), KTC noise, $1/f$ (flicker) noise, and fixed pattern noise.

Correlated double sampling (CDS) is commonly used in CCD and APS cameras, in order to reduce readout noise. The pixel output is read (i.e. sampled) twice: one sampling operation captures the absolute signal value that is obtained by photocurrent integration in CCD and APS cameras, and another sampling operation captures the reset value. The output signal, which has less noise than the absolute signal value, is obtained by subtracting both samples, as shown in Figure 2.4. The reset switch (SR) is opened at the end of the reset phase, and the signal switch (SS) is opened at the end of the photocurrent integration. As any errors caused by pixel mismatch or process variations affect the absolute signal value and the reset value equally, those errors are cancelled at the differential amplifier output. For pixel read-out, the CDS approach in Figure 2.4 requires global control signals.

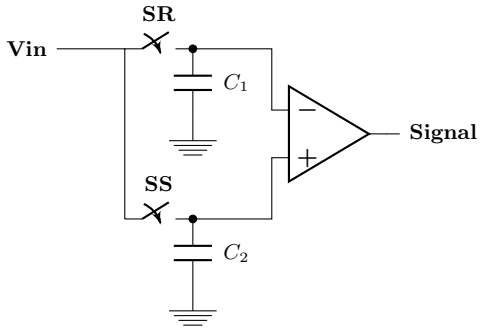


Figure 2.4: CDS schematic diagram.

2.2.2 True Correlated Double Sampling

The true CDS technique was designed for time-based vision sensors, which is the ATIS case. This CDS method is referred to as *true* CDS because the double sampling is carried out in a single integration cycle, rather that within two subsequent

integration cycles, which corresponds to the basic CDS case. Keeping the two samples within the same integration cycle reduces reset noise (i.e. noise associated with the pixel reset voltage) by a factor approximately equal to 2 [18].

Figure 2.5 shows two possibilities (single threshold or two thresholds) for true CDS operation in a time-based vision sensor. In either case, an active (high level) reset pulse causes the photodiode cathode voltage (V_d, which is denoted by V_{int} in this figure) to be raised to the reset level that is denoted by V_{rst} in this figure. When the reset pulse is turned off, the photodiode depletion capacitance is discharged because of the photocurrent integration, which is illustrated by the oblique straight line segments in the figure. In the true CDS with a single threshold (Figure 2.5a), a voltage comparator generates an output (V_{comp}) step, which is later shaped into a voltage pulse, when V_{int} reaches the threshold value V_{ref}. This indicates the end of the integration interval. The integration interval duration t_{int} is measured by the time difference between the comparator step rising edge and the reset pulse falling edge. It is inversely proportional to local light intensity. In the true CDS with two thresholds (Figure 2.5b), t_{int} does not depend on the photodiode reset level. In conventional photodiodes, the cathode voltage may fluctuate during the reset pulse, which is due to charge injection through the reset transistor among other uncertainty factors. Eliminating t_{int} dependence on the photodiode reset voltage thus improves signal accuracy by reducing reset noise, as mentioned in the previous paragraph. The upper threshold V_{high} is chosen according to the maximum expected photodiode reset error. The lower threshold V_{low} is chosen according to the maximum expected light intensity.

The true CDS implementation using a differential amplifier is shown in Figure 2.6. The exposure measurement logic controls the V_{low} and V_{high} threshold switches. It also requests AER encoding of the pixel event (V_{comp} pulse in Figure 2.5). The integrated light intensity signal is represented by the time difference between the pulses that are generated by threshold crossings at V_{high} and V_{low}. Figure 2.5b shows that, to obtain the time difference, only one integration cycle (between two reset pulses) is necessary. Comparator mismatch errors, as well as time delay differences between the threshold crossing events, tend to cancel out, which further reduces readout noise. The ratio between the sampling error ϵ_{tint} (readout noise) and the readout signal t_{int} was evaluated for the single-threshold and two-threshold cases [18], and it was shown that the ratio is smaller if two thresholds are used. Furthermore, the analysis in [18] indicates that the ratio decreases as the supply voltage is reduced, which is convenient in modern CMOS technologies such as 180 nm, 110 nm, and so forth.

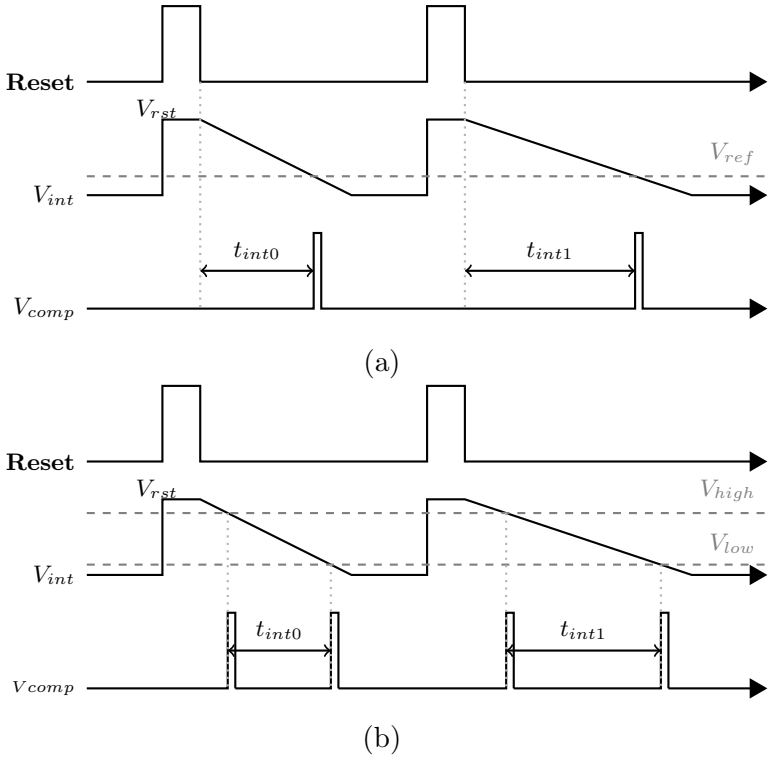


Figure 2.5: True CDS operation in time-based vision sensors: (a) single threshold, and (b) two thresholds [18].

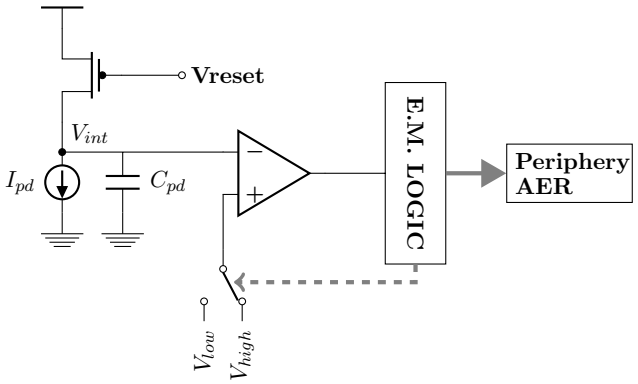


Figure 2.6: True CDS implementation.

2.2.3 ATIS Model

The ATIS temporal change detector is the DVS pixel itself. The exposure measurement logic is described in this section. When the reset signal is off, the photodiode cathode voltage is:

$$V_{int}(t) = \frac{1}{C}\int_0^t i_{pd}(\tau)d\tau + V_{int}(0) \qquad , t > 0, \tag{2.9}$$

where i_{pd} is the photocurrent, C is the photodiode cathode node capacitance, and $V_{int}(0)$ is the initial condition. We assume $V_{int}(0) = V_{DD}$, which is the imager supply voltage. At first, $V_{int}(t)$ is compared with V_{refh}. After V_{int} crossed the V_{high} and an

acknowledgement signal was received, $V_{int}(t)$ is compared with V_{low}:

$$V_{out,EM} = \begin{cases} Req\_High, & \text{if } V_{int} \leq V_{high} \\ Req\_Low, & \text{if } V_{int} \leq V_{low} \end{cases} \tag{2.10}$$

The integration interval duration for a pixel is:

$$t_{int} = t(Req\_Low) - t(Req\_High). \tag{2.11}$$

If the temporal change detector detects another event before the light intensity measurement is completed, i.e. before $V_{out,EM} = Req\_Low$, then the off-chip AER system ignores the first request, which had happened when V_{int} crossed the V_{high} threshold, and it starts again.

To map t_{int} values into grayscale values, the t_{int} values are measured for different incoming light intensity values, say 256 different values, and the t_{int} measurements are stored in a look-up table. Using the look-up table, off-chip intensity value decoding is performed more accurately than through Equation (2.9).

2.2.4 Additional ATIS Sensor Functionality

Besides reacting to temporal input changes and encoding light intensity values, the ATIS sensor may capture photographs [16], because the exposure measurement circuit does not remove the time-domain DC component of the input signal (which the DVS does remove). A logical circuit that enables asynchronous capture of static images in the ATIS sensor is proposed in Figure 2.7a. Using this logical circuit, each pixel decides its own integration time. To use the proposed circuit, minor modifications in the original ATIS pixel design are required Input and output signals are shown in Figure 2.7b.

If RA and CA (row and column acknowledgment) signals are received by the pixel, or if a global reset is active (at low level), then the pixel exposure measurement circuit and temporal change detector are reset (the Reset_ATIS signal goes to the low level). To take a photo using ATIS, the temporal change detector must be blocked for time enough for reading out the information from all the pixels. For a short time, a reset signal is generated at the exposure measurement circuit Global_-Rst input. The Req_fr* signal is a delayed version of Req_fr, where the Req_fr symbol stands for a *frame request* that is generated when the ATIS camera captures a photograph. The time delay is long enough for a stable reset voltage to be defined at the photodiode cathode. When both Req_fr and Req_fr* are high, the exposure measurement circuit reset signal is turned off, and the integration interval starts. While Req_fr or Req_fr* are high, the temporal change detector is blocked. Off-chip,

a frame-mode function would wait as the values from all pixels are read out. After light intensity information from all pixels has been received, this off-chip frame-mode function disables the Req_fr signal.

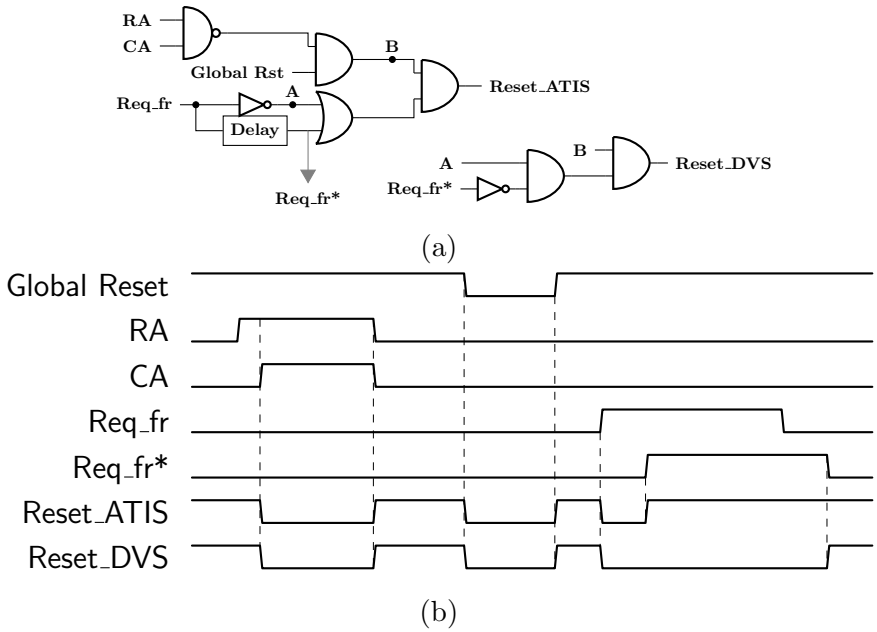


Figure 2.7: Logical circuit enabling ATIS photograph capture: (a) schematic diagram, and (b) timing diagram.

In an ATIS camera that does not take pictures, the Reset_DVS in Figure 2.7b is the same as the Reset_ATIS signal. Otherwise, i.e. if the ATIS camera does take pictures, the Reset_DVS signal is obtained from the logical circuit presented in Figure 2.7a.

2.3 ADMDVS pixel

The ADMDVS pixel is an improved version of the DVS pixel. To reset the output signal to a starting voltage reference level after a temporal contrast change event, the ADMDVS pixel uses a modified version of asynchronous delta modulation (ADM) rather than the reset cycle that was used in the DVS pixel. The ADMDVS also uses a different information encoding method, which is denoted as feedback and subtract. The basic DVS encoding method is denoted as feedback and reset. Both encoding methods are shown in Figure 2.8. The feedback and reset method uses a reset switch to reset the operation point after a temporal contrast change event. The feedback and subtract method, instead of using a reset switch to reset the operation point after a temporal contrast change event, subtracts a fixed δ value from the V_{diff} input. Whereas the feedback and reset method interrupts the input signal flow during the reset, in the feedback and subtract method the input signal flow is

never interrupted.

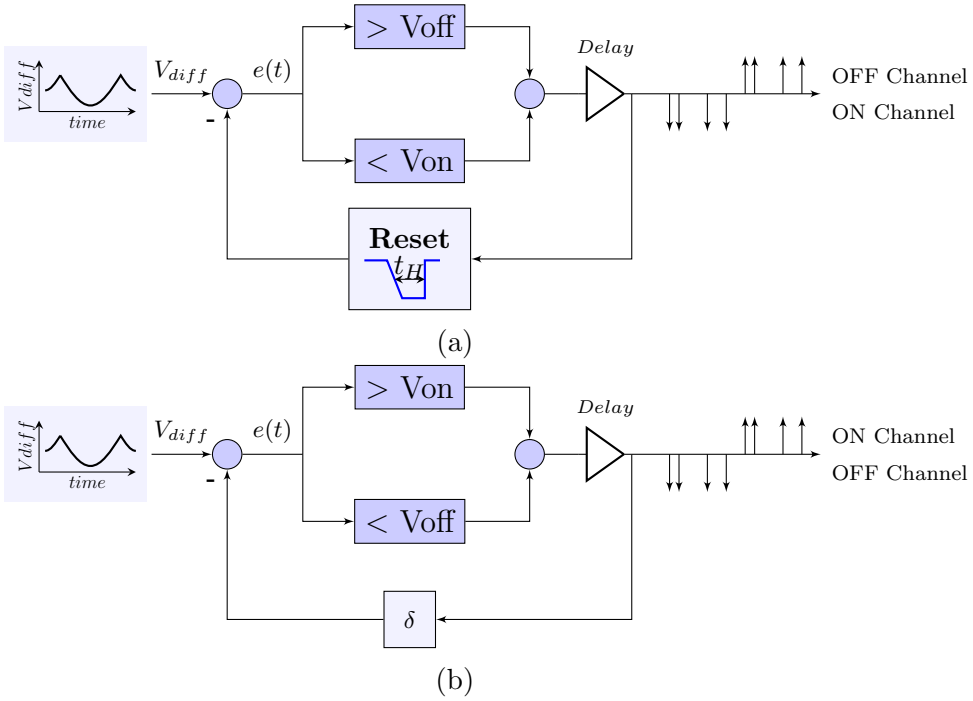


Figure 2.8: DVS information encoding methods: (a) feedback and reset and (b) feedback and subtract [21].

Figure 2.9 illustrates the ADMDVS pixel as a building block. It has the same input and output ports of the DVS pixel that was shown in Figure 2.2. The only difference at the instance symbol is that CA has been changed in order to be active at the low level, rather than high. The ADMDVS pixel uses the same AER system that is used by the DVS pixel.

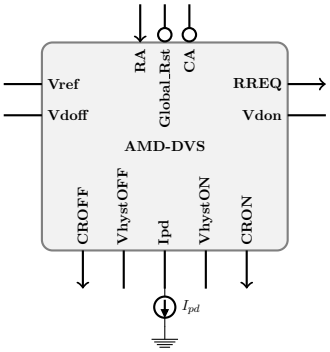


Figure 2.9: ADMDVS pixel instance.

The ADMDVS pixel basic block diagram is shown in Figure 2.10. Differently from the conventional DVS sensor (Figure 2.2), which always produces *on* events even if temporal contrast change events do not occur [4], [15], [22], the ADMDVS pixel do not generate *on* events in the absence of input temporal change. The undesired *on* events are regarded as temporal noise, which is due to the reset transistor

charging (by means of source-to-bulk leakage current) the operational amplifier inverting input up to the supply voltage V_{DD}. The TIA block is the transimpedance amplifier. It is equal to the TIA in DVS and ATIS pixels. The SF (source follower) drives the large capacitive load that corresponds to the CC-PGA (capacitively-coupled programmable gain amplifier) input. The CC-PGA replaces the differencing circuit in the DVS and ATIS pixels. The CC-PGA does not have a reset transistor, so that its input signal flow never gets interrupted. The ADM makes it possible to adjust the output level whenever an event occurs. Finally, logical circuits control all ADM switches and handle the communication links with AER systems.

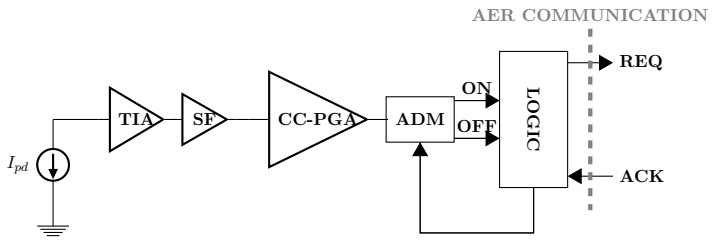


Figure 2.10: ADMDVS basic block diagram [22].

In the original ADMDVS design [22], the authors propose using a programmable close loop gain, in order to make several TCS levels possible. We designed the differencing circuit with a fixed closed loop gain. In Section 2.3.1, we describe the implementation of the feedback and subtract encoding method in ADMDVS.

2.3.1 ADM in DVS

To understand how the ADM technique eliminates the DVS self-timed reset, we analyze the circuit in Figure 2.11. The operation of the circuit is as follows: initially, all switches are off. So, after some time we have $V_{out} = V_{FB} = V_{ref}$. Let us assume that, initially, there is a positive gradient (i.e. temporal variation) at the input V_{diff}, so that is starts, for example, to decrease. In that case, V_{out} would start to increase. If V_{out} increases enough, so that is crosses the V_{refh} threshold, then an *on* event is generated at V_{ON}. The *on* event is transmitted to an AER system external to the pixel array. The asynchronous logical circuit activates the S_3 switch through ϕ_h control signal, so that V_x reaches V_{refh}. The S_3 switch remains active until an acknowledgment signal arrives from the AER system. After having received the acknowledgment signal, the asynchronous logical circuit actives the S_1 switch through ϕ_s control signal, which causes the V_{out} signal to be subtracted by a δ value after some time. The δ value is defined as $|V_{refh} - V_{ref}|$, which is equal to $|V_{refl} - V_{ref}|$.

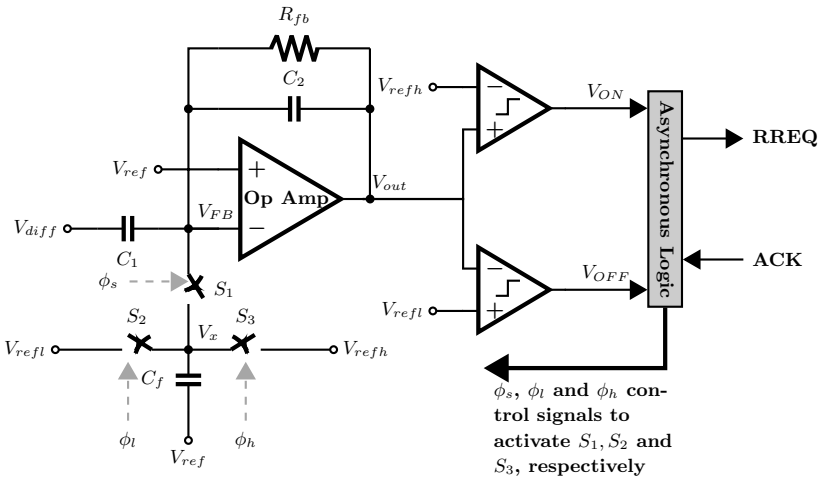


Figure 2.11: Asynchronous delta modulation circuit [21].

Let us define t_1 and t_2 as the time instants at which S_3 and S_1 are activated. By taking into account the conservation of charge stored in C_1, C_2, and C_f, we have [21]:

$$\begin{aligned}(V_{in}(t_1) - V_{FB}(t_1))C_1 + (V_{out}(t_1) - V_{FB}(t_1))C_2 + (V_{ref}(t_1) - V_x(t_1))C_f = \\ (V_{in}(t_2) - V_{FB}(t_2))C_1 + (V_{out}(t_2) - V_{FB}(t_2))C_2 + (V_{ref}(t_2) - V_x(t_2))C_f \end{aligned} \quad (2.12)$$

After V_{ref} was defined, it does not change during the entire exposure measurement interval. It is thus the same at t_1 and at t_2. The same reasoning applies for V_{FB}. We also have $V_{in}(t_1) \approx V_{in}(t_2)$. We expect that at t_2 the V_{out} signal reaches V_{ref}. Then, $V_x(t_1) = V_x(t_2) + \delta$. To simplify the analysis, we assume that $C_f = C_2$. Manipulating Equation (2.12) according to these assumptions yields:

$$V_{out}(t_2) = \begin{cases} V_{out}(t_1) - \delta, & \text{for an } \textit{on event}, \\ V_{out}(t_1) + \delta, & \text{for an } \textit{off event}. \end{cases} \quad (2.13)$$

This result is an approximation, because V_{in} is never interrupted (i.e. reset), so that V_{out} may be changing (i.e. increasing or decreasing) at any given time. The time delay that is created by the handshaking with the external AER system causes V_{out} not to reach V_{ref} exactly. The error in V_{out} accumulates across all measurement intervals, so that the number of spikes is somewhat higher (or lower) than the reference spike number that is obtained with zero time delay.

2.3.2 ADMDVS Model

The ADMDVS pixel model is similar to the DVS pixel. The V_{diff} signal is the same, but the total closed loop gain is $A = A_{diff}A_{ADM}$, where A_{diff} is the differencing circuit closed loop gain, and A_{ADM} is the ADM circuit closed loop gain. Each request type is generated according to Equation (2.14), and the operating point after each event is computed according to Equation (2.15):

$$\text{Request Type} = \begin{cases} on, & V_{out} > V_{refh}, \\ off, & V_{out} < V_{refl}, \\ no\ event, & \text{otherwise.} \end{cases} \tag{2.14}$$

The operating point for each event is calculated as:

$$\text{Operating Point} = \begin{cases} V_{out} - \delta, & \text{for an } on \text{ event}, \\ V_{out} + \delta, & \text{for an } off \text{ event}, \\ V_{out}, & otherwise. \end{cases} \tag{2.15}$$

2.4 Asynchronous Logical Circuit

According to the existence of a global time reference signal, digital circuits are divided into two large classes [23]. The first digital circuit class, which is denoted as the class of *synchronous circuits*, contains circuits whose behavior depends on a global timing signal. Tasks and communication among different sub-circuits are synchronized according to the global timing signal. The other digital circuit class, which is denoted as the class of *asynchronous circuits*, contains circuits that do not have a global timing signal. For communication among themselves, different sub-circuits use handshaking protocols. The asynchronous approach has recently received considerable attention from researchers in industry and in academia, because of some advantages that it has with respect to the synchronous approach [23], [24]:

- *Lower power consumption*: only required circuit parts respond to a specific task. Once that task is accomplished, those parts go back into an idle state, unless they are required for the next task;
- *High operating speed*: specific sub-circuits or circuit parts are assigned to specific tasks. Such sub-circuits are faster than their synchronous counterparts;
- *No clock distribution or clock skew problems*: because the clock signal arrives at different times in different sub-circuits, the clock period in synchronous systems must be carefully designed to ensure correct operation. Asynchronous

circuit design does not have that problem, simply because asynchronous circuits do not have a global clock;

- *Automatic adaptation to physical properties*: because of different causes such as fabrication process variations, temperature variation, power supply variation, and so on, digital circuits present different delay types. In synchronous digital circuit design, the solution topology and complexity vary according to the delay type. Asynchronous digital circuits usually adapt to delay type variations [23].

In spite of the asynchronous circuit advantages, we must still take into account wiring and gate delays, in order to improve system performance and communication among different sub-circuits. The communication among different sub-circuits is performed according to a handshaking protocol. Two or more sub-circuits are connected without any clock. In this dissertation, we use the four-phase handshaking protocol, which is also called *return-to-zero* protocol. Figure 2.12 shows a typical return-to-zero protocol timing diagram. The communication between the sender and the receiver has four phases: i) the sender activates the REQ signal in order to request data transfer; ii) when the receiver detects the high level at REQ, it activates (rises) the ACK signal for acknowledgment. At this point, the receiver reads data; iii) when the sender detects the high level at ACK, it sets the REQ signal to a low voltage level; iv) the receiver detects the low level at REQ, and then it finishes the communication by sending a low-voltage ACK signal to the sender.

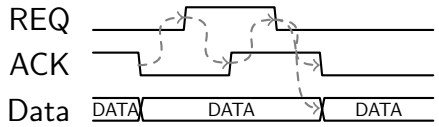


Figure 2.12: Four-phase handshaking protocol timing diagram.

In the communication between the pixel array and AER systems (in the DVS, ATIS or ADMDVS cases), the transmitted data correspond to the addresses of the pixels where the events occur. The sender of the four-phase protocol is implemented inside each pixel of the pixel array. Within each pixel, the address information is generated by an encoder with a particular time delay. The REQ signal must be generated with a similar time delay, so that the receiver reads valid information.

2.4.1 Delay-Insensitive Designs

In practical applications of asynchronous digital circuits, the designer must know the extent to which the signal processing is affected by gate and wire time delays, as well as by different delays that are caused by fabrication process variations. With

respect to robustness to time delays, the most robust asynchronous digital circuits are achieved by delay-insensitive (DI) designs [23]. In DI designs, the amount of delay is usually not important. However, DI designs are complex and, usually, not suitable for large-scale circuit implementation. To reduce complexity, quasi-delay-insensitive (QDI) designs have been proposed. A QDI circuit is defined, in [25], as a circuit whose "*correct operation is independent of the delays of gates and wires except for certain wires that form isochronic forks*". Figure 2.13 illustrates the isochronic fork concept. In a fork, the output of a logical gate is used by more than one logical gate. The isochronic assumption applies to the logical OR gate in Figure 2.13: the A signal arrives before the B signal. This means that C is stable after a minimum delay, which corresponds to the sum of the time delays of the inverters shown in the figure. Asynchronous digital circuit design becomes more flexible if QDI circuits are used. We assume that the AER system mentioned in Section 2.5 is a QDI circuit. To make sure it works properly, controlled delay circuits are inserted throughout the signal path (in pixel design, in our case).

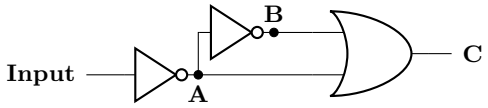


Figure 2.13: A fork and the isochronic assumption.

2.5 Address-Event Representation

Address-Event Representation (AER) is a time-multiplexing technique. Event temporal order is preserved. As AER development usually takes place in biological contexts, the building blocks in charge of generating, transmitting, or receiving spikes are usually referred to as *neurons*, with a particular emphasis on the neurons that generate spikes. In vision sensors, a spike-generating neuron is simply referred to as a *pixel*. Among some of the AER main features [26], we point out that: i) communication is active only if a pixel has at least one spike to be transmitted; ii) every pixel is represented uniquely by a few bits, and for all pixels those bits are transmitted through the same data bus; iii) information transmission is asynchronous, which has a positive impact on bandwidth and power consumption [27].

In figure 2.14 is depicted the concept of AER. On the left side, we have the sender where is encoding and conveyed each neuron with some event. On the right side, we have the receiver system, which decoding the information, and updates the respective neuron with its new value. The sender and receiver are linked by means of Digital Bus. In AER communication the time represents itself.

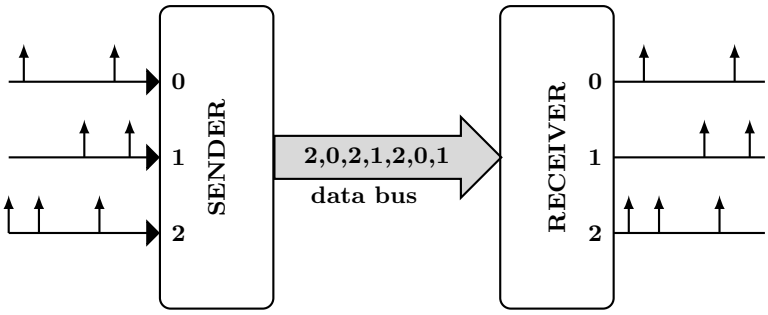


Figure 2.14: AER concept [28].

At the sender side, the AER system involves an encoder and a decision mechanism. The encoder assigns a unique integer number for each pixel, and tries to transmit this integer number whenever the pixel generates an event. The decision mechanism organizes pixel access to the digital data bus, so that each pixel is usually able to transmit its own event to a receiver (which is external to the pixel array, and often placed off-chip). A collision (i.e. a transmission conflict) occurs when two or more pixels generate events simultaneously. To decide which of the pixels will be granted access to the digital data bus, the decision mechanism must apply some criterion. The decision mechanism may be *arbitered* or *unfettered* (i.e. unconstrained) [28]. Ideally, the arbitered decision mechanism ensures that every spike generated by any pixel will be transmitted. Collisions are solved by queuing requests, which may lead to timing precision loss. The unfettered decision mechanism allows collisions to keep unattended to, which prevents timing precision loss, but leads to data loss (i.e. spikes not registered) on the other hand. For typical AER applications, the use of an arbitered decision mechanism is recommended [28]. For simplicity, the decision mechanism may also take decisions at random, to solve ties. Since this dissertation does not focus on AER system design, we will use the random decision method.

The AER usually has a tree structure such as the one shown in Figure 2.15. The j-th arbiter at the i-th decision level has label $A_{i,j}$. If the number of inputs is N, then the number of levels is $\lceil \log_2 N \rceil$. For example let us assume that, at a given time instant, only In0 and In4 generate simultaneous transmission requests. If all arbiters at Levels 1 and 2 have the same time delay, then the arbiter at Level 3 ($A_{3,0}$) must decide the winning input. It takes that decision randomly, let us say, in favor of In4. To encode the winning pixel index, the path connecting $A_{3,0}$ to In4 is used. To implement the encoder, which is located close to the inputs before Level 1, simple multiplexers are used (Section 3.10). The winning input transmission request finally generates a transmission request to the receiver. This transmission request is accompanied, according to Figure 2.15, by the encoded winning pixel address, which is 4. The AER attends to one input request at a time (In4, in this example). It keeps the other request, In0 in the example, blocked (i.e. without an acknowledgment)

until the In4 request has been acknowledged by an off-chip receiver according to Figure 2.15.

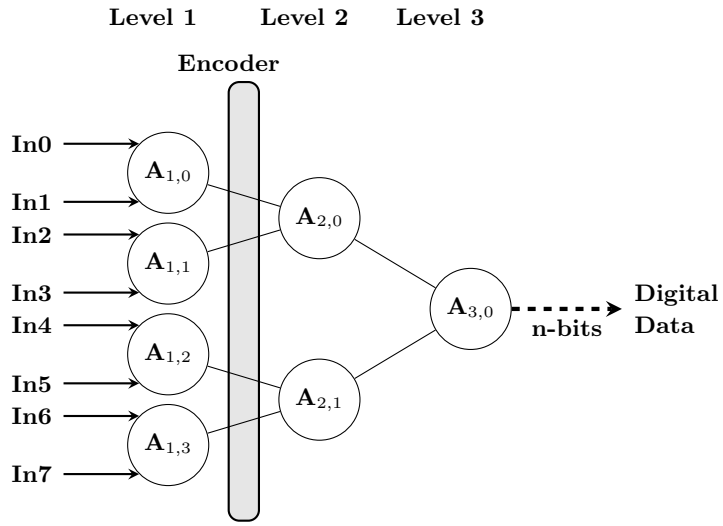


Figure 2.15: Eight-input arbitered AER example. At each level, a winning input is selected pairwise. Only one index, corresponding to the pixel having a circuit path connecting it to the winning input at Level 3, is encoded for transmission through the data bus.

2.6 Integrated Circuit Design based on the g_m/I_D Method

Conventional integrated circuit design methods take into account two transistor operation regions, namely strong or weak inversion. Such design method do not consider the moderate region, where most circuits could achieve a low ratio between power consumption and bandwidth, as well as reasonable transistor size. The key parameter in conventional integrated circuit design is the *overdrive voltage* V_{ov}. The n-channel MOS transistor is in strong inversion if $V_{ov} > V_{gs} - V_T$ and $V_{gs} > V_T$, where V_{gs} is the voltage difference between the gate and source terminals of the MOS transistor, and V_T is the technology threshold voltage. In the weak inversion regime, V_{ov} is not important, because $V_{gs} < V_T$ leads to the transistor drain current I_D being entirely generated by charge carrier diffusion [29], which does not depend on V_{ov}. In low-power circuits, V_{ov} could be lower than the V_{ov} values typically used in strong inversion, with a minimum value at $3\phi_t$, where $\phi_t = 26$ mV is the thermal voltage. The g_m/I_D method, on the other hand, takes into account all operation regions [30]. It focuses on the ratio between transistor transconductance (g_m) and DC drain current (I_D). This ratio is called *efficiency of the transconductance*. It is analyzed versus the *normalized drain current* $I_D L/W$. Using the g_m/I_D ratio as a

key parameter leads to control over transistor W/L ratios throughout the circuit, and those ratios may subsequently be used for reducing circuit area. It also leads to control over transistor operation regime, which is useful for a low-power design.

The g_m/I_D ratio is shown, in Equation (2.16), as a function of $I_D L/W$:

$$\frac{g_m}{I_D} = \frac{1}{I_D}\frac{\partial I_D}{\partial V_G} = \frac{\partial\left(\log I_D\right)}{\partial V_G} = \frac{\partial\left\{\log\left[\frac{I_D}{\left(\frac{W}{L}\right)}\right]\right\}}{\partial V_G}. \tag{2.16}$$

In Equation 2.17, the g_m/I_D is shown as a function of the EKV model *inversion coefficient* I_D/I_s:

$$\frac{g_m}{I_D} = \frac{1}{n\phi_t}\frac{1-\exp\left(-\sqrt{\frac{I_D}{I_s}}\right)}{\sqrt{\frac{I_D}{I_s}}}, \tag{2.17}$$

Replacing the I_D/I_s term in Equation (2.17) by $I_D/(2n\phi_t^2 K_p W/L)$, we have g_m/I_D as a function of the normalized drain current $I_D L/W$, as shown in Equation (2.18):

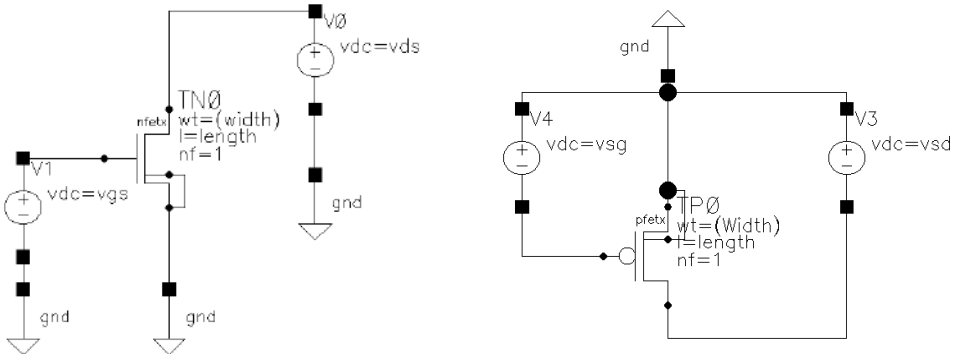
$$\frac{g_m}{I_D} = \frac{\sqrt{2n\mu_n C_{ox}}}{n}\frac{1-\exp\left(\frac{-\sqrt{I_D/(\frac{W}{L})}}{\phi\sqrt{2n\mu_n C_{ox}}}\right)}{\sqrt{I_D/(\frac{W}{L})}}. \tag{2.18}$$

Equation (2.16) relates g_m/I_D, regardless of transistor size or W/L ratio, to the derivative of $\log(I_D)$ with respect to the gate voltage V_G. The maximum g_m/I_D value is achieved in subthreshold regime, and it decreases as the transistor is taken towards strong inversion. By exploring many g_m/I_D versus $I_D L/W$ curves, a designer may reduce circuit area.

In a semi-empirical application of the g_m/I_D method [31], a g_m/I_D ratio look-up table is extracted from simulations of an advanced transistor model. Using transistor widths large enough to avoid model border effects is recommended. The g_m/I_D method is also applied to the design of an intrinsic gain stage. We follow a similar semi-empirical approach in this dissertation.

Figure 2.16 shows the transistor setup to be used for extraction of g_m/I_D versus I_D curves. In this figure, we assume generic W and L for both transistors. The simulation is a DC sweep of V_{gs} for the n-channel transistor, and of V_{sg} for the p-channel transistor, with a carefully chosen (i.e. small enough) step size. To specify the desired g_m/I_D curve for the n-channel transistor, we use a specific command line expression in Spectre : *(deriv(i("TN0:d"?result"DC"))/i("TN0:d"?result"dc"))*. In the p-channel transistor case, the *TN0* object is replaced by *TP0*.

Figure 2.17 depicts several g_m/I_D versus I_D curves extracted from an advanced n-channel MOS transistor model [32]. The matching between the simulated curves and the theoretical curve gets better as the transistor length gets larger. According



(a) (b)

Figure 2.16: Transistor setup for the extraction of g_m/I_D versus I_D curves: (a) n-channel transistor, and (b) p-channel transistor.

to the EKV model, we have $(g_m/I_D)_{max} = 1/(n\phi_t)$ [31].

In Figure 2.19, similar results are shown (simulated g_m/I_D versus I_D curves) for the case of a p-channel transistor model. The relationship between g_m/I_D and V_{gs}, for an n-channel transistor, is shown in Figure 2.19. The relationship between g_m/I_D and the absolute value of V_{sg} (which is denoted as V_{GS} in the plots), for a p-channel transistor, is shown in Figure 2.20. After having set the transistor dimensions to $L = 1$ μm and $W = 22$ μm, we also extracted the relationship between g_m/I_D and V_{gs} for six different V_{ds} values, which is shown in Figure 2.21 for both n-channel and p-channel transistor models.

To choose transistor width, starting from an arbitrary point of a g_m/I_D versus I_D curve corresponding to a fixed length (say $L = 1$ μm), we applied the following procedure: i) within the desired operation region, pick one g_m/I_D value and its corresponding normalized current I_D^*, and ii) find the W/L ratio according to $W/L = I_D/I_D^*$. Additionally, if the design uses g_m as a key parameter, I_D may be found according to $I_D = g_m/(g_m/I_D)^*$ before step (i) is executed.

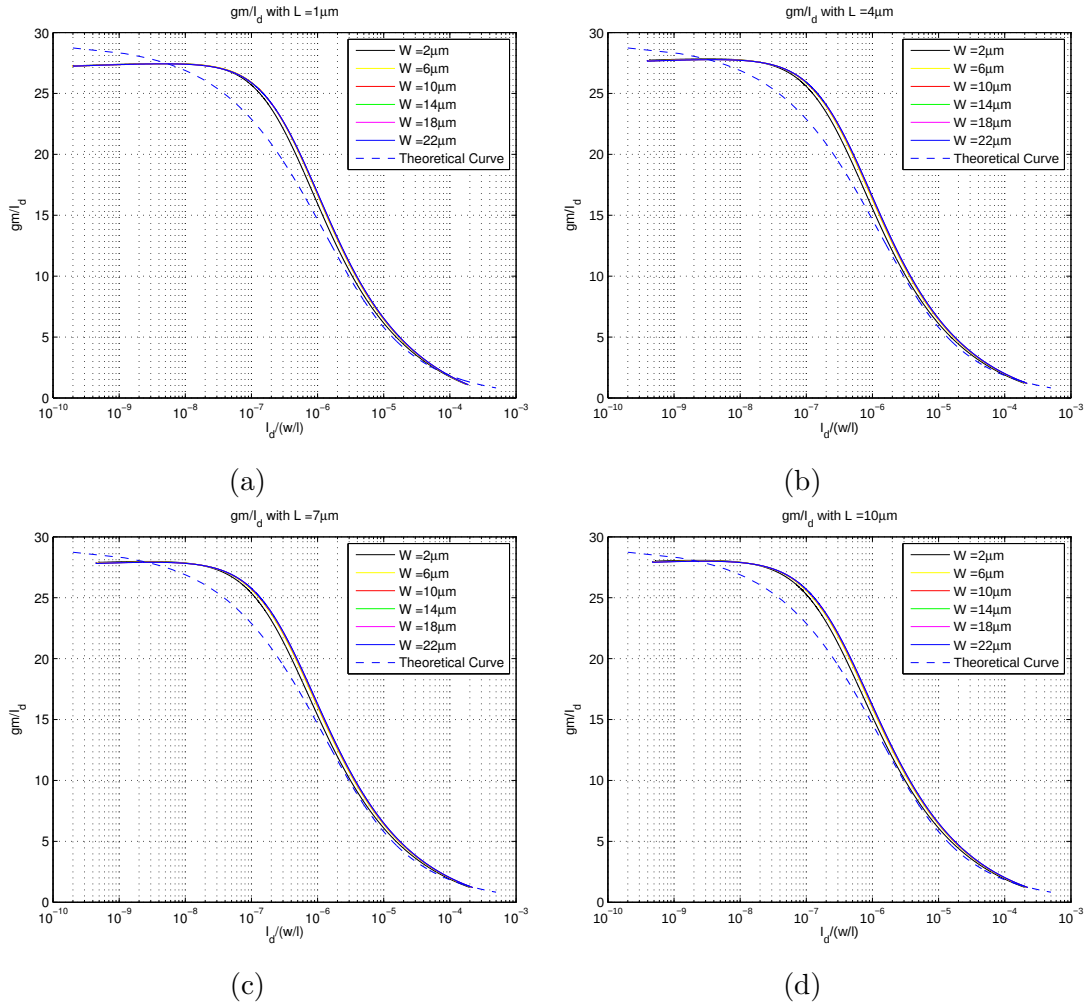


Figure 2.17: Simulated g_m/I_D versus I_D curves extracted from an n-channel transistor model. In each plot, transistor width varies from 2 μm to 22 μm. Transistor length varies as follows: (a) $L = 1$ μm, (b) $L = 4$ μm, (c) $L = 7$ μm, and (d) $L = 10$ μm.

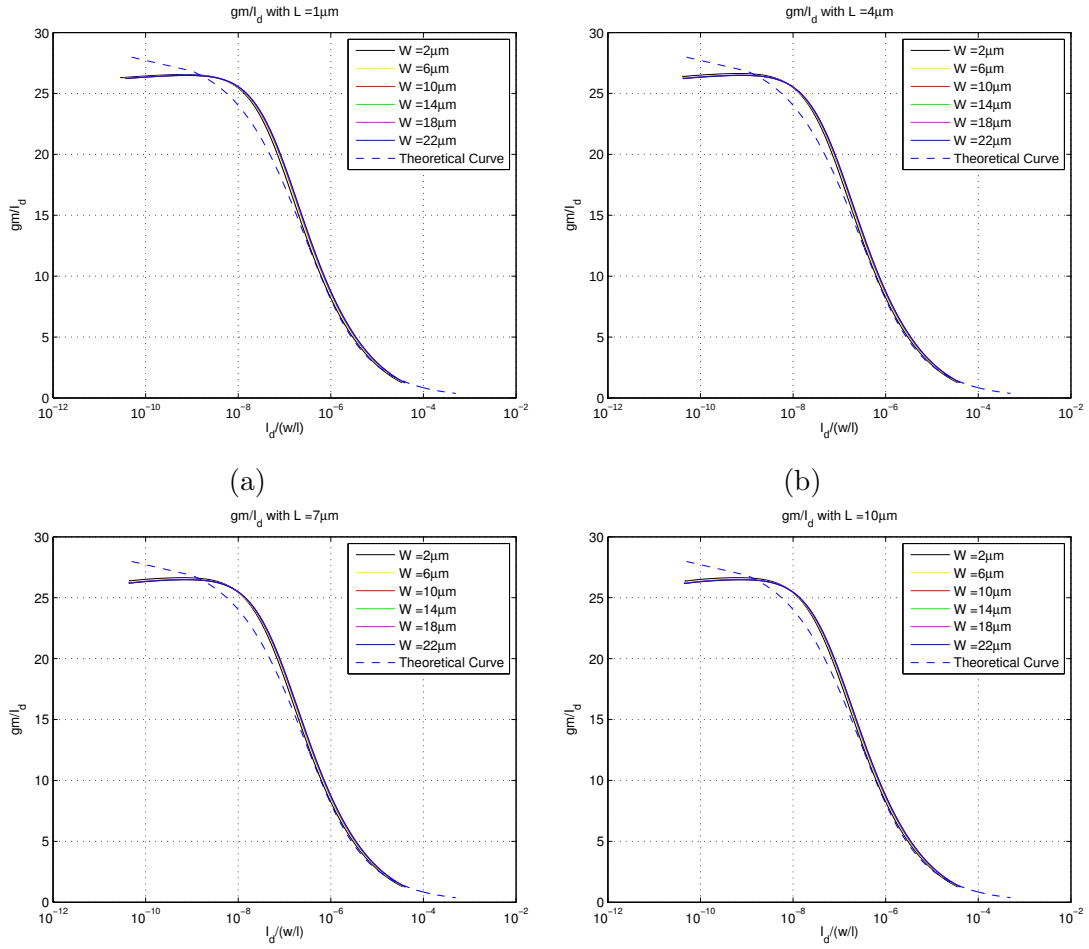


Figure 2.18: Simulated g_m/I_D versus I_D curves extracted from a p-channel transistor model. In each plot, transistor width varies from 2 μm to 22 μm. Transistor length varies as follows: (a) $L = 1$ μm, (b) $L = 4$ μm, (c) $L = 7$ μm, and (d) $L = 10$ μm.

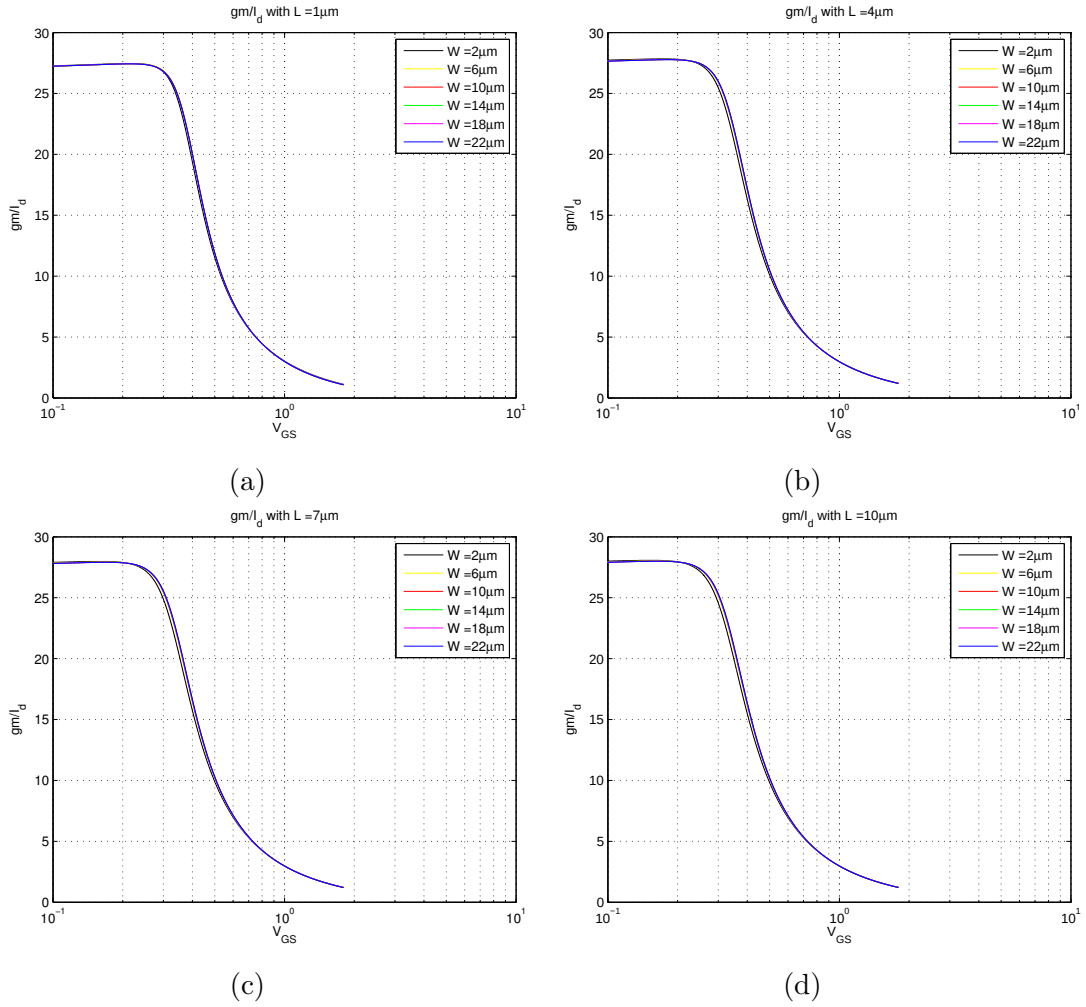


Figure 2.19: Relationship between g_m/I_D and V_{gs} extracted from an n-channel transistor model. In each plot, transistor width varies from 2 μm to 22 μm. Transistor length varies as follows: (a) $L = 1$ μm, (b) $L = 4$ μm, (c) $L = 7$ μm, and (d) $L = 10$ μm.

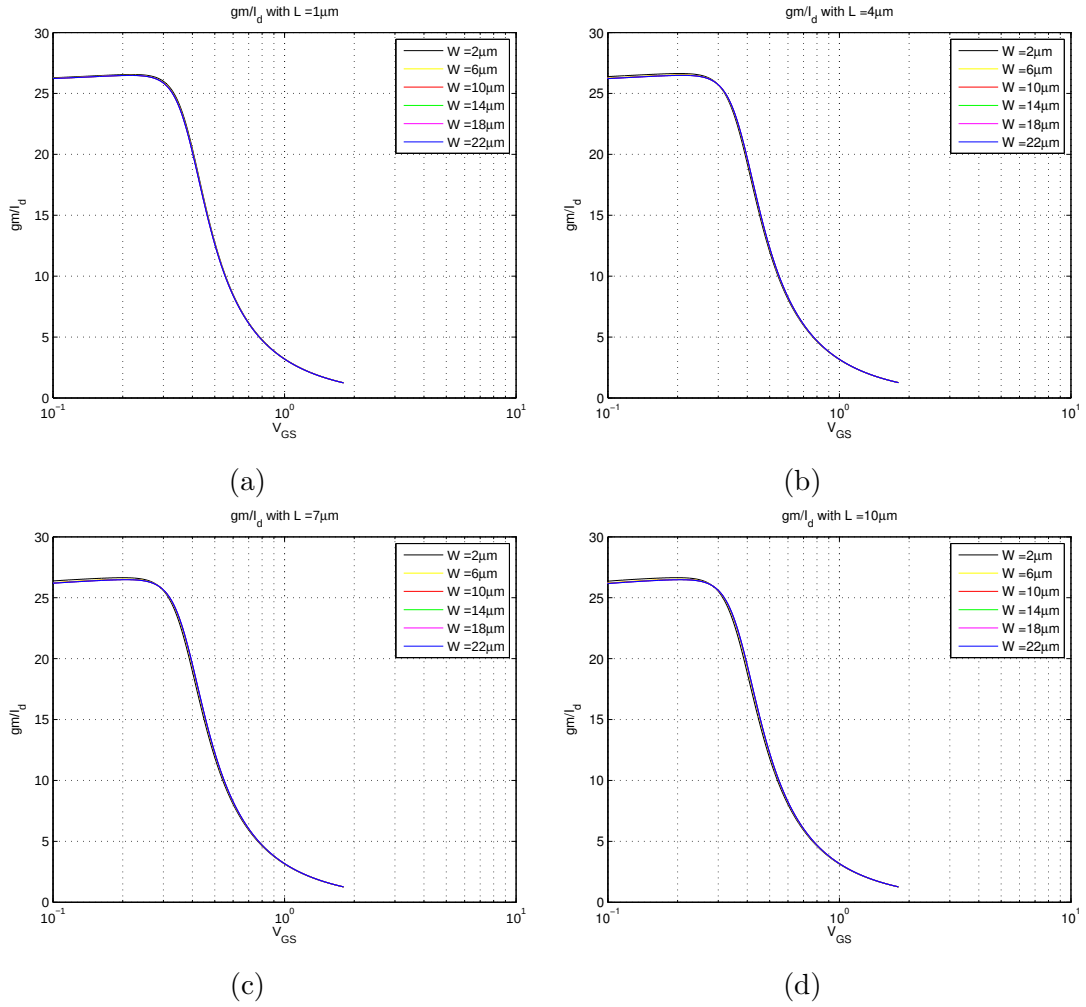


Figure 2.20: Relationship between g_m/I_D and V_{sg} extracted from an n-channel transistor model. The absolute value of V_{sg} is denoted as V_{gs} in the plots. In each plot, transistor width varies from 2 μm to 22 μm. Transistor length varies as follows: (a) $L = 1$ μm, (b) $L = 4$ μm, (c) $L = 7$ μm, and (d) $L = 10$ μm.

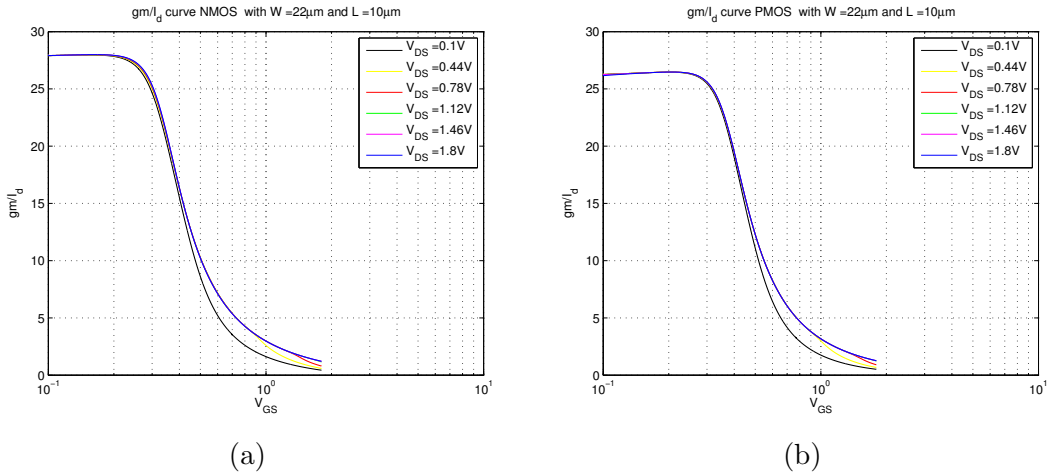


Figure 2.21: Relationship between g_m/I_D and V_{gs} in the case of V_{ds} variation: (a) n-channel transistor and (b) p-channel transistor.

Chapter 3

Pixel Design

In this chapter, we present the design of each pixel that is studied in this work (DVS, ATIS, and ADMDVS) using AMS (austriamicrosystems.com) CMOS 180 nm fabrication technology. The g_m/I_D methodology was only used for operational amplifier and voltage comparator design. The electrical simulation results were obtained from Cadence Spectre.

3.1 Photoreceptor based on Transimpedance Amplifier

The photosensitive cell, which is based on the logarithmic transimpedance amplifier, is the same for all pixels that are studied in this work. It is shown in Figure 3.1. It corresponds to a well-known cascode amplifier with a source-follower amplifier (i.e. the M3 transistor, which is operated in common-drain configuration) in the feedback loop [33]. The photoreceptor based on transimpedance amplifier is suitable for sensing temporal contrast change rather than absolute light intensity values. The M2 cascode transistor cancels the Miller effect that would otherwise amplify the C_{gd1} capacitance, which exists between the gate and drain terminals of M1, and it also doubles the amplifier gain if $g_{ds2} = g_{ds4}$ (M1 and M4 have the same transconductance) and if $(W/L)_1 = (W/L)_2$. The amplifier gain is $A_{amp} = -g_{m1}/g_{ds4}$.

In Figure 3.1a, the transistor sizes are as follows: $L = W = 1$ μm for M1; $L = 180$ nm and $W = 220$ nm for M2; $L = 200$ nm and $W = 300$ nm for M3; and $L = 1$ μm and $W = 220$ nm for M4 and M5. The M3 transistor (feedback amplifier) works in the subthreshold regime, because it is biased by the photodiode reverse current I_{pd}, which is very small (typically below 1 nA):

$$V_p = n\phi_t \log\left(\frac{I_{pd}}{I_s}\right) + V_T + V_d \tag{3.1}$$

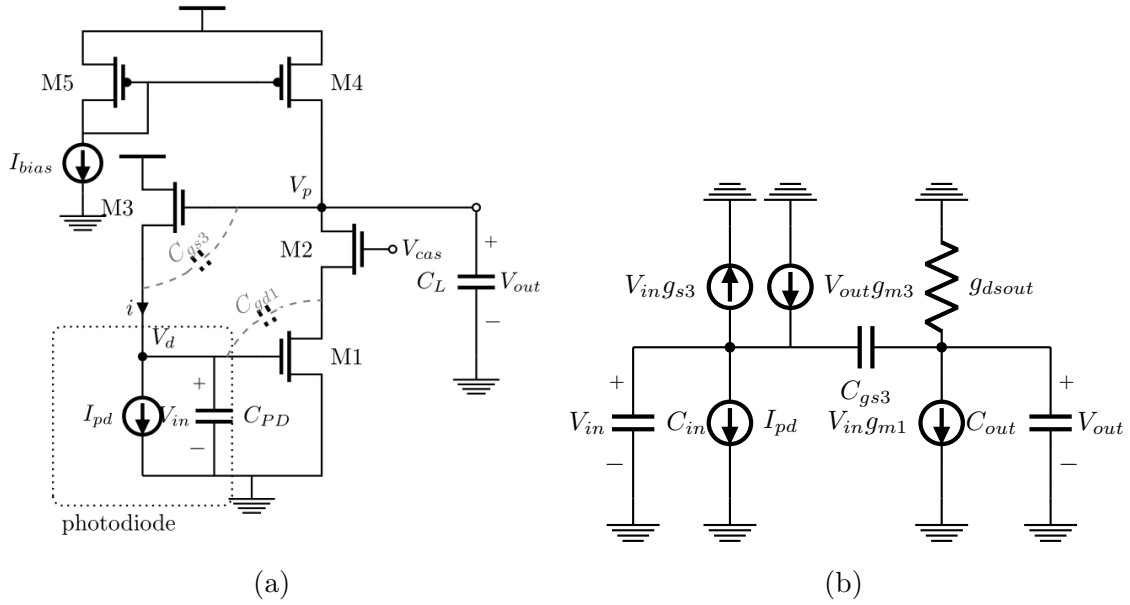


Figure 3.1: Photoreceptor based on transimpedance amplifier: (a) schematic diagram and (b) small-signal model.

I_{pd} depends on the pixel input light intensity. The M3 gate voltage is thus a logarithmic function of I_{pd}. The amplifier bias current, I_{bias}, allows modification of the photosensitive cell cutoff frequency and gain. The gain depends on M1 transconductance, which is $g_{m1} = I_{bias}(g_m/I_D)_1$.

If the photocurrent I_{pd} increases slightly, such that small-signal behavior applies, then the M3 source voltage decreases slightly (actually V_p at the M3 gate remains approximately constant, because of negative feedback, so that v_d tends to decrease, but it remains approximately constant). If the M3 source voltage decreases, then the cascode amplifier increases the M3 gate voltage, which implements a negative feedback effect on the M3 source voltage. The V_d potential remains approximately fixed, as if this node was connected to a virtual ground, which increases the amplifier bandwidth [15]. The current flowing through the M3 source is sensitive to absolute light intensity values, as well as to temporal contrast change.

To compute the photoreceptor transimpedance gain, we solve for the M3 gate voltage ($V_p = V_{out}$) in the equation that describes the exponential relationship between I_{pd} and V_p (or V_{out}), and then linearize the V_p (or V_{out}) expression around the bias point that is defined by I_{pd0}. Assuming a small I_{pd} input signal, which as denoted as ΔI, we have $V_{out}/\Delta I = \phi_t/(\kappa I_{pd0})$, where $I_{pd0} = I_s \exp(V_{G3} - nV_{S3} - V_T)/(n\phi_T)$ is a DC current corresponding to an I_{pd} operating point, and $\kappa = 1/n = 0.75$ is the slope factor.

Using a small-signal model, we can obtain the photoreceptor transfer function. By applying Kirchoff current law to both nodes in Figure 3.1b, we have:

$$\begin{aligned}(sC_{in} + g_{s3} + sC_{gs3})V_{in} - (g_{m3} + sC_{gs3})V_{out} &= -i_{pd} \\ (sC_{gs3} + sC_{out} + g_{dsout})V_{out} + (g_{m1} - sC_{gs3})V_{in} &= 0\end{aligned} \tag{3.2}$$

We define the same constants as in [34], which are listed here for convenience:

$$\mathrm{A} = \frac{\mathrm{g_{m1}}}{\mathrm{g_{dsout}}}, \tau_{\mathrm{in}} = \frac{\mathrm{C_{in}}}{\mathrm{g_{s3}}}, \tau_{\mathrm{out}} = \frac{\mathrm{C_{out}}}{\mathrm{g_{m1}}}, \alpha = \frac{\mathrm{C_{gs3}}}{\mathrm{C_{in}}}, \beta = \frac{\mathrm{C_{gs3}}}{\mathrm{C_{out}}}, \mathrm{R}_\tau = \frac{\tau_{\mathrm{in}}}{\tau_{\mathrm{out}}} = \kappa \frac{\mathrm{C_{in}}}{\mathrm{C_{out}}} \frac{\mathrm{I_{bias}}}{\mathrm{I_{pd}}}.$$

By solving for V_{out}/V_{in} in (3.2), we obtain:

$$\begin{aligned}\frac{V_{out}}{V_{in}} &= \frac{V_{out}}{i/gs_3} = \frac{A_{DC}(1 - \frac{s}{\omega_z})}{\frac{s^2}{\omega_n^2} + \frac{s}{\omega_n Q} + 1} \\ &= \frac{\gamma \cdot (1 - s\beta\tau_{out})}{s^2\gamma(1 + \alpha + \beta)\tau_{in}\tau_{out} + s\gamma[\tau_{in}(\alpha(1 + \frac{1}{A}) + \frac{1}{A}) + \tau_{out}(\beta(1 - k) + 1)] + 1},\end{aligned} \tag{3.3}$$

where ω_n is the linear system natural frequency in a canonical representation, ω_z is the frequency of a transfer function zero, A_{DC} is the DC voltage gain, Q is the filter (i.e. linear system) quality facotr, $\gamma = A/(1 + \kappa A)$, and A is the cascode amplifier voltage gain. Assuming $A >> 1$ yields $A_{DC} = 1/\kappa$. By equating the second-order expression in the second line of Equation (3.3) to the canonical second-order form in the first line of Equation (3.3), we obtain:

$$\omega_n = \sqrt{\frac{\kappa}{(1 + \alpha + \beta)\tau_{in}\tau_{out}}} \qquad Q = \frac{\sqrt{\kappa(1 + \alpha + \beta)R_\tau}}{\alpha R_\tau + 1 + \beta(1 - \kappa)} \tag{3.4}$$

For example, if typical constant values such as $\alpha = 0.05$, $\beta = 0.25$, and $\kappa = 0.75$ are chosen, then Q can be considered as a function of R_τ, as shown in Figure 3.2, which is useful for design purposes.

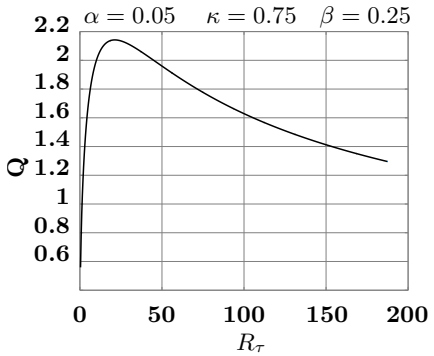


Figure 3.2: Quality factor versus R_τ.

The photoreceptor response to a input current pulse is shown in Figure 3.3, for

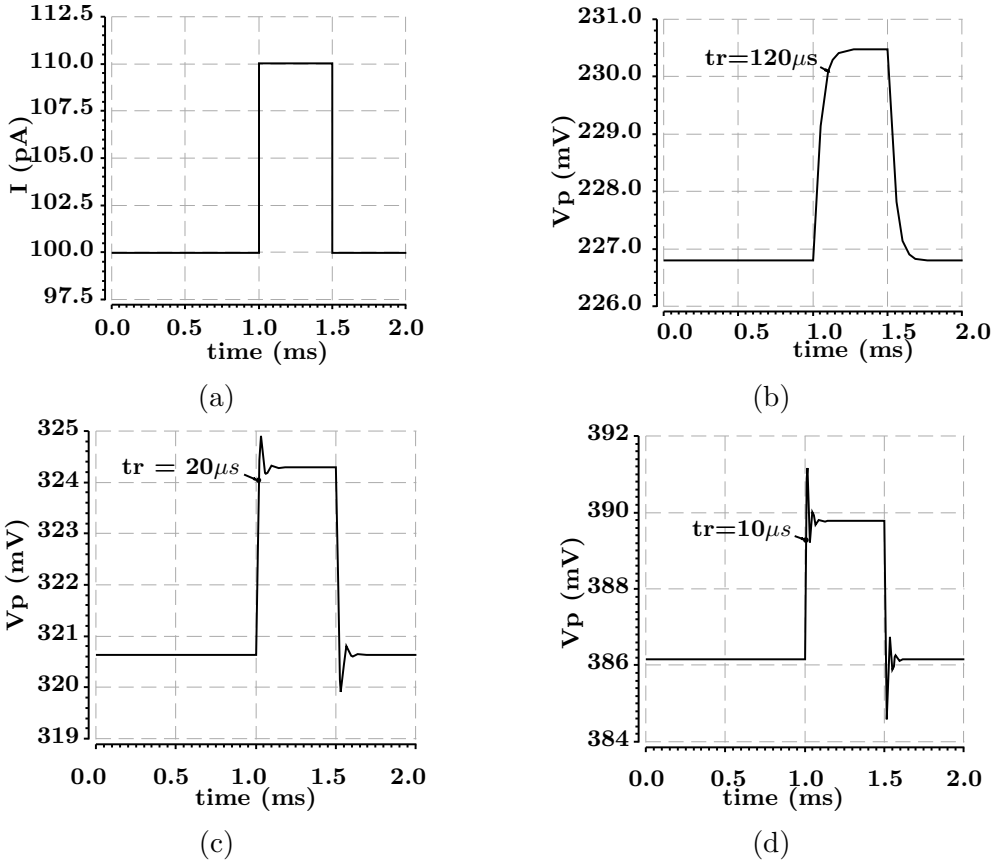


Figure 3.3: Photoreceptor input current pulse (a), and photoreceptor pulse response for three different I_{bias} values: (b) 50 pA, (c) 300 pA, and (d) 1 nA.

three different I_{bias} values: 50 pA, 300 pA, and 1 nA. As I_{bias} gets larger, the rise time (t_r) gets shorter, and Q gets larger.

3.2 Operational Amplifier

Whenever an operational amplifier is required in this work (in DVS, ATIS or AD-MDVS designs), the circuit shown in Figure 3.4 is used. It is a two-stage operational amplifier, and the transistor sizes as follows: $L = 1$ μm and $W = 1.71$ μm for M1a and M1b; $L = 1$ μm and $W = 3.55$ μm for M2; $L = 1.34$ μm and $W = 0.5$ μm for M3a and M3b; $L = 1$ μm and $W = 16.35$ μm for M4; and $L = 1$ μm and $W = 3.43$ μm for M5. The C_m capacitance value is 55.3 fF. The transistor sizes are obtained from Table 3.1, and the design procedure is explained next. A complete analysis of this circuit may be found elsewhere [35], but for convenience some details are presented here.

The transfer function of the two-stage operational amplifier in Figure 3.4 is shown in Equation (3.5):

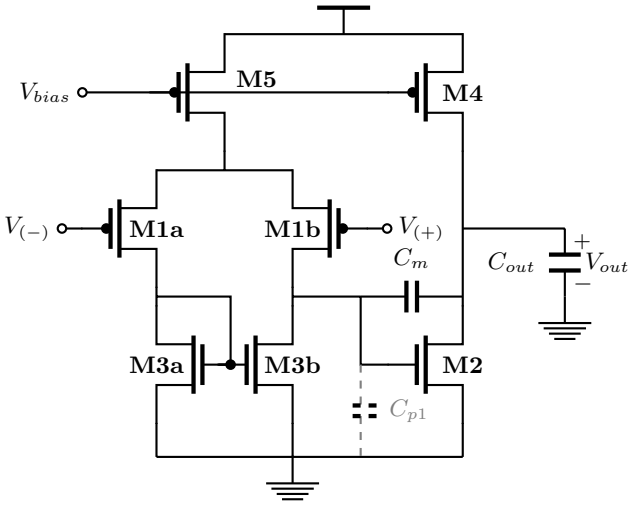


Figure 3.4: Two-stage operational amplifier.

$$\frac{V_o(s)}{V_{in}(s)} = \frac{\frac{g_{m1}g_{m2}}{G_I G_{II}}(1 - sC_m/g_{m2})}{s^2\left[\frac{C_{p1}C_{out}+CmC_{p1}+C_mC_{out}}{G_I G_{II}}\right] + s\left[\frac{C_{p1}+C_m}{G_I} + \frac{C_{II}+C_m}{G_{II}} + \frac{g_{m6}C_m}{G_I G_{II}}\right] + 1}, \tag{3.5}$$

where G_I is the first-stage output conductance and G_{II} is the second-stage output conductance. The first and second-stage output transconductances are g_{m1} and g_{m2}. The two poles and the zero of the transfer function shown in Equation (3.5) are:

$$P_1 = \frac{-G_I G_{II}}{g_{m2}C_m} \tag{3.6}$$

$$P_2 = \frac{-g_{m2}C_m}{C_{p1}C_{out} + C_{out}C_m + C_{p1}C_m} \tag{3.7}$$

$$z = \frac{g_{m2}}{C_m} \tag{3.8}$$

We next compute three figures of interest [14], which are namely: unity-gain bandwidth (GBW), non-dominant pole (NDP) and z/GBW ratio (Z):

$$\text{GBW} = \frac{g_{m1}}{C_m} \tag{3.9}$$

$$\text{NDP} = \frac{P_2}{\text{GBW}} = \frac{g_{m2}}{g_{m1}} \frac{C_m^2}{C_{p1}C_{out} + C_m(C_{p1} + C_{out})} \tag{3.10}$$

$$\text{Z} = \frac{z}{\text{GBW}} = \frac{g_{m2}}{g_{m1}} \tag{3.11}$$

$$C_m = 0.5\frac{\text{NDP}}{\text{Z}} \left[C_{p1} + C_{out} + \sqrt{(C_{p1} + C_{out})^2 + 4C_{p1}C_{out}\frac{\text{Z}}{\text{NDP}}} \right] \tag{3.12}$$

Note, from Equation (3.9), that the operational amplifier speed (GBW) depends linearly on M1 transconductance, which is $g_{m1} = I_{D1}(g_m/I_D)_1$. In subthreshold operation, g_m/I_D may be expressed as a constant value. Combining Equations (3.9) to (3.11) yields Equation (3.12):

Using the analysis expressions in Equations (3.9) to (3.12), we can apply the g_m/I_D method to design the two-stage operational amplifier. For each transistor, we might get one particular coordinate pair $(g_m/I_D; I_D L/W)$ from the g_m/I_D versus $I_D L/W$ curves, taking into account a particular operation region (for example, inversion coefficient equal to 0.1), as it was explained in Chapter 2. An alternative g_m/I_D approach consists in using offset voltage specifications. It is possible to know in advance (i.e. before the design) the inversion level at which each transistor reaches such specifications. We assumed that only one inversion level will be applied for the entire design. To yield an operational amplifier with enough phase margin, the Z and NDP values must be chosen carefully. Choosing NDP ≥ 2.2 and $Z \geq 10$ leads to phase margin between 60^{O} and 70^{O}. After the coordinate pairs $(g_m/I_D; I_D L/W)$ are selected for each transistor, we design the operational amplifier in five steps [31]:

1. *Initial Conditions*: choose an arbitrary value for C_m. For example, $C_m = 50$ fF.

2. *M1 and M2 Sizing*: Calculate $g_{m1} = \text{GBW} \times C_m$ and $g_{m2} = Z g_{m1}$. To find the W/L ratio for M1, compute $I_{D1} = g_{m1}/(g_m/I_D)_1$, and then use I_{D1} in $(W/L)_1 = I_{D1}/I^*_{D1}$. Similarly, for M2, compute $I_{D2} = g_{m2}/(g_m/I_D)_2$, and then use I_{D2} in $(W/L)_2 = I_{D2}/I^*_{D2}$.

3. *M3, M4, and M5 Sizing*: To avoid systematic offset, we assume that $(g_m/I_D)_3 = (g_m/I_D)_2$. The W/L ratio for M3 is then found according to $(W/L)_3 = I_{D1}/I^*_{D2}$. To find the W/L ratio for M5, we consider the maximum tail current of the differential amplifier, then choose a normalized current I^*_{D5} within the adequate inversion level, and then solve $(W/L)_5 = 2I_{D1}/I^*_{D5}$. Finally, we find M4 size minimizing systematic offset according to $(W/L)_4 =$

$1/2(I_{D2}/I_{D1})(W/L)_5$.

4. *Re-calculating* C_m: the C_{p1} and C_{out} values are computed according to the following expressions:

$$C_{p1} = C_{diff1} + C_{diff3} + C_{g2} \tag{3.13}$$

$$C_{out} = C_{p2} + C_L = C_{diff2} + C_{diff4} + C_L \tag{3.14}$$

where C_{p1} is the first-stage output capacitance, C_{out} is the second-stage output capacitance, C_{diff} is the diffusion capacitances, LDS is the source-drain metal width, C_{jw} is the junction capacitance, C_{jsw} is the junction sidewall capacitance and C_{ov} is the overlap capacitance. By plugging the AMS 180 nm technology parameters into the expressions above, we find $C_{jn} = 1.12$ fF, $C_{jwn} = 0.155$ fF, $C_{jp} = 1.15$ fF, $C_{jwp} = 0.09$ fF, $C_{ovn} = 0.33$ fF, and LDS = 0.26. The other parameters such as: C_{diff} and C_{g2} depend on the transistor width (W) and are defined next:

$$C_{diff1} = C_{jp}\mathrm{W}_{1b}\mathrm{LDS} + C_{jwp}(2\mathrm{W}_{1b} + 2\mathrm{LDS}) \tag{3.15}$$

$$C_{diff2} = C_{jn}\mathrm{W}_2\mathrm{LDS} + C_{jwn}(2\mathrm{W}_2 + 2\mathrm{LDS}) \tag{3.16}$$

$$C_{diff3} = C_{jn}\mathrm{W}_{3b}\mathrm{LDS} + C_{jwn}(2\mathrm{W}_{3b} + 2\mathrm{LDS}) \tag{3.17}$$

$$C_{diff4} = C_{jp}\mathrm{W}_4\mathrm{LDS} + C_{jwp}(2\mathrm{W}_4 + 2\mathrm{LDS}) \tag{3.18}$$

$$C_{g2} = C_{gs2} + C_{gb2} + C_{ovn}\mathrm{W}_2 \tag{3.19}$$

The C_{gs} and C_{gb} capacitances were extracted for different (W/L) values (different transistors) versus i_f (inversion coefficient), so that the simulation results get closer to the real circuit behavior.

5. *Back to Step 1*: recompute C_m according to Equation (3.12). Iterate over these five steps until the C_m value converges.

We designed the operational amplifier to work up to 1 MHz with a maximum 200 fF capacitive load. The C_m capacitance value is 55.3 fF, and transistor dimensions are presented in Table 3.1. Simulation results (two-stage operational amplifier frequency response) are shown in Figure 3.5, for different bias current values (I_{bias}). Operational amplifier GBW varies from 200,0 kHz to 1 MHz, and the phase margin remains constant at 60^o, as the bias current varies from 5 nA to 30 nA. Constant

phase margin and GBW variation from 200,0 kHz to 1 MHz are useful features, because they allow image sensor power consumption selection according to the speed required for sensing specific temporal contrast change events.

Table 3.1: Two-stage operational amplifier transistor sizes.

Transistor	W (μm)	L (μm)
M1a,M1b	1.76	1
M2	3.64	1
M3a,M3b	0.5	1.3
M4	16.8	1
M5	3.52	1

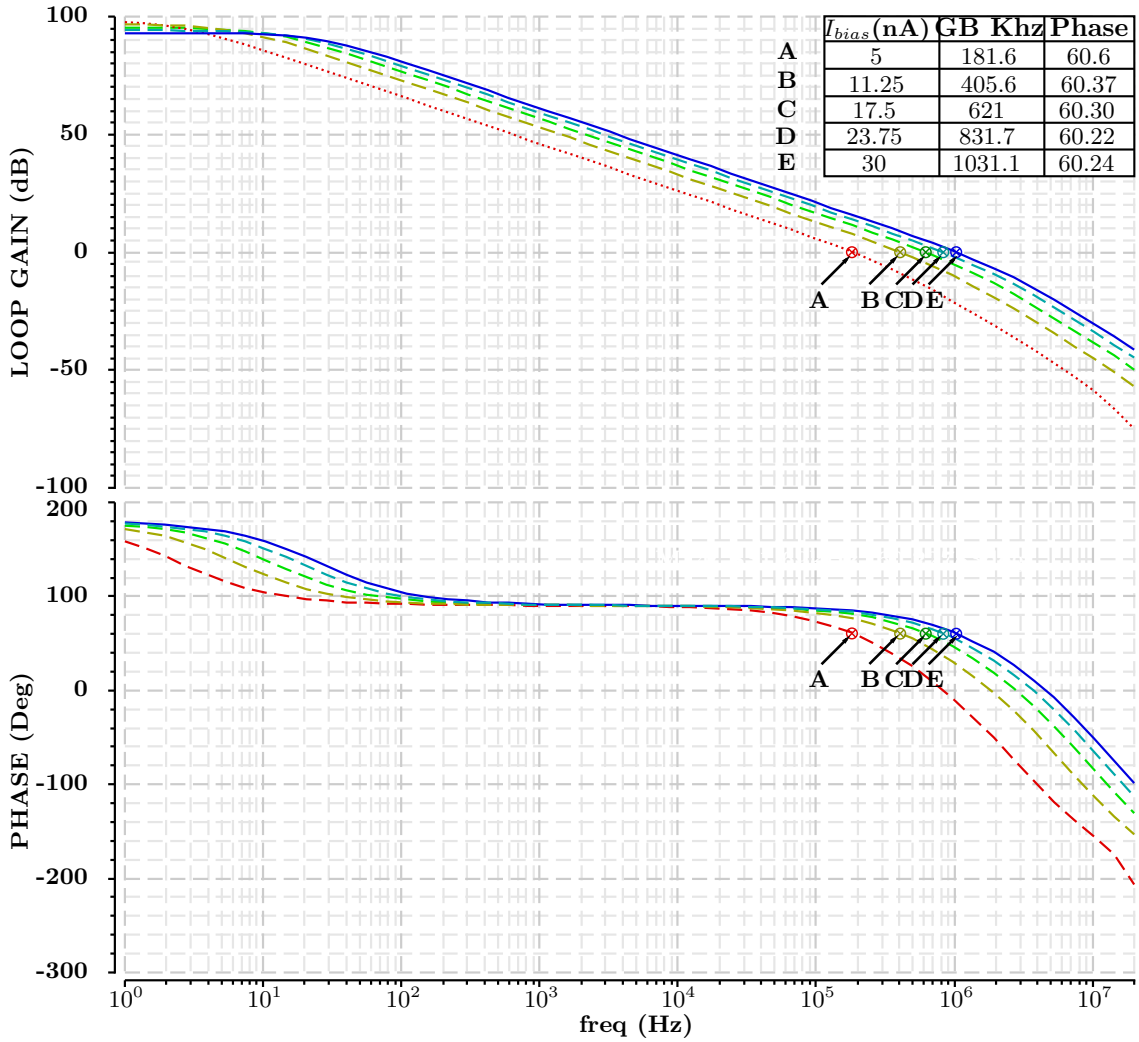


Figure 3.5: Two-stage operational amplifier frequency response.

3.3 Voltage Comparator with Hysteresis

We implemented the same voltage comparator with hysteresis that was described in [16]. It is shown in Figure 3.6. The transistor sizes are as follows: for M1a and M1b,

$L = 1$ μm and $W = 270$ nm; for M2, $L = 1$ μm and $W = 10.5$ μm; for M3a and M3b, $L = 1$ μm and $W = 1.77$ μm; for M4, $L = 1$ μm and $W = 3$ μm; for M_{hyst}, $L = W = 1$ μm; for Ms1 and Ms2, $L = 180$ nm and $W = 220$ nm. This design assumes that the offset voltage is within a desired range. In our case, the maximum offset voltage is expected to be around 20 mV. Voltage hysteresis is implemented by means of the M_{hyst} transistor. The Ms1 transistor acts as a switch, which helps power consumption to be reduced when the non-inverting input voltage is below the inverting input voltage. The Ms2 transistor, which also acts as a switch, enables hysteresis only when the non-inverting input voltage is greater than the inverting input voltage.

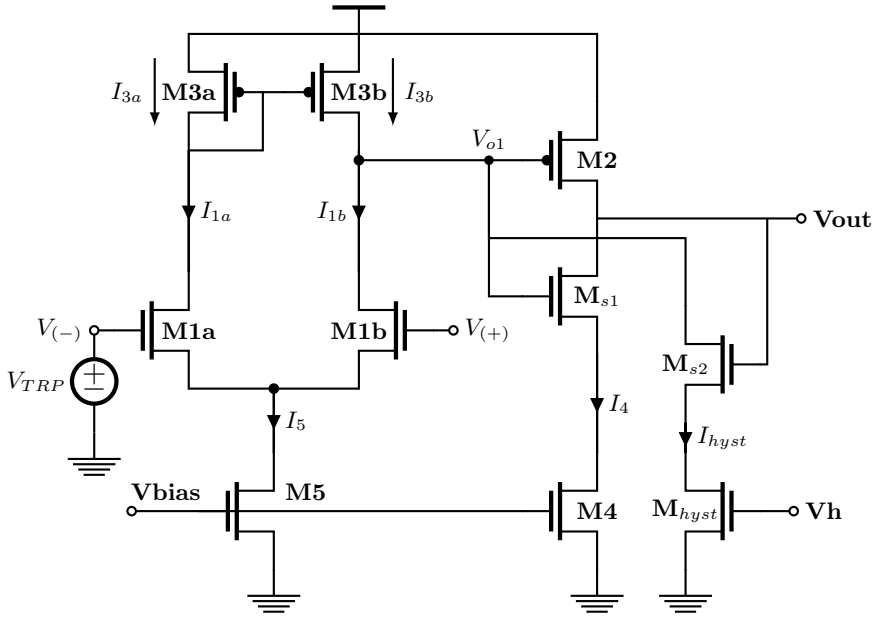


Figure 3.6: Voltage comparator [18].

The voltage comparator input voltage offset is due to random mismatch between M1a and M1b. Mismatch is a time-independent difference (error) that exists between the designed circuit and the fabricated circuit [29], which, among many other things, causes M1a and M1b not to be identical. *Systematic* mismatch is created during fabrication, because of limited accuracy in lithographic and chemical processes. By using appropriate layout techniques, we can reduce systematic mismatch effects. *Random* mismatch originates in local variations that occur in the fabrication process, and its effects dominate over the systematic mismatch effects for small distances between devices such as M1a and M1b. Random mismatch may be modelled using a particular error model that is known as the Pelgrom model [36]. The random offset voltage between the differential pair (M1a and M1b) inputs may be estimated according to Equation (3.20):

$$V_{offset,total} = \sqrt{V_{offset,pair}^2 + V_{offset,mirror}^2} \tag{3.20}$$

where

$$V_{offset,pair} = \sqrt{\frac{A_{VTO}^2}{WL} + \left(\frac{I_D}{g_m}\right)^2_{pair} \frac{A_\beta^2}{WL}} \tag{3.21a}$$

$$V_{offset,mirror} = \sqrt{\frac{A_\beta^2}{WL} + \left(\frac{g_m}{I_D}\right)^2_{mirror} \frac{A_{VTO}^2}{WL}}, \tag{3.21b}$$

The parameters A_β and A_{VTO} depend on the fabrication process, and W and L are the transistor width and length. According to Equations (3.21a) and (3.21b), the offset voltage may be adjusted by a careful selection of W and L.

We used the g_m/I_D methodology to design the voltage comparator so that its offset voltage is within the desired range (below 20 mV). Figure 3.7 shows the circuit used for verifying whether the voltage comparator offset was within the desired range or not. The voltage comparator offset estimation, based on a Monte Carlo simulation[1], is shown in Figure 3.8.

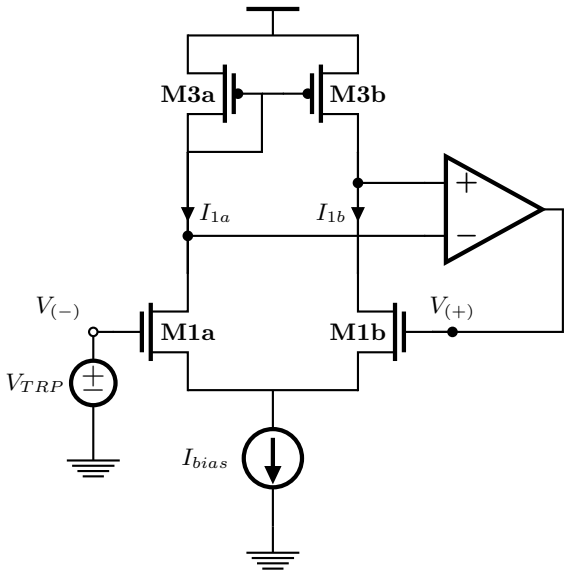


Figure 3.7: Circuit for verifying the voltage offset based on Monte Carlo simulation.

We next describe how Ms2 and M_{hyst} implement the hysteresis voltage We first define the hysteresis upper limit V_{TRP}^+, which is relevant when $V_{(+)} < V_{(-)}$. In this case, we have $I_{1a} > I_{1b}$, $I_{1a} < I_5$, and $I_{1b} > 0$, so that Ms2 is off and $I_{hyst} = 0$. When voltage V_{o1} reaches the minimum value for which M1b and M3b remain in the saturation region ($|V_{ds}| > 3\phi_t$ for M_{1b} and M_{3b}), I_{1a} starts to decrease, and it decreases until $I_{1a} = I_{1b} = I_{3a} = I_{3b}$. We state that $I_{1a} = I_{1b}$ corresponds to $V_{TRP}^+ = 0$, an so V_{TRP}^+ is defined as $V_{TRP}^+ = V_{gs1a} - V_{gs1b}$. After that, we define the

[1]This Monte Carlo simulation was accomplished with the objective of validating the voltage offset specification on the voltage comparator design. This result indicates that the voltage offset stays within the desired range (below 20 mV)

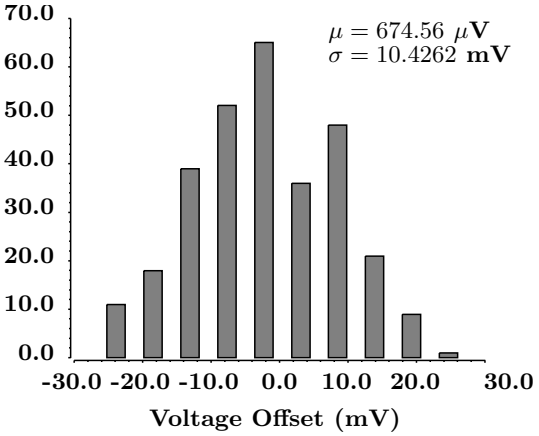


Figure 3.8: Voltage comparator offset estimation based on Monte Carlo simulation.

hysteresis lower limit V_{TRP}^{-}, which is relevant when $V_{(+)} > V_{(-)}$. In this case, we have $I_{1a} < I_{1b}$, $I_{1b} < I_5$, $V_{out} = V_{DD}$, and $I_{hyst} > 0$. Applying Kirchoff current law to the V_{o1} node, we have $I_{3b} = I_{1b} - I_{hyst}$. If V_{o1} is low enough for M1b and M3b to operate in the saturation region, then:

$$I_5 = I_{1a} + I_{1b} \qquad \rightarrow I_5 = 2I_{1a} - I_{hyst} \qquad \rightarrow I_{1b} = \frac{I_5 + I_{hyst}}{2}, \tag{3.22}$$

$$I_{1a} = I_{3a} = I_{3b} \qquad \rightarrow I_{1a} = \frac{I_5 + I_{hyst}}{2} - I_{hyst} \qquad \rightarrow I_{1a} = \frac{I_5 - I_{hyst}}{2}. \tag{3.23}$$

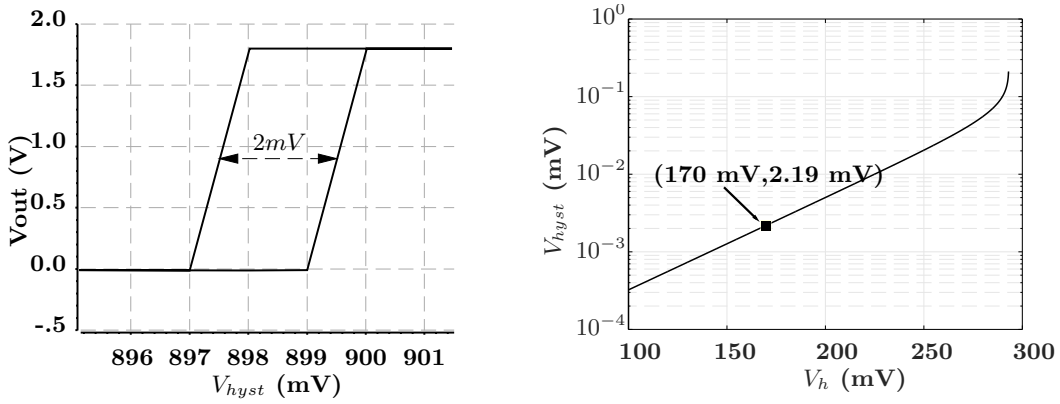
Using the EKV model in the subthreshold regime and defining $V_{TRP}^{-} = V_{gs1a} - V_{gs1b}$, where V_{gs1a} and V_{gs1a} are defined by the two equations provided below, we have $V_{TRP}^{-} = n\phi_t \log{(I_5 - I_{hyst})/(I_5 + I_{hyst})}$.

$$V_{gs1a} = n\phi_t \log \frac{I_{1a}}{I_5} + V_{tn} = n\phi_t \log \frac{I_5 - I_{hyst}}{2I_5} + V_{tn}, \tag{3.24}$$

$$V_{gs1b} = n\phi_t \log \frac{I_{1b}}{I_5} + V_{tn} = n\phi_t \log \frac{I_5 + I_{hyst}}{2I_5} + V_{tn}. \tag{3.25}$$

Finally, the comparator hysteresis voltage is found according to $V_{hyst} = V_{TRP}^{+} - V_{TRP}^{-}$. Expressing the currents according to the EKV model in weak inversion yields Equation (3.26). According to Equation (3.26), V_{hyst} does not depend on the transistor threshold voltage . The comparator design, with an emphasis on hysteresis verification, was validated by electrical simulations, as shown in Figure 3.9.

$$V_{hyst} = n\phi_t \log \left[\frac{\exp \frac{V_{bias} - V_h}{n\phi_t} + 1}{\exp \frac{V_{bias} - V_h}{n\phi_t} - 1} \right] \tag{3.26}$$



(a) (b)

Figure 3.9: Voltage comparator simulation results: (a) Spectre simulation indicating hysteresis, using $V_h = 170$ mV; (b) hysteresis voltage V_{hyst}, plotted as a function of V_h.

3.4 AER Circuit

As mentioned in Sec. 2.5, AER allows off-chip transmission of spikes generated within the pixel array. For that purpose, pixel requests must be multiplexed somehow. The AER system presented in this work uses time-domain multiplexing. Figure 3.10 shows an AER system that was designed for a 4×4 pixel array. The Y-AER system, which is shown in Figure 3.10b, enables row requests (RREQ) one at a time. Pixels in the acknowledged row request access to the communications channel (which would be a serial bus, in the context of our work) via X-AER, which is shown in Figure 3.10a. All such pixels will have their events (spikes) transmitted through the communications channel, and their row will only be released after all events were successfully transmitted. Ultimately, the communication between pixels and an off-chip system is thus managed by the X-AER system. The X-AER system has an additional arbiter that is referred to as X-ARBITER. By generating a PARITY signal, the X-ARBITER whether the event that was detected by the pixel is an *on* event or an *off* event. In the DVS and ADMDVS pixels, PARITY is at a high voltage level to indicate an *on* event, and it is at a low voltage level otherwise. In the ATIS pixel, PARITY indicates which threshold has been crossed by the curve corresponding to integrated charge: PARITY at a high voltage level indicates that $V_{req,Hx}$ has been crossed, and PARITY at a low voltage level indicates that $V_{req,Lx}$ has been crossed. The X-ARBITER is only used at the second level of the arbiter tree.

The encoders are implemented using logical multiplexers. The fact that the data are valid (i.e. ready for transmission) in indicated by the En_Read_Pixel signal, which is sent, after some time delay, by the X-AER system to an off-chip receiver. The time delay must be adjusted to ensure that the data are valid. Recall that in

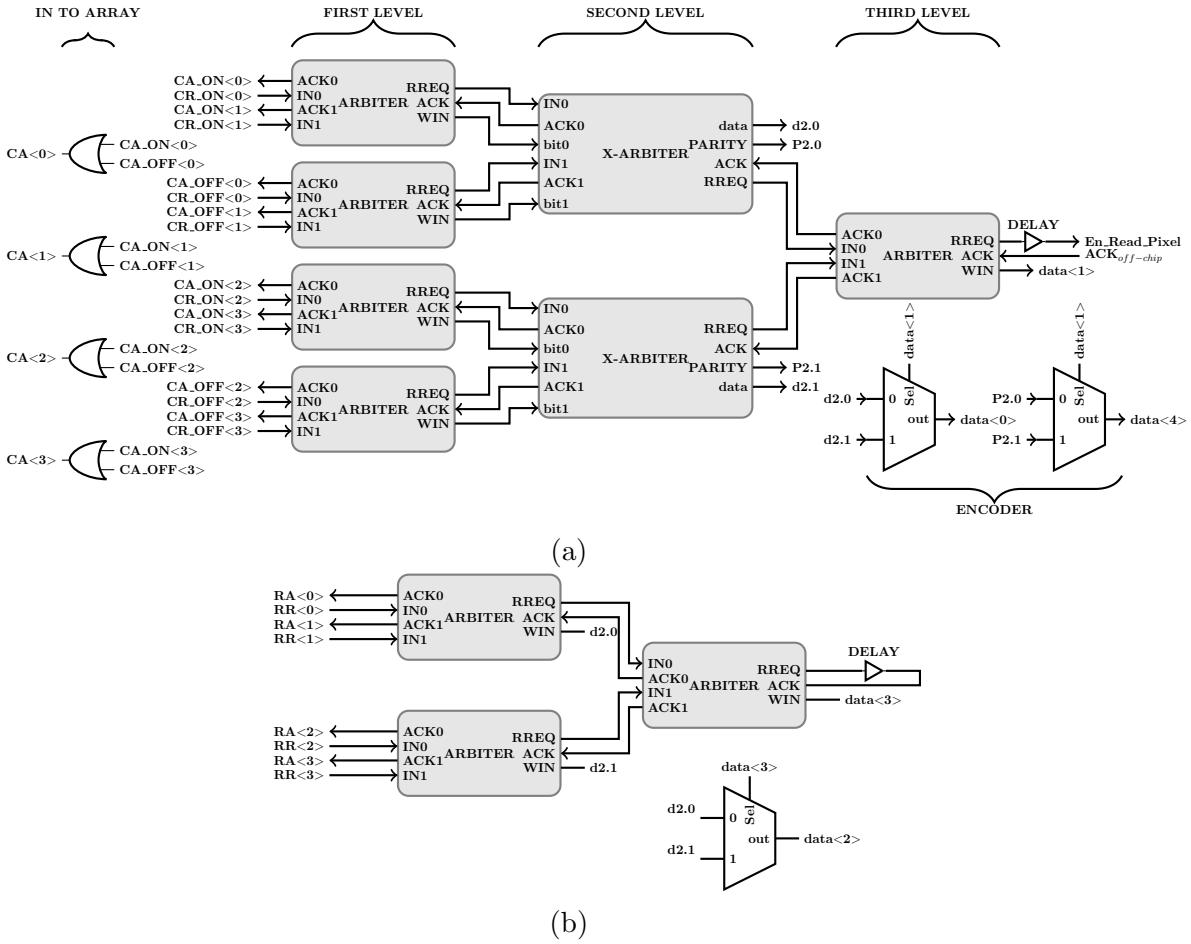


Figure 3.10: AER system for 4×4 pixel array: (a) X-AER system (for enabling column requests); (b) Y-AER system (for enabling row requests).

the quasi-delay-insensitive approach we have no control over the time delay of each logical element, so we must assume arbitrary delays in gates and wires. By using the quasi-delay-insensitive approach, we obtain low-complexity asynchronous logic designs, at the expense of an increase in power consumption and circuit area.

Figure 3.11 shows the circuits that compose each part of the X-AER and Y-AER systems. The arbiters are equal to the ones described in [24]. Each arbiter consists of a mutual exclusion circuit, AND gates, and two Muller C-elements. The mutual exclusion element can be implemented using a pair of cross-coupled NAND gates and a metastability filter, as shown in Figure 3.11a. The metastability filter is needed for solving potential conflicts between the IN0 and IN1 inputs, so that one of the input signals gets selected as a winner. Conflicting inputs lead the NAND gate outputs to a metastable state, in which one of the output voltage signals is halfway between ground and V_{DD}. The mutual exclusion element thus ensures that only one of the inputs will be attended to. Because of the AND gates and the Muller C-elements, the handshaking is mutually exclusive. When attention is being paid to one of the input signals, the other signal is blocked until the processing of the winning signal

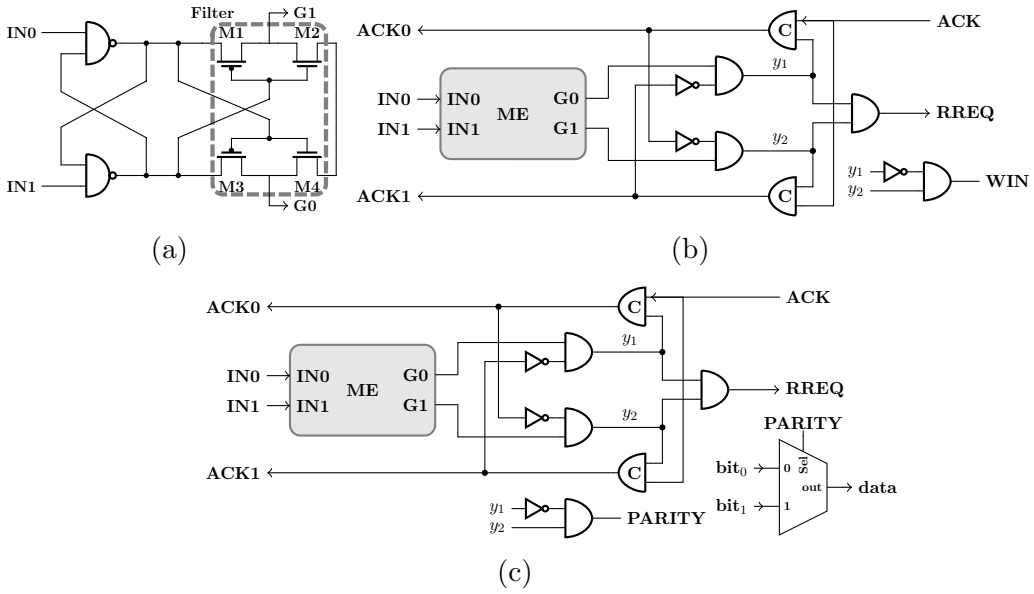


Figure 3.11: Circuits that compose parts of the AER systems: (a) mutual exclusion (ME) circuit, (b) basic arbiter, and (c) X-ARBITER.

is complete.

3.5 Exposure Measurement Logic

We use the same exposure measurement logic that was proposed in [16]. By activating each switch in Figure 2.6 depending on the integration cycle, explained in subsection 2.2.2, the exposure measurement logic enables True CDS. In the initial integration cycle, the logical state is '0', which sets the comparator reference voltage to V_{high}. When the integrated photocurrent crosses the V_{high} threshold, the exposure measurement logic communicates with both the X and Y-AER systems. It lets the Y-AER system know that one pixel has started its brightness measurement cycle. Immediately after the Y-AER system acknowledges the exposure measurement logic request, the exposure measurement logic request access to the data bus through X-AER. At that point, the exposure measurement logic state changes to '1', which means that the exposure measurement logic is handling the main charge integration phase (i.e. the brightness measurement cycle). The comparator reference voltage is changed to V_{low}. The exposure measurement logic state defines which switch is activated. State '0' means reference voltage equal to V_{high}, and state '1' means reference voltage equal to V_{low}. Whenever the temporal change detector generates an event, the exposure measurement logic must return to state '0', not caring whether the brightness measurement cycle has already started or not.

During the design, we observed an exposure measurement logic circuit problem, caused by the comparator time delay, which was not analyzed in [16]. Long simu-

lations showed that the exposure measurement logic would not return to state '0' immediately after the temporal change detector had generated an event. If the pixel is in its brightness measurement cycle (state '1'), and the temporal change detector generates an event, then the exposure measurement logic state must immediately return to '0'. However, we observed that the reset pulse width is typically not large enough to wait until the voltage comparator output goes to the low voltage level, which is shown in Figure 3.12a. In the original exposure measurement circuit design, both the reset signal and voltage comparator output signal must be low, for the state to return to '0'. Possible solutions would be: i) increasing voltage comparator speed by increasing its bias current, and ii) adjusting, with an off-chip control signal, the reset signal pulse width. The first solution is not recommended in vision sensors, because it increases power consumption. The second solution requires an external control signal.

We propose a simple on-chip solution that uses a few additional devices, as shown in Figure 3.13. The circuit shown in Figure 3.13a works according to the finite state machine that is shown in Figure 3.13b. The main idea is that, for the purpose of changing the exposure measurement logic state to '0', the reset signal must dominate over the voltage comparator output. After the reset signal was generated, the exposure measurement logic state returns to '0' through the Out signal. The exposure measurement logic state behavior based on the proposed solution is shown in Figure 3.12b. It is clear that the invalid state condition is suppressed. Besides, this solution does not require larger comparator bias currents or an off-chip control signal.

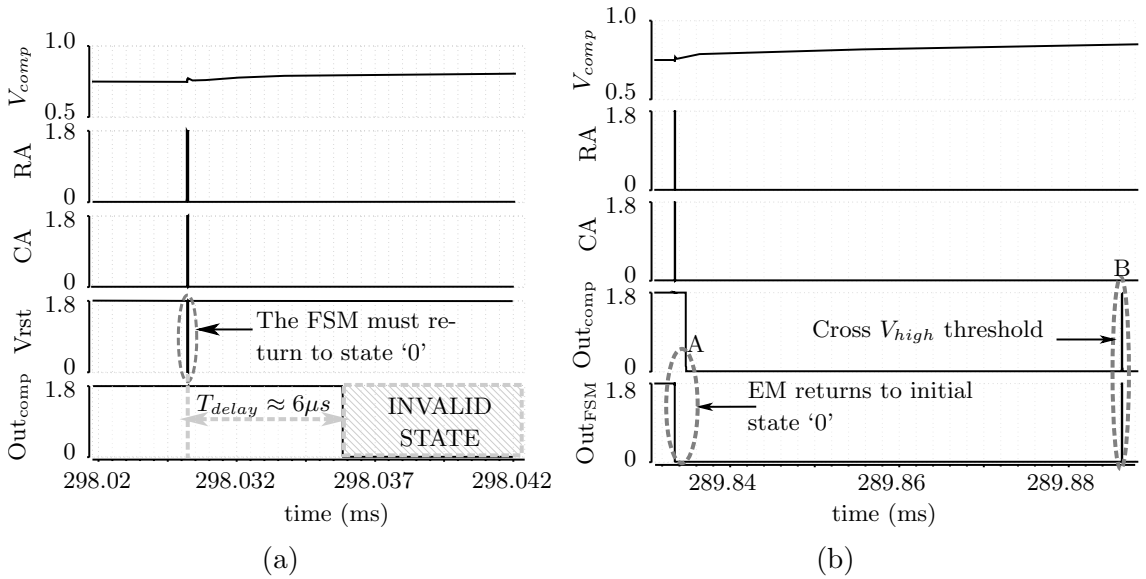
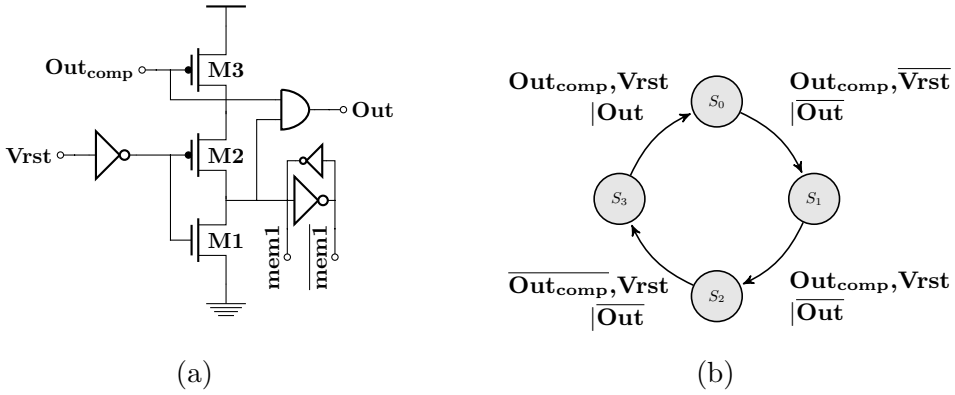


Figure 3.12: Timing diagram for a single *off* event detected at DVS pixel: (a) exposure measurement logic in invalid state because of voltage comparator delay, and (b) correct response (Out_{FSM}) obtained with the proposed solution.



(a) (b)

Figure 3.13: Circuit for solving the exposure measurement logic invalid state problem: (a) finite state machine implementation, and (b) finite state machine implementation state diagram.

3.6 Delay Element Circuit

We designed a CMOS delay element for the ADMDVS pixel, according to the topology presented in [37]. The circuit is shown in Figure 3.14. Transistor sizes are as follows: for M1a and M1b, $L = 180$ nm and $W = 220$ nm; for M2a and M2b, $L = 2$ μm and $W = 1$ μm, for M3a and M3b, $L = 300$ nm and $W = 500$ nm; for M4a and M4b, $L = 2$ μm and $W = 500$ nm; for M5a and M5b, $L = 2$ μm and $W = 1$ μm; and for Mb1 and Mb2, $L = W = 1$ μm. The delay is adjusted through the bias current I_{bias}, and this delay element is recommended for low-power systems. We tried to use the delay element provided in the CORELIB from AMS. Unfortunately, the total delay was too short. Also, the pixel was larger and power consumption was higher than desirable. The delay is given by Equation (3.27) [38], where V_{tp} and V_{tn} are p-channel and n-channel transistor threshold voltages, C1 is the node capacitance at M4a gate, C2 is the node capacitance at the output, κ is $1/n$ (n is the slope factor) and δ_t is the time for the regeneration at the CMOS thyristor. According to Equation (3.27), time delay t_d is inversely proportional to the bias current I_{bias}, as long as δ_t is relatively small. We used repeated simulations to manually find adequate transistor sizes.

$$t_d = \frac{C_1 V_{tp}}{I_{bias}} + \sqrt[3]{\frac{6C_2C_1^2}{\kappa I_{bias}^2} V_{tn}} + \delta_t \tag{3.27}$$

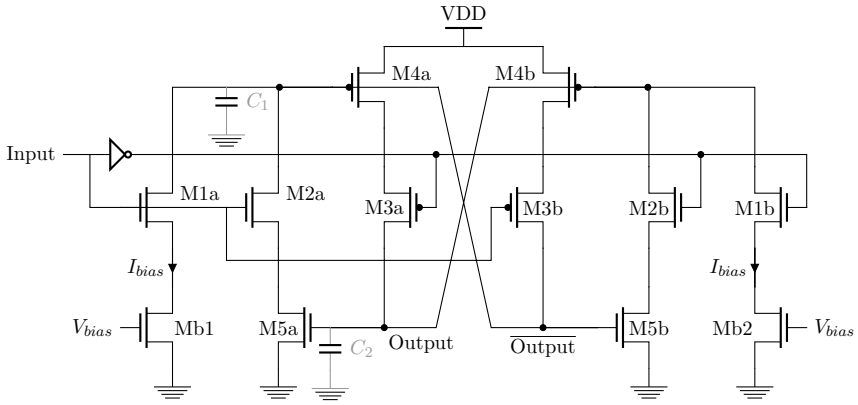


Figure 3.14: Delay element circuit.

3.7 Summary of Designed Pixels

Transistor count for each pixel is summarized in Table 3.2. The area figures correspond to active elements only. The area is estimated by adding transistor channel areas WL, not taking into account the rules for minimum spacing between active elements. The area required by capacitors and wiring was not taken into account.

Table 3.2: Active components within each pixel, and estimated area figures.

Pixel Type	**Part**	**# Transistors**	**Area (μm)**
DVS	Analog	35	73.3
	Digital	36	4.9
ATIS	Analog	38	73.5
	Digital	144	44.2
ADMDVS	Analog	68	128.1
	Digital	105	16.3

Chapter 4

Simulation Results

This chapter presents results obtained from Spectre electrical simulations of the pixels that were studied in this work (basic DVS, ATIS, and ADMDVS). The goal was to verify the correct behavior of each pixel. Scripts that were used for automatic control of the simulations are presented in Appendix A. Scripts for DVS, ATIS and ADMDVS simulations themselves, as well as algorithms for reading simulation output data and making plots, are presented in Appendix B.

As DVS pixels respond to temporal contrast change, they must be simulated with transient photocurrents. Sinusoidal current sources were used in this work, in order to provide photocurrent stimulus for a pixel. So, to verify a single instance of each pixel, we modelled the photodiode using a sinusoidal current source $I_{pd} = 50 \times 10^{-12} \sin(2\pi \times 10t) + 100 \times 10^{-12}$ ampères. The V_{diff} and global reset pulse width parameters were set to 150 mV and 1 ms. The global reset signal aims at taking the DVS pixel back to its operation starting point at any given time instant. The global reset pulse width was different from 1 ms only for ADMDVS pixel simulations: in that case, it was larger than 1 ms. The ADMDVS pixel does not have any reset switch, so the operational amplifiers output must reach the voltage reference, which may take a long time (e.g. 30 seconds).

After having verified a single instance of each pixel, we investigated the behavior of a set of pixels composing a small sensor array, such as an 8 × 8 or a 4 × 4 pixel array. We considered three different cameras: DVS 8 × 8, ATIS 4 × 4, and ADMDVS 4 × 4. To verify the designs, we used arbitrary space-domain spiral-shaped signals (Figures 4.9, 4.12, and 4.13). In each case, the electrical simulation results obtained from Spectre were compared to numerical predictions that were obtained from each pixel model presented in Chapter 2.

We want to emphasize that the responses of the different types of pixels that have been studied in this work are not related in number of frames, however they are related in number of events. For example, in [22] for a specific input signal, the DVS camera detected up to 650,000 events per second. This is not equal to the

amount of frames because this type of sensors are frame-free and they only respond to temporal contrast changes in its field of view.

4.1 DVS Pixel Simulation

Figure 4.1 shows a transient (i.e. time-domain) simulation for a single DVS pixel instance. The I_{pd} input (photodiode current) is shown in the plot at the top. The temporal contrast change that is detected from the photodiode generates *on* or *off* events. In the fifth and fourth plots (counting from top to bottom), we can see that: i) *off* events (CROFF) were generated, for example, between 250 ms and 275 ms, because then the photodiode current derivative is sufficiently negative, and ii) similarly, *on* events (CRON) were generated between 275 ms and 325 ms, because the photodiode current derivative is sufficiently positive along that time interval. The $V_{in,comp}$ signal, in the second plot, is the input for the voltage comparators. For a single pixel instance simulation, the AER systems were not taken into account. They were replaced by simple inverters, that act as logical buffers with a low capacitive input load. The row and column acknowlegment signals, which are shown in the last two plots (RA and CA), are the same for *on* and *off* events. They are also equal to the row requests (RREQ) that are shown in the third plot. In the general pixel array case, which involves more than a single pixel, a RA signal is shared by all pixels in that row, and a CA signal is shared by all pixels in that column.

Figure 4.2 presents some details of an *off* event that occured around 263 ms in Figure 4.1. In a DVS pixel, an *off* event is generated when $V_{in,comp}$ is higher than the V_{doff} threshold. In that case, the DVS pixel generates a row request (RREQ) through its logical circuit. A row control unit, which is just an inverter pair in this single-pixel case, receives the RREQ signal and acknowledges the receipt by sending an RA signal back to the pixel, after some delay that corresponds to the row control unit signal processing. When the pixel receives the RA signal, it generates a CROFF signal. The column control unit, which is also just an inverter pair in this single-pixel case, receives the CROFF signal and acknowledges the receipt by sending a CA signal back to the pixel, which also happens after some delay corresponding to the column control unit signal processing. Immediately after the RA and CA signals were received by the pixel, it is reset. The reset takes the pixel to its initial operation point, at $V_{ref} = 0.9$ V. According to the simulation, the initial operating point is at 940 mV, which is close to expected.

We also simulated the DVS pixel response numerically (in a numerical computation environment), using the DVS model described in Section 2.1.1, and using the same sinusoidal input I_{pd} that was shown in Figure 4.1. In the numerical simulation, the V_{doff} and V_{don} parameters were set to 1.05 V and 750 mV, respectively.

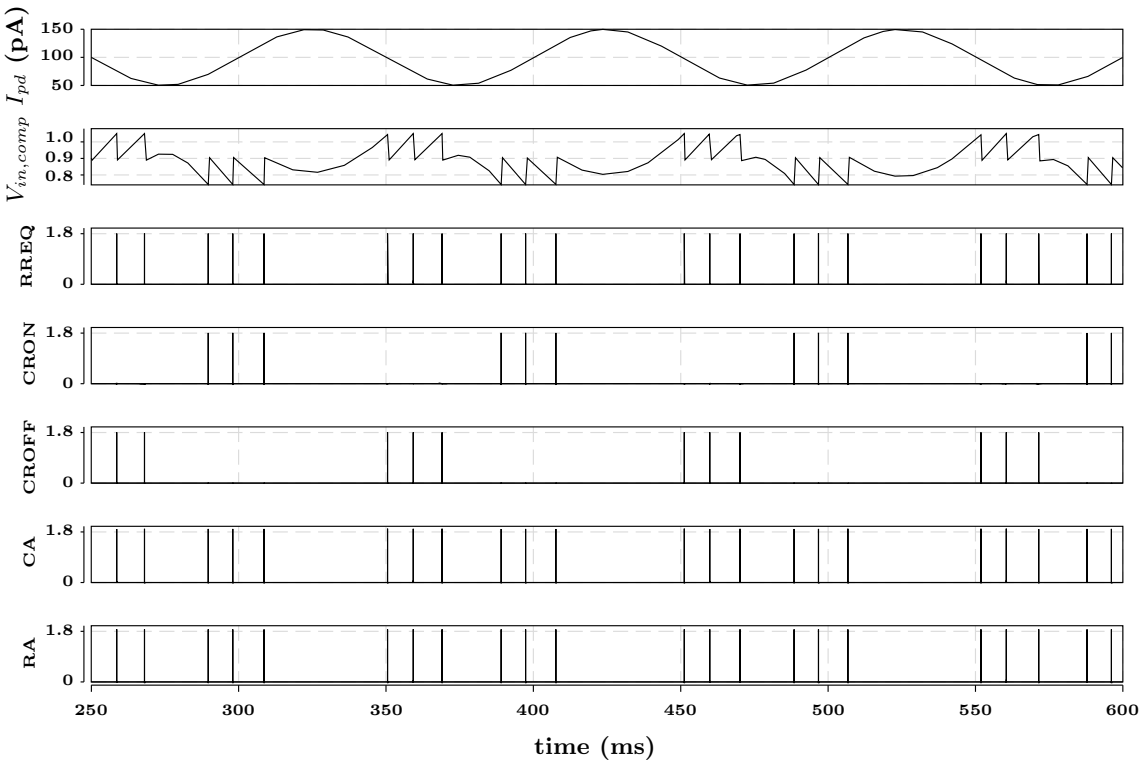


Figure 4.1: Single DVS pixel transient simulation (250 ms to 600 ms) with a sinusoidal input I_{pd} (first plot, at the top). The second plot shows the voltage comparator output V_{comp}. The third plot shows data bus pixel request (RREQ). The fourth and fifth plots show *on* and *off* events (CRON and CROFF). The sixth and seventh plots show column and row acknowledgment (CA and RA) signals.

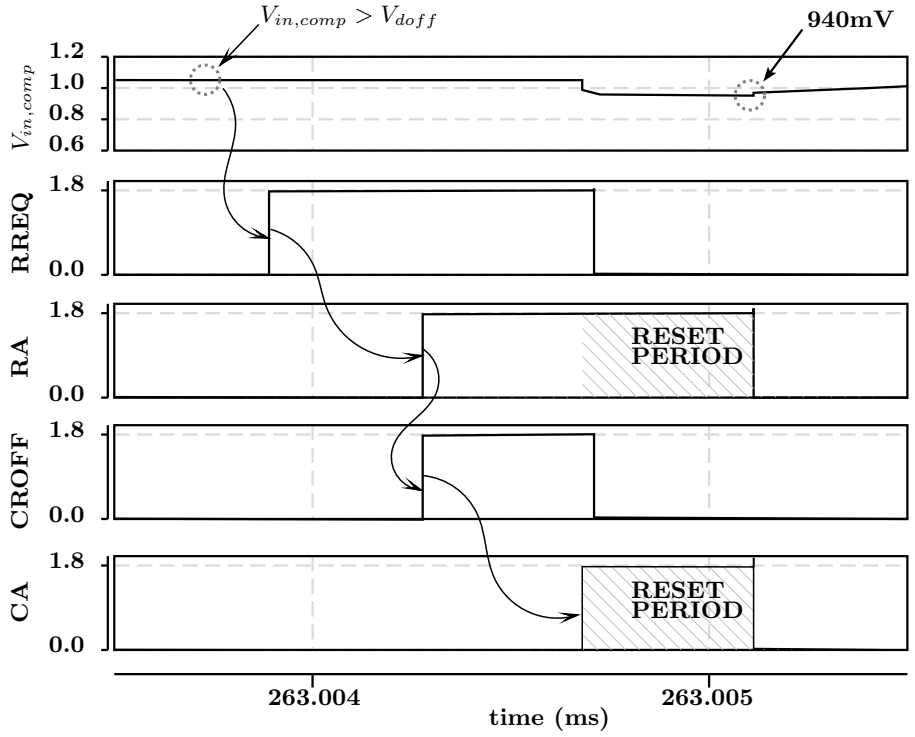


Figure 4.2: Timing diagram corresponding to a communication cycle that is triggered by a detected *off* event in Figure 4.1.

The numerical model does not take into account the delay that is due to the logical circuits (either in the pixel or in the row/column control units). The result is shown in Figure 4.3. A comparison between the $V_{in,comp}$ plots in Figures 4.1 and 4.3 indicates that, in electrical simulations, the DVS pixel behavior corresponds to what was expected from the numerical simulations.

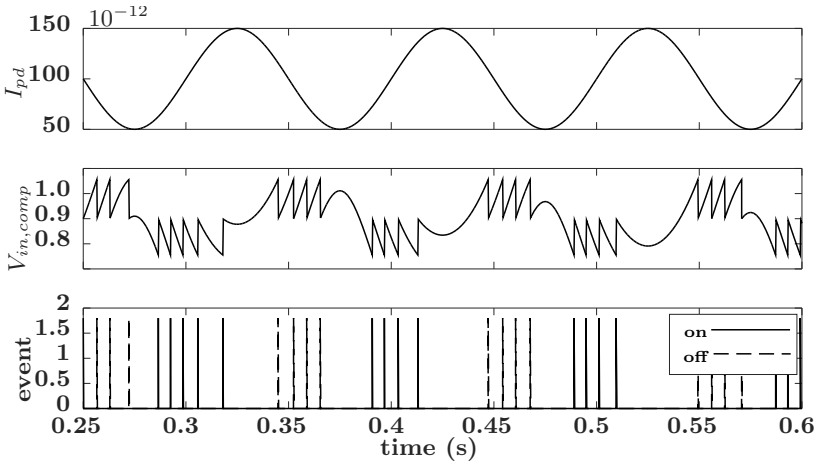


Figure 4.3: Single pixel DVS simulation using a numerical model from Section 2.1.1, and the same sinusoidal input shown in Figure 4.1. The second plot shows the predicted voltage comparator output, and the third plot shows predicted events. Communications with row and column control units are not taken into account.

Table 4.1 presents the number of spikes observed, in each period of the input signal, for three different voltage comparator threshold values V_{diff}. The spike numbers observed in the electrical simulation correspond reasonably well to the numerically predicted spike numbers. The numerical DVS pixel model does not take into account AER circuitry and its delay. In a real camera experimental characterization stage, if V_{diff} is adjusted to the lowest possible value (in order to yield high sensitivity), the pixel generates many more events, but a large part of those events corresponds to temporal noise [15], [10]. On the other hand, where there is no temporal contrast change, the $V_{in,comp}$ signal will decrease slowly, which eventually generates background *on* events, as mentioned in Section 2.3. In electrical simulations, the Spectre parameter *gmin* was adjusted to 1 aS (i.e. 10^{-18} siemens), taking into account the low bias currents of the analog circuits, in order to avoid background *on* events.

Table 4.1: Quantity of spikes with different setting of comparator threshold per Channel and period signal

V_{diff}	Model		Simulation	
	# *on* events	# *off* events	# *on* events	# *off* events
100 mV	7	6	7	4
150 mV	4	4	3	3
200 mV	3	3	2	2

4.2 ATIS Pixel Simulation

Figure 4.4 shows transient simulation results for a single ATIS pixel instance. For visualization simplicity, we show only one period of the input signal. The input signal, which appears at the top plot, is the same that was used in Section 4.1.temporal change detector signals such as RREQ, CRON and CROFF are not shown in Figure 4.4, because their behavior is equal to what was presented in Figure 4.1. Figure 4.5 presents details of the communication cycle that is triggered by an *off* event that occurs around 349 ms in Figure 4.4. When the temporal change detector (equal to the DVS pixel) detects an event, the exposure measurement circuit starts the brightness measurement cycle, what is to say, it starts the photodiode current integration cycle. Initially, the photodiode cathode voltage is set to V_{DD} by the reset transistor, and the voltage comparator reference input is connected to V_{high} (sixth plot in Figure 4.5). As the photodiode current is progressively integrated, V_{int} decreases. When V_{int} becomes less than V_{high}, the voltage comparator output changes to logical '1'. When the comparator output is high, the exposure measurement logic is activated, which means that the exposure measurement logic starts a communication cycle with the row control unit. To start the communication cycle, the exposure measurement logic sends a $V_{req,By}$ signal to the row control unit. The row control unit acknowledges this request with a $V_{ack,By}$ signal. Immediately after receiving the $V_{ack,By}$ signal, the exposure measurement logic generates a $V_{req,Hx}$ request signal, which indicates that the pixel has started the brightness encoding cycle. Because the first threshold to be crossed is always V_{high}, the pixel always generates the $V_{req,Hx}$ signal before the $V_{req,Lx}$ signal. Immediately after the pixel receives the $V_{ack,By}$ signal, (see $V_{ack,x}$ in Figure 4.4), in response to $V_{req,Hx}$, the voltage comparator reference input changes to V_{low}. When V_{int} crosses the V_{low} threshold, as similar signaling sequence ($V_{req,By}$, then $V_{ack,By}$) leads to the $V_{req,Lx}$ signal, which indicates that the brightness encoding process is finished. The time difference (or, equivalently, the pulse width) between $V_{req,Hx}$ and $V_{req,Lx}$ defines the brightness measured by the ATIS pixel.

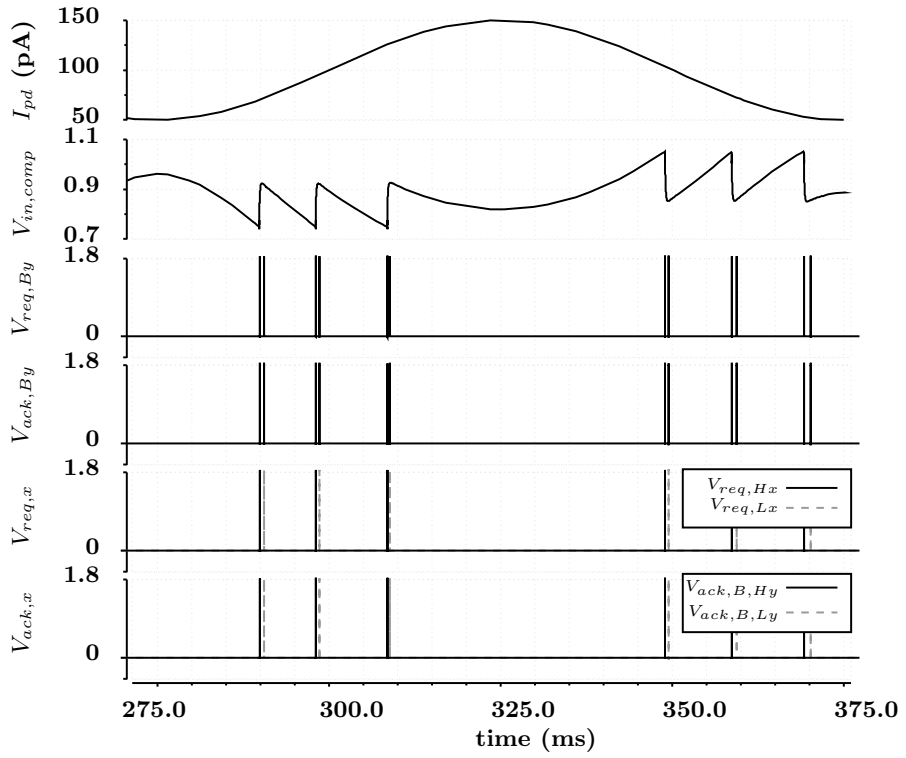


Figure 4.4: Single ATIS pixel transient simulation (275 ms to 375 ms) with a sinusoidal input I_{pd} equal to the one used in Figure 4.1. Relevant exposure measurement signals, which are described in the text, are shown in the third, fourth, fifth, and sixth plots.

4.3 ADMDVS Pixel Simulation

Figure 4.6 shows transient simulation results for a single ADMDVS pixel instance. The signals in Figure 4.6 are similar to those in Figure 4.1 (basic DVS pixel). The only exception is the column acknowledgment signal CA, which is now valid at the low voltage level. Every time an *on* event or an *off* event occurs, the V_{comp} signal decreases or increases by a δ value, respectively, according to Section 2.3.1. The δ value is the same to V_{diff} value mentioned above. Figure 4.7 presents details of the communication cycle that is triggered by an *off* event that occurs around 30.63 s in Figure 4.6. When an *off* event occurs, the VOFF signal at the respective voltage comparator output (see Figure 2.11) goes high. The VOFF$_o$ in the last plot of Figure 4.7 is a latched version of VOFF. Both VOFF and VOFF$_o$ are reset when the ϕ_s signal in the seventh plot goes high. The VOFF$_o$ signal activates the ϕ_l signal that is shown in the sixth plot. The pulse width of ϕ_l is defined by the time delay $t_{dly,1}$, which is shown in the second plot of Figure 4.7. Immediately after $t_{dly,1}$, when the ϕ_l signal goes down, the pixel generates the communication access request RREQ, which goes to the row control unit. The row control unit receives the RREQ signal, and it sends back an acknowledgment signal RA to the pixel. The ϕ_l signal causes the C_f capacitance to be charged to V_{low}. Immediately after receiving the RA signal,

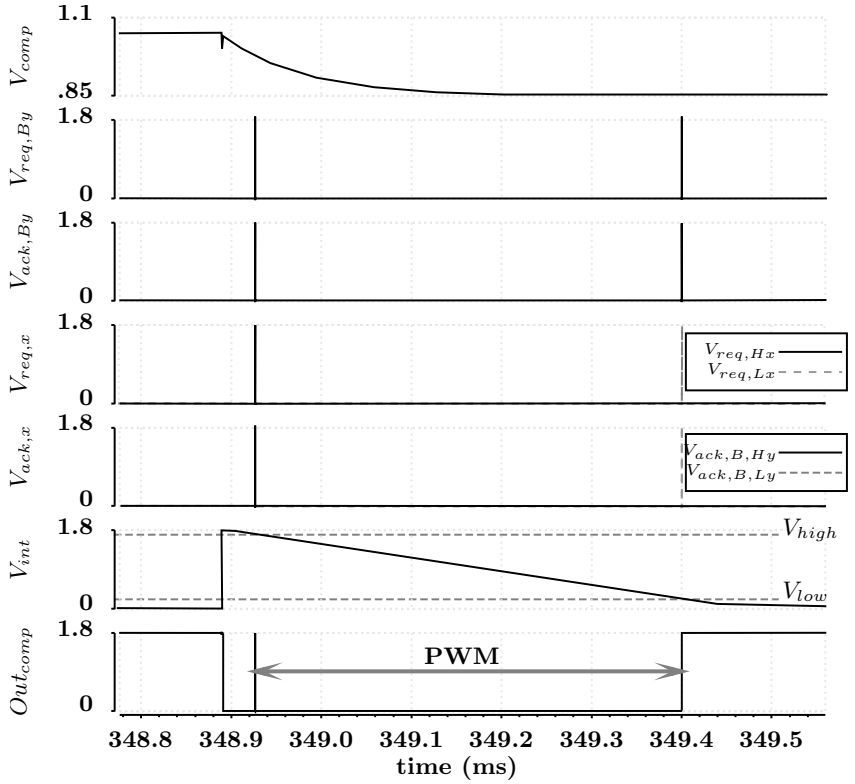


Figure 4.5: Timing diagram corresponding to a communication cycle that is triggered by a detected *off* event in Figure 4.4.

the pixel generates the CROFF signal. The column control unit receives the CROFF signal, and it sends back an acknowledgment signal CA (active at the low voltage level) to the pixel. Immediately after the pixel received the CA signal, the ϕ_l signal goes down, which enables the ϕ_s signal to rise after some time ($t_{dly,2}$ in the seventh plot of Figure 4.7. While ϕ_s is high, the ADM output is increased by V_{diff}. When ϕ_s goes down, the communication cycle is finished, and the pixel is ready to detect new events.

We also simulated the ADMDVS pixel response numerically, using the ADMDVS model described in Section 2.3.2, and using the same sinusoidal input I_{pd} that was shown in Figure 4.1. In the numerical simulation, the time delay was set to 2 μs. The numerical simulation result is shown in Figure 4.8. A visual comparison between Figures 4.6 and 4.8 indicates that the spike count in ADMDVS electrical simulation (see RREQ signal in Figure 4.6) is similar to the spike count that was predicted by ADMDVS numerical simulation (see Event signal in Figure 4.8). So, the ADMDVS behavior corresponds to what was expected from numerical simulations. Slight differences between the spike sequences are due to the fact that the CMOS circuit, which was used in Spectre electrical simulations, is more complex than the ADMDVS numerical model.

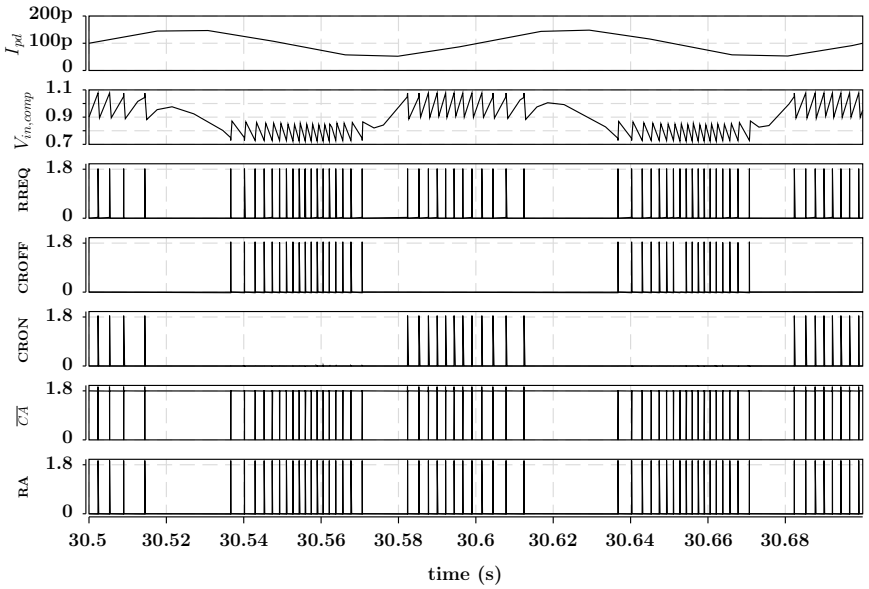


Figure 4.6: Single ADMDVS pixel transient simulation (30.5 s to 30.7 s) with a sinusoidal input I_{pd} equal to the one used in Figure 4.1.

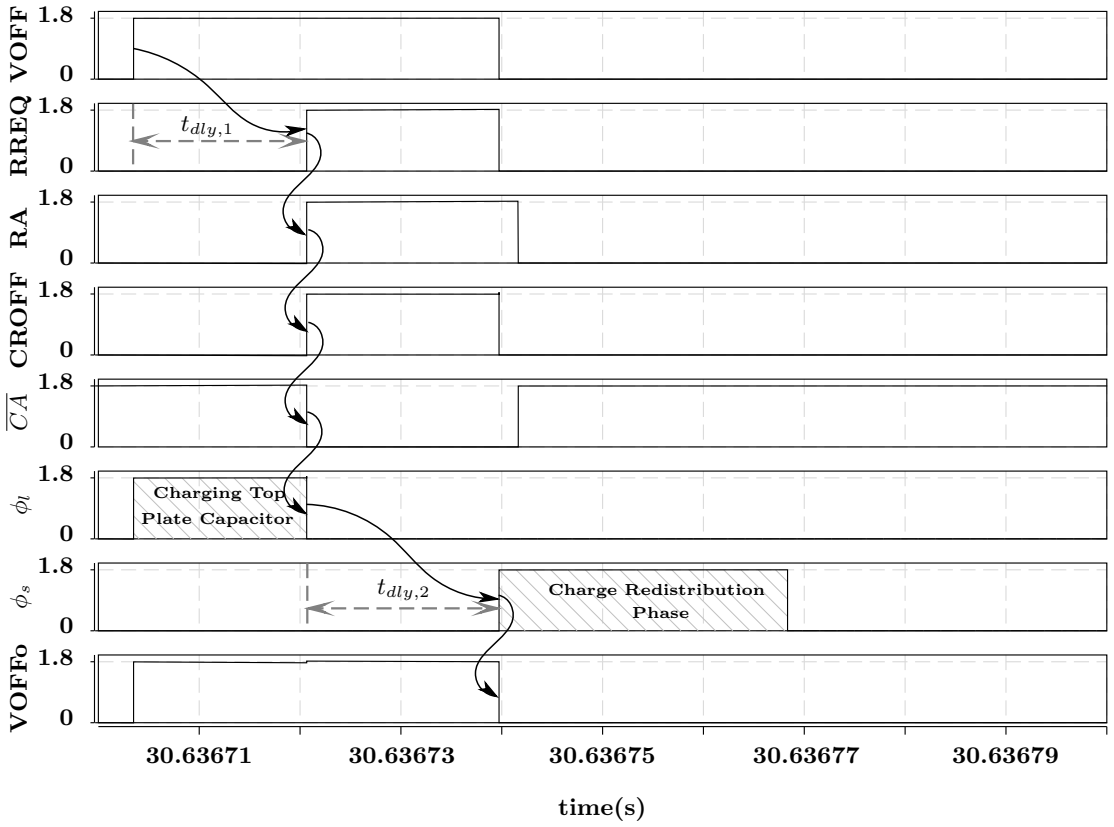


Figure 4.7: Timing diagram corresponding to a communication cycle that is triggered by a detected *off* event in Figure 4.6. Time delays $t_{dly,1}$ and $t_{dly,2}$ allow for capacitor charging C_f and charge redistribution among the capacitors in the ADMDVS pixel (see Figure 2.11).

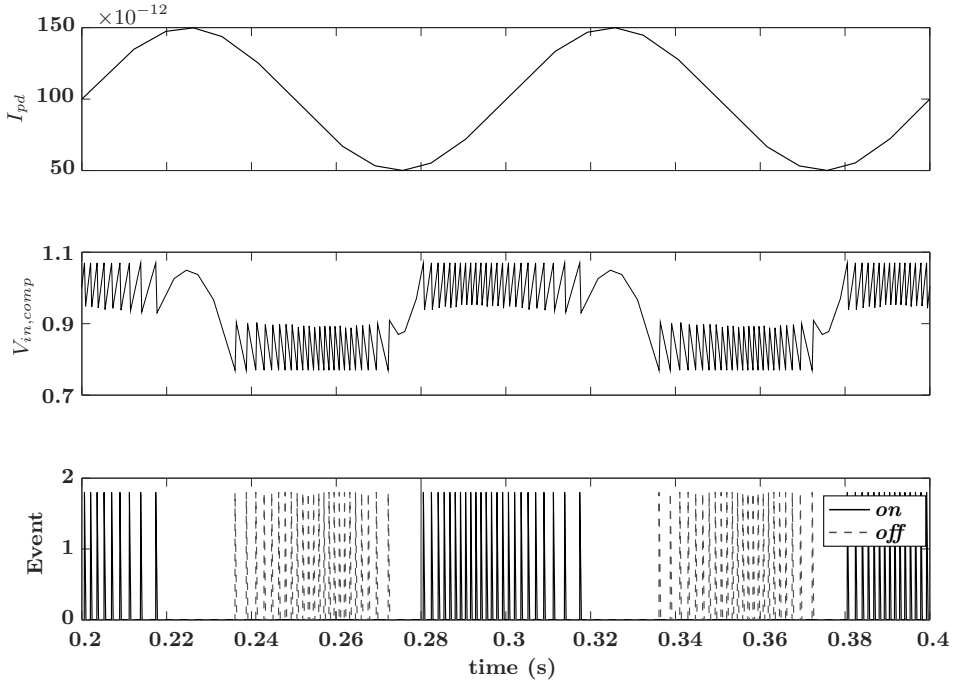


Figure 4.8: Single pixel ADMDVS simulation using a numerical model from Section 2.3.2, and the same sinusoidal input shown in Figure 4.1. The second plot shows the predicted voltage comparator output, and the third plot shows predicted events. Communication with row and column control units is not taken into account.

4.4 DVS 8 × 8 Pixel Array Simulation

We performed Spectre electrical simulations of an 8 × 8 DVS pixel array using, as an input signal, a 2-D spiral with angular frequency corresponding to 300 Hz. These simulations are also useful for tuning the pixel array to the AER circuits located at its periphery. Different algorithms, which are shown in Appendix B, are used for processing Spectre simulation results in order to measure performance figures, and in order to make plots.

Figure 4.9 compares the DVS electrical simulation results to the numerical predictions that were obtained with the DVS model that was described in Section 2.1.1. The gray scale in the figures indicates the time instant at which an event occurred. From black to white, early to late events are progressively represented. For visualization simplicity, we separate the pixel array response into its *on* and *off* components. A visual comparison shows that the electrical simulation results are very similar to the numerical predictions. To clarify further, we zoomed in at the pixel number 45, which is located on row 5, column 5. The events that occurred in this pixel are shown in greater detail in the plots of Figure 4.9. The number of events, both *on* and *off*, predicted by numerical model is greater than the number of events obtained in the electrical simulations, and the event timing is not exactly the same. Simultaneous events did not occur in this test. Collisions are not being treated by AER

systems that are specific for that purpose. So, should simultaneous events occur, they would all be treated at the same time. As the timing differences are not large, and considering that the circuit used for Spectre simulations is much more complex than its numerical model, we conclude that the pixel design is validated. Using a 2-D spiral as an input signal is useful for avoiding request collisions.

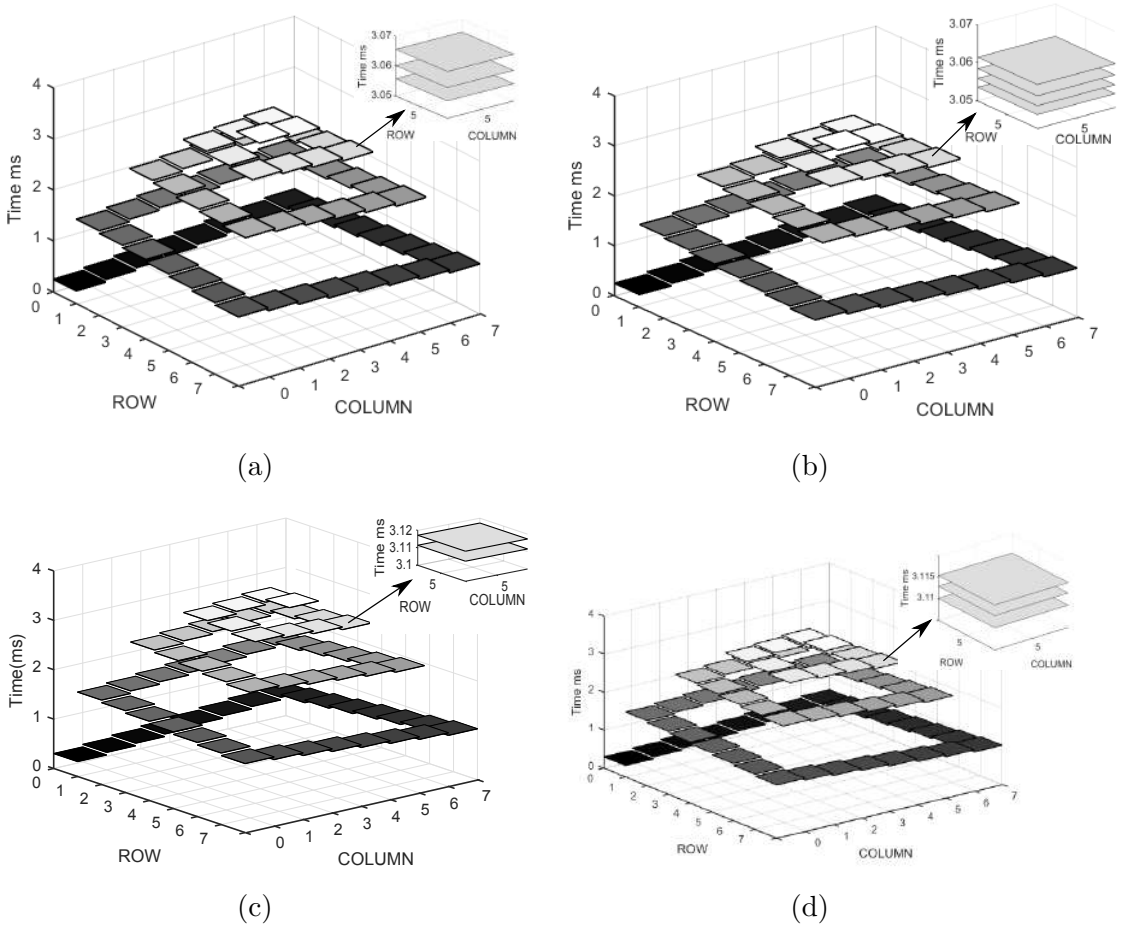


Figure 4.9: 8×8 DVS pixel array. Comparison between electrical simulation results and predictions based on the numerical model in Section 2.1.1: (a) and (c) *on* and *off* events generated by electrical simulation; (b) and (d) *on* and *off* events estimated from a numerical model. Event timing details are provided for the pixel on row 5 and column 5.

4.5 ATIS 4×4 Pixel Array Simulation

To validate the ATIS pixel array, we performed simulations with two different inputs: a) time-domain triangular waveform input, and b) 2-D spiral input. These simulations are described next.

4.5.1 Triangular Waveform Input

In this simulation, an input corresponding to a 200 Hz triangular waveform in the time domain was applied to all pixels in a 2 × 2 pixel array. The small array size was due to constraints in elapsed simulation time. Unfortunately, under similar conditions (200 Hz time-domain triangular waveform), the Spectre simulation generated too much output data for 8 × 8 or 4 × 4 pixel arrays, and it was not able to finish the simulation in a reasonable execution time, which would be around one week.

Figure 4.10 shows the response of the temporal change detector stage, which is itself a DVS block, in the ATIS pixels. The response was decomposed into its *on* and *off* components. In this figure, we can visually compare the ATIS electrical simulation results with the ATIS numerical simulation using the model that was presented in Section 2.2.3. In terms of TCS, the electrical simulation of the temporal change detector indicated slightly lower contrast sensitivity than what would be expected from the numerical simulation. Also, in either case (electrical or numerical simulation), applying the same input to all pixels led to request collisions. Although the designed AER responded to all requests, exact timing information was lost whenever a collision happened. The brightness measurement cycle starts with an event being generated by the temporal change detector. If the time interval between events in the same pixel is not long enough, then the brightness encoding process is not completed. According to Figure 4.10, at the initial electrical simulation time instants, as the photocurrent rises according to a triangular waveform, the logarithm of I_{pd} varies quickly, which leads to many *on* events. However, as the photocurrent at the initial time instants is not very high, the corresponding brightness measurement cycles are not completed. Taking into account the size of the figure and the point of view in Figure 4.10, visualization is not very clear. However, by looking at the top four events in Figure 4.10a, we can notice a slight variation on event timing produced by the requested collision as mentioned before.

The brightness encoding results from electrical and numerical simulations are shown in Figure 4.11. Remember that, in the ATIS context, the word 'frame' is not used in the same sense that it is used in the context of conventional frame-based vision sensors. We define an ATIS frame as all brightness values encoded by the entire pixel array within a particular time range. For visualization simplicity, the same light intensity values are shown for all pixels in one ATIS frame. In Figure 4.11, each subfigure represents one decoded ATIS frame that is valid within a particular time range. The time range, which is shown at the top part of each subfigure, indicates the time interval during which an off-chip receiver acquired the brightness information from all pixels. The gray-level color bar indicates the gray-level intensity that was assigned to the acquired brightness information. Although the exact pixel

timing information is lost, these results suggest that the exposure measurement circuit behavior corresponded to what was expected from the numerical simulations.

In Figure 4.11a, we have eight ATIS frames (estimated by numerical simulation), corresponding to the six ATIS frames obtained by electrical simulation, which are shown in Figure 4.11b. The electrical simulation results were as expected: the gray-level encoding (color bar close to the frames) shows that the photocurrent increased and decreased according to the input triangular waveform input. The gray-level values obtained from the electrical simulations were similar to those obtained from the numerical model, but not equal, because the ATIS circuits used for Spectre electrical simulations are much more complex an non-ideal effects causes the electrical simulation results to deviate from those obtained with the numerical model. We varied the V_{high} and V_{low} thresholds, and observed that the ATIS pixel array response remains similar to the response presented in Figure 4.11. Both thresholds might be adjusted in order to achieve different brightness encoding results aiming at particular specifications.

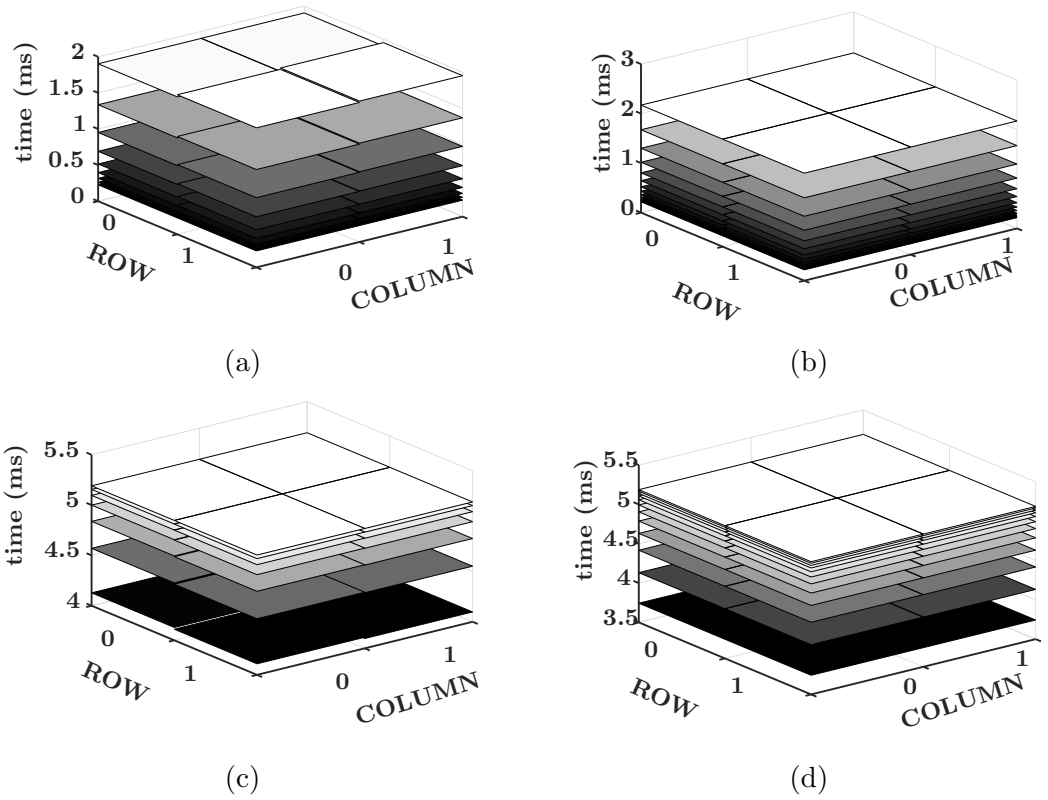
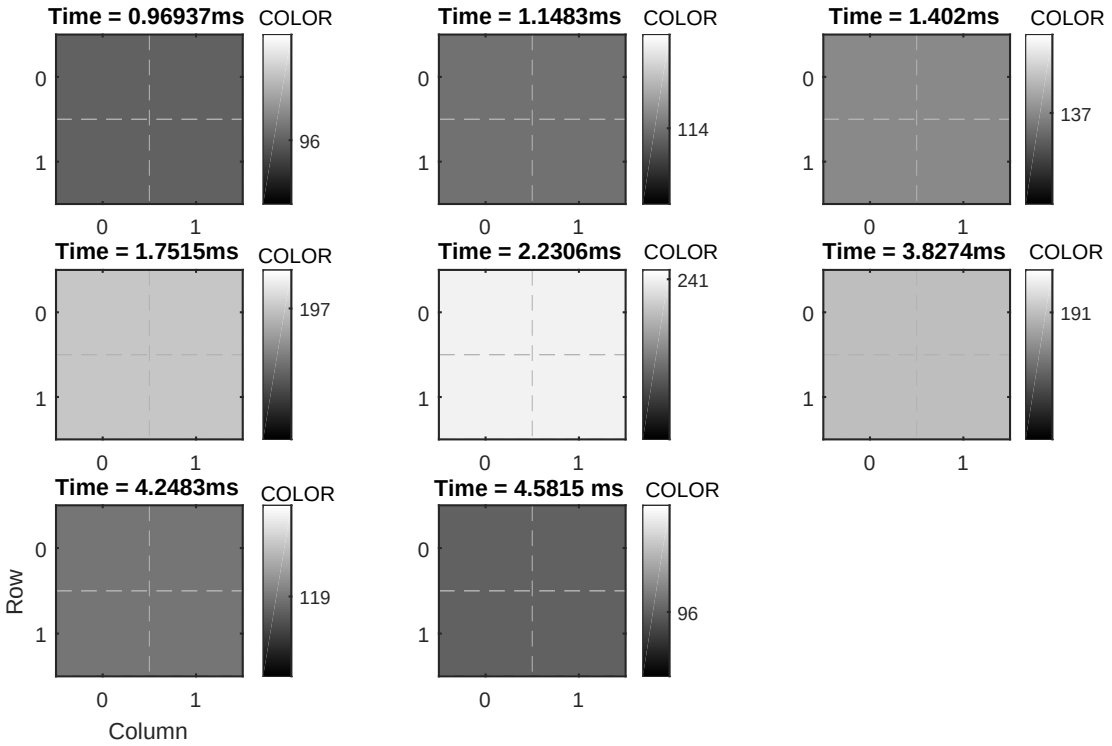
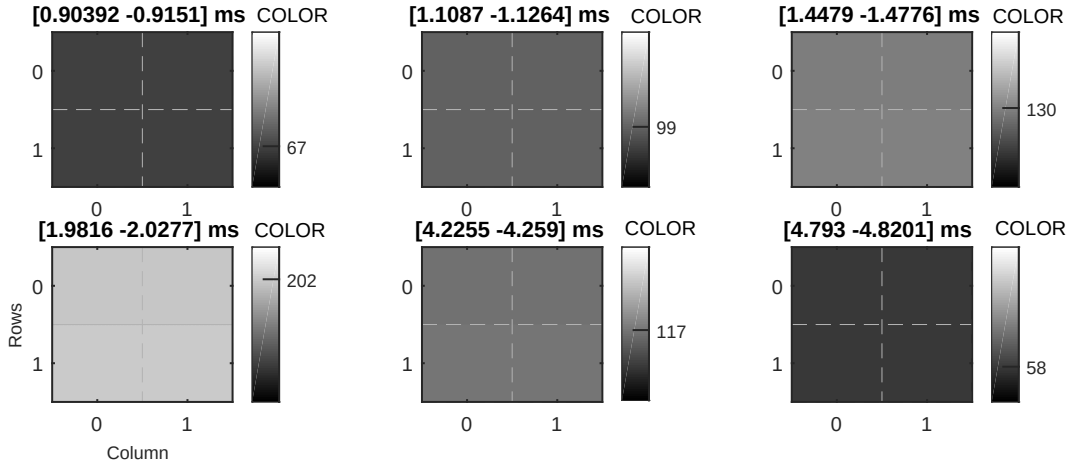


Figure 4.10: 2 × 2 ATIS pixel array. Comparison between electrical simulation results and predictions based on the numerical model in Section 2.2.3: (a) and (c) *on* and *off* events generated by electrical simulation; (b) and (d) *on* and *off* events estimated from a numerical model.



(a)



(b)

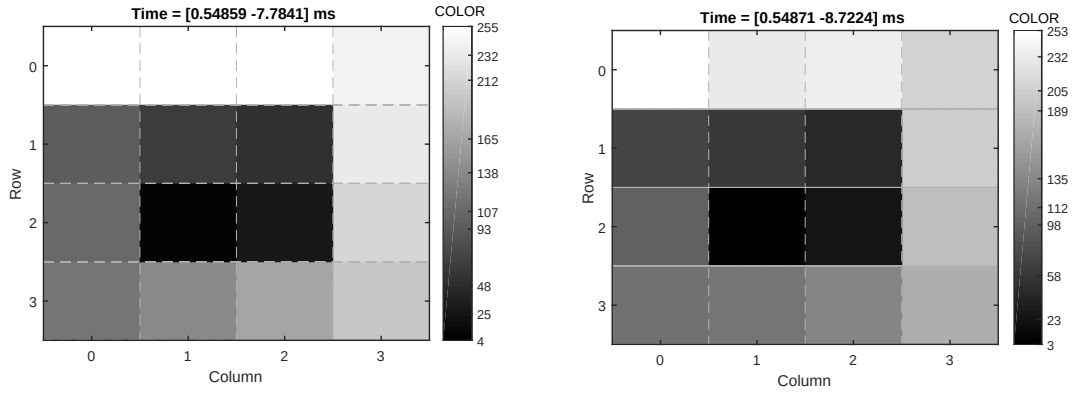
Figure 4.11: Brightness encoding results from (a) numerical simulations, and (b) electrical simulations, using the same photocurrent input with a time-domain triangular waveform for all pixels.

4.5.2 2-D Spiral Input

The second simulation was similar to the one presented for a DVS pixel array in Section 4.4. A 2-D spiral with angular frequency corresponding to 200 Hz, and decreasing light intensity, was used as the array input signal. It is slightly different from the 2-D spiral signal from Section 4.4, because elapsed simulation time con-

straints limited the array size to 4 × 4 pixels in the present section. The V_{high} and V_{low} signals were adjusted to 1.7 V and 200 mV, respectively.

Numerical simulation results are shown in Figure 4.12a and electrical simulation results are shown in Figure 4.12b. The 2-D spiral input that was used in the DVS pixel array in Section 4.4 had a constant photocurrent input for each pixel, and a gray-level code was used to denote time in the plots. In the present section, the 2-D spiral input that was applied to the ATIS pixel array has photocurrent that decreases linearly with time. As the light intensity is inversely proportional to time, 3-D plots are not required in Figure 4.12. As the temporal change detector response was similar to the temporal change detector responses that were previously described, it is not shown in this section. Figure 4.12 indicates that the brightness encoding obtained from the electrical simulation was similar to the brightness encoding that was predicted by the numerical ATIS model. Because the input signal is a 2-D spiral, request collisions did not occur, and so the event count was similar in both cases (electrical and numerical simulations). These results validate the proposed ATIS pixel design.



(a) (b)

Figure 4.12: 4 × 4 ATIS pixel array. Comparison between (a) decoded light intensity predictions based on the numerical model in Section 2.2.3, and (b) decoded light intensity values obtained from an electrical simulation.

4.6 ADMDVS 4 × 4 Pixel Array Simulation

To validate the ADMDVS pixel array design, we used a 2-D spiral input similar to the one used for the DVS pixel array in Section 4.4. The differences are that the spiral angular frequency was reduced to 10 Hz and the spiral resolution was reduced to 4 × 4 pixels. By using a lower frequency, we aimed at testing pseudo-resistors that were designed for the ADMDVS pixel. To allow all operational amplifiers to reach the correct reference voltage, we activated the global reset signal for 30 seconds. The

comparison between electrical simulation results and numerical predictions based on the model from Section 2.3.2 is presented in Figure 4.13. To make the visual comparison easier, we zoomed in at pixel number 10, which is located on row 2, column 2 of the pixel array. Figure 4.13 indicates that the electrical simulation results are close to the numerical predictions. Also, the ADMDVS output is similar to the output obtained by the basic DVS in Section 4.4, but the ADMDVS array is much more sensitive to temporal contrast change. In spite of the lower frequency, the ADMDVS generated more output events. To process the simulation data for display purposes, we used the same algorithms that were used for the basic DVS pixel array (Appendix B).

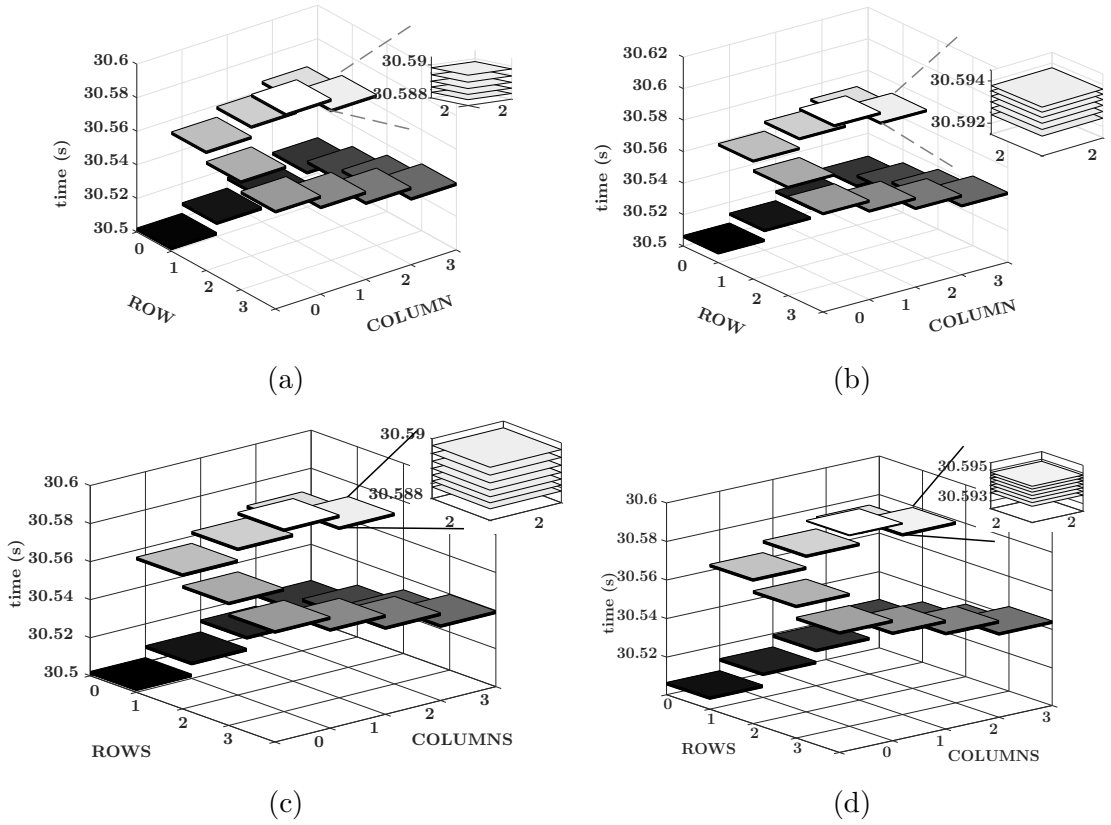


Figure 4.13: 4×4 ADMDVS pixel array. Comparison between electrical simulation results and predictions based on the numerical model in Section 2.3.2: (a) and (b) *on* and *off* events generated by electrical simulation; (c) and (d) *on* and *off* events estimated from a numerical model. Event timing details are provided for the pixel on row 2 and column 2.

4.7 DVS, ATIS and ADMDVS Comparison

Comparative remarks regarding DVS, ATIS and ADMDVS pixels and pixel arrays are provided next.

- The DVS pixel and the ADMDVS pixel do not have a brightness measurement cycle, and so they do not determine the absolute light intensity that is associated with a detected event. The decoded pixel value assumes only two possible values, say black or white. A pixel that detects positive light intensity variation is decoded as a white pixel. Otherwise, if temporal contrast change is negative, then the pixel is decoded as a black pixel. The background color is gray, which indicates the absence of events;

- The ADMDVS pixel array simulation yields more spikes than the DVS pixel array simulation. We concluded that the ADMDVS pixel TCS is larger than the DVS pixel TCS. The TCS improvement is expected, because the ADMDVS pixel uses two operational amplifiers, and the overall gain is the product of the closed-loop gains of both operational amplifiers. In the DVS pixel, on the other hand, the overall gain corresponds to a single differencing circuit;

- The ADMDVS pixel encoding mechanism never interrupts the differencing circuit. As a consequence, if no request collision occurs, then no input information ever gets lost. The DVS pixel, on the other hand, stops evaluating temporal contrast until a previously generated event has been acknowledged and treated. If the waiting time is too long, then input information may be lost;

- All three pixel types (DVS, ATIS and ADMDVS) use the same AER systems. The ATIS pixel uses an additional AER system for managing the brightness encoding process.

Unfortunately, for different reasons, we were not able to numerically estimate TCS from the simulation data. In the literature, the reported TCS figures were experimentally obtained from real pixel arrays that are much larger than 8×8 pixels [10],[15], [22]. In Spectre simulations, we were not able to make one pixel significantly different from the other, which would be useful for assessing the effects of mismatch errors. To force the Spectre simulator to consider different pixels in the array, we tried to generate a Spectre netlist (text representation of a pixel array schematic diagram) using a Monte Carlo simulation with a single run. The Monte Carlo single run netlist was used in an electrical simulation with the same input that was used in [10] for TCS characterization. However, the electrical simulation was too slow and required around 20 GB, which is too much memory. Still, we processed the available simulation data aiming at TCS computation, but we observed that all pixels had the same spike count, which was not useful for estimating the TCS of each pixel in the array.

Chapter 5

Conclusions

In this work, we modelled and simulated three different DVS pixels: the basic DVS pixel itself, ATIS, and ADMDVS. Some conclusions are presented next:

- The DVS pixel is the simplest one. It features the lowest complexity in all analog and digital parts. To reach this conclusion, we compared the pixel transistor counts in Table 3.2;
- The ATIS pixel, which includes the DVS pixel within itself as a temporal change detector, is obviously more complex than the DVS pixel. It has the same basic functionality of the DVS pixel, but it also encodes brightness into grayscale and detects temporal changes;
- The ADMDVS pixel is more complex than the DVS pixel. It also has the same basic functionality of the DVS pixel, but it achieves higher TCS than the original DVS pixel design. Besides that, the ADMDVS pixel uses a novel approach for establishing the pixel operating point after an event occurrence, which eliminates information loss as the input signal is never interrupted. In the basic DVS pixel encoding mechanism, the input signal is blocked while a pixel request is being handled, which leads to possible information loss;
- Using the ADMDVS pixel as a temporal change detector for ATIS pixels is possible, and it would lead to better grayscale encoding of incoming light intensity, at the expense of an increase in pixel complexity and size;
- Smart and efficient imaging systems may be designed with the contribution of biological neural signal processing fundamentals. For instance, the DVS, ATIS, and ADMDVS pixels asynchronously encode incoming visual information into spike trains, thus saving bandwidth and power consumption. The biological counterparts efficiency with respect to bandwidth and power are well-known [39];

- The ADMDVS and ATIS pixels implement data encoding techniques that are more advanced that the technique used in basic DVS. The ADMDVS and ATIS designs show that the use of asynchronous logic leads to an improve in efficiency;

- The EKV model in subthreshold regime allowed us to obtain a numerical model for three different DVS pixels. The numerical model response is very similar to that obtained by the electrical simulation, as shown in Chapter 4, even though the numerical method uses a simpler transistor model than the one generally utilized in electrical simulations;

- The g_m/I_D methodology is useful for low-power vision sensor design. In this work, it led to operational amplifier and voltage comparator designs with low area and good performance.

In a future investigation, pixel layout should be addressed. A comparison among several pixel layout options would allow ranking the DVS, ATIS, and ADMDVS pixels with respect to their physical size. A layout option with a good trade-off between functionality and pixel area might then be used for a pixel array fabrication. Whether or not an exposure measurement function should be adapted for the ADMDVS pixel is also an interesting question. The AER systems also require additional investigation, which would lead to more efficient AER implementations. An analysis using the events generated by each pixel to reconstruct the input signal may lead to a more specialized comparison among the pixel designs than the comparison presented in this work.

Bibliography

[1] SCHMUKER, M., SCHNEIDER, G. "Processing and classification of chemical data inspired by insect olfaction". In: Axel, R. (Ed.), *The National Academy of Sciences of the USA*, v. 104, New York, NY, jun 2007.

[2] POSCH, C., SERRANO-GOTARREDONA, T., LINARES-BARRANCO, B., et al. "Retinomorphic Event-Based Vision Sensors: Bioinspired Cameras With Spiking Output". In: *Proceedings of the IEEE*, v. 102, pp. 1470–1484, Oct 2014.

[3] MCLEAN, I. S. "Charge-coupled devices". In: *Electronic Imaging in Astronomy: Detectors and Instrumentation*, 2 ed., cap. 7, Heidelberg, Springer-Verlag Berlin Heidelberg, 2008.

[4] LICHTSTEINER, P. *An AER temporal contrast vision sensor*. Phd thesis, ETH Zurich, ETH Zürich, 2006.

[5] ATMARAM, P. *CMOS Active pixel sensors for digital cameras: current state-of-the-art*. Msc dissertation, University of North Texas, Denton, Texas, USA, 2007.

[6] BIGAS, M., CABRUJA, E., FOREST, J., et al. "Review of CMOS image sensors", *Microelectronics Journal*, v. 37, n. 5, pp. 433 – 451, May 2006.

[7] LIU, S.-C., DELBRUCK, T., INDIVERI, G., et al. *Event-based neuromorphic systems*. Wiley, 2014.

[8] DONG-IL, C., TAE-JAE, L. "A Review of Bioinspired Vision Sensors and Their Applications", *Sensors and Materials*, v. 27, n. 6, pp. 447–463, Nov 2015.

[9] DELBRUCK, T. "Frame-free dynamic digital vision". In: *Proceedings of Intl. Symp. on Secure-Life Electronics, Advanced Electronics for Quality Life and Society*, pp. 21–26, Tokyo, Mar 2008.

[10] LEÑERO BARDALLO, J., SERRANO-GOTARREDONA, T., LINARES-BARRANCO, B. "A 3.6 μs Latency Asynchronous Frame-Free Event-

Driven Dynamic-Vision Sensor", *IEEE Journal of Solid-State Circuits*, v. 46, n. 6, pp. 1443–1455, June 2011.

[11] SERRANO-GOTARREDONA, T., LINARES-BARRANCO, B. "A 128 × 128 1.5% Contrast Sensitivity 0.9% FPN 3 μs Latency 4 mW Asynchronous Frame-Free Dynamic Vision Sensor Using Transimpedance Preamplifiers", *IEEE Journal of Solid-State Circuits*, v. 48, n. 3, pp. 827–838, March 2013.

[12] POSCH, C., MATOLIN, D., WOHLGENANNT, R. "A two-stage capacitive-feedback differencing amplifier for temporal contrast IR sensors", *Analog Integrated Circuits and Signal Processing*, v. 64, n. 1, pp. 45–54, Jul 2010.

[13] CHEN, D. G., MATOLIN, D., BERMAK, A., et al. "Pulse-Modulation Imaging-Review and Performance Analysis", *IEEE Transactions on Biomedical Circuits and Systems*, v. 5, n. 1, pp. 64–82, Feb 2011.

[14] PABLO, A. *Automatic Reusable Design for Analog Micropower Integrated Circuits*. Master thesis, Instituto de Ingeniería Eléctrica, Facultad de Ingeniería, Universidad de la República, Uruguay, 2004.

[15] LICHTSTEINER, P., POSCH, C., DELBRUCK, T. "A 128 x 128 120 dB 15μs Latency Asynchronous Temporal Contrast Vision Sensor", *Solid-State Circuits, IEEE Journal of*, v. 43, n. 2, pp. 566–576, Feb 2008.

[16] POSCH, C., MATOLIN, D., WOHLGENANNT, R. "A QVGA 143dB dynamic range asynchronous address-event PWM dynamic image sensor with lossless pixel-level video compression". In: *2010 IEEE International Solid-State Circuits Conference - (ISSCC)*, pp. 400–401, Feb 2010.

[17] ENZ, C. C., KRUMMENACHER, F., VITTOZ, E. A. "An analytical MOS transistor model valid in all regions of operation and dedicated to low-voltage and low-current applications", *Analog Integrated Circuits and Signal Processing*, v. 8, n. 1, pp. 83–114, 1995.

[18] MATOLIN, D., POSCH, C., WOHLGENANNT, R. "True correlated double sampling and comparator design for time-based image sensors". In: *2009 IEEE International Symposium on Circuits and Systems*, pp. 1269–1272, May 2009.

[19] KURODA, T. "Major Types of Noise in Image Sensors". In: *Essential Principles of Image Sensors*, 1 ed., cap. 3, Boca Raton, FL, CRC Press, 2014.

[20] JIN, X. "Principle of simple correlated double sampling and its reduced-area low-noise low-power circuit realization", *Analog Integrated Circuits and Signal Processing*, v. 65, n. 2, pp. 209–215, Jun 2010.

[21] YANG, M., LIU, S. C., DELBRUCK, T. "Comparison of spike encoding schemes in asynchronous vision sensors: Modeling and design". In: *2014 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 2632–2635, June 2014.

[22] YANG, M., LIU, S. C., DELBRUCK, T. "A Dynamic Vision Sensor With 1% Temporal Contrast Sensitivity and In-Pixel Asynchronous Delta Modulator for Event Encoding", *IEEE Journal of Solid-State Circuits*, v. 50, n. 9, pp. 2149–2160, Sept 2015.

[23] BEEREL, P. A. *A Designer's guide to Asynchronous VLSI*. 1 ed. Cambridge, Cambridge University Press, 2010.

[24] SPARSØ, J., FURBER, S. *Principles of Asynchronous Circuit Design: A Systems Perspective*. Springer, 2001.

[25] MANOHAR, R., MARTIN, A. J. *Quasi-Delay-Insensitive Circuits Are Turing-Complete*. Technical report, California Institute of Technology, Pasadena, CA, USA, 1995.

[26] MAHOWALD, M. *VLSI analogs of neuronal visual processing: a synthesis of form and function*. Phd thesis, California Institute of Technology, Pasadena, CA, USA, 1992.

[27] LANDE, T. S. "Introduction to Neuromorphic Communication". In: Lande, T. S. (Ed.), *Neuromorphic Systems Engineering: Neural Networks in Silicon*, Kluwer Academic Publishers, cap. 8, pp. 193–200, Norwell, MA, USA, 1998.

[28] BOAHEN, K. A. "Point-to-point connectivity between neuromorphic chips using address events", *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, v. 47, n. 5, pp. 416–434, May 2000.

[29] SCHNEIDER, M. C., GALUP-MONTORO, C. *CMOS Analog Design Using All-Region MOSFET Modeling*. 1 ed. New York, Cambridge University Press, 2010.

[30] SILVEIRA, F., FLANDRE, D., JESPERS, P. G. A. "A g_m/I_D based methodology for the design of CMOS analog circuits and its application to the

synthesis of a silicon-on-insulator micropower OTA", *IEEE Journal of Solid-State Circuits*, v. 31, n. 9, pp. 1314–1319, Sep 1996.

[31] JESPERS, P. *The g_m/I_D Methodology, a Sizing Tool for Low-voltage Analog CMOS Circuits: The Semi-empirical and Compact Model Approaches*. Springer US, 2009.

[32] LI, X., WU, W., GILDENBLAT, G., et al. "PSP 102.3". Retrieved from : <`http://www.nxp.com/wcm_documents/models/mos-models/model-psp/psp102p3_summary.pdf`>. Accessed: 1 December 2016.

[33] DELBRUCK, T., MEAD, C. A. "Adaptive photoreceptor with wide dynamic range". In: *Proceedings of IEEE International Symposium on Circuits and Systems - ISCAS '94*, v. 4, pp. 339–342, May 1994.

[34] YANG, M. *Silicon Retina and Cochlea with Asynchronous Delta Modulator for Spike Encoding*. Phd thesis, ETH Zurich, ETH Zürich, 2015.

[35] ALLEN, P., HOLBERG, D. *CMOS Analog Circuit Design*. OUP USA, 2012.

[36] PELGROM, M. J. M., DUINMAIJER, A. C. J., WELBERS, A. P. G. "Matching properties of MOS transistors", *IEEE Journal of Solid-State Circuits*, v. 24, n. 5, pp. 1433–1439, Oct 1989.

[37] KIM, G., KIM, M.-K., CHANG, B.-S., et al. "A low-voltage, low-power CMOS delay element", *IEEE Journal of Solid-State Circuits*, v. 31, n. 7, pp. 966–971, Jul 1996.

[38] ZHANG, J., COOPER, S. R., LAPIETRA, A. R., et al. "A low power thyristor-based CMOS programmable delay element". In: *Circuits and Systems, 2004. ISCAS '04. Proceedings of the 2004 International Symposium on*, v. 1, pp. I–769–72, May 2004.

[39] INDIVERI, G., HORIUCHI, T. "Frontiers in Neuromorphic Engineering", *Frontiers in Neuroscience*, v. 5, n. 118, pp. 1–2, Oct 2011.

Appendix A

Pixel Array Simulation

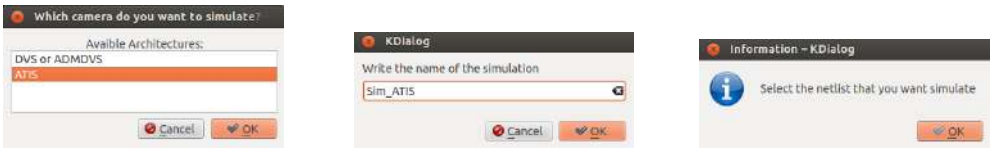
To make the simulation stage described in Chapter 4 automatic, we used three programming languages: Bash, MATLAB, and Python. To design low-complexity algorithms for processing simulation data, aiming at displaying results that are useful for validating pixel design, we took advantage of each programming language. For all pixel array simulations, we used a directory structure such as the following one:

```
Cadence Analysis
├── Inputs/
│   └── Spiral4x4_ATIS/
├── Netlist_spectre/
│   └── ATIS4x4.scs
├── Scripts/
│   ├── matlab/
│   │   ├── exe_sim_DVSs.sh (Bash)
│   │   ├── plot2dATIS.m (Matlab)
│   │   ├── plot3dDVS.m (Matlab)
│   │   └── plotTran_ATIS.m (Matlab)
│   └── python/
│       ├── setting_input_netlist_ATIS.py
│       └── sort_data_ATIS_pixel.py
└── Cameras_simulation
    └── Sim_ATIS/
        ├── images/
        ├── Spectredata.raw/
        ├── output2matlab/
        ├── netlist.scs
        ├── env_var.sh
        └── Readme.txt
```

The directory structure presented above corresponds to an ATIS pixel array

simulation, because ATIS pixels have, inside them, a DVS pixel working as a TCD. The scripts that were designed for this pixel array are similar to the scripts that are used for the other pixel arrays. The directory structure is composed mainly by four folders: *Inputs*, *Netlist_spectre*, *Scripts*, and *Cameras_simulation*. The *Inputs* folder contains input stimuli used for DVS simulations. The *Netlist_spectre* contains netlist descriptions of all DVS pixel array circuits that were designed. The *Scripts* folder contains two folders, in which algorithms are separated according to the programming language (MATLAB or Python). The single Bash script that we used is located in the *matlab* folder. The *Cameras_simulation* folder contains successfully accomplished DVS simulations. For each simulation, a folder is created. For example, we can see a *Sim_ATIS* folder containing valid simulation data within the presented directory structure. The *output2matlab* folder contains the same simulation output data, but in this folder the output data are save in a 'comma-separated value' (CSV) file that can be easily imported by MATLAB.

Simulation parameters are defined by the user through a sequence of interactive prompts. The *exe_sim_DVSs* script implements graphical interaction with the user. In Figures A.1, A.2, and A.3, we show the steps that are necessary for an ATIS pixel array simulation configuration. For a basic DVS sensor, the simulation configuration is very similar, but the user receives fewer messages. Initially, the user chooses the pixel array type (Figure A.1(a)). After that, the user is asked for a simulation name (e.g. *Sim_ATIS*) (Figure A.1(b)). The script itself creates the required files in the *Cameras_simulation* folder. Next, the user must select a circuit netlist representation within the available ones (Figures A.1(c) and A.2(a)).

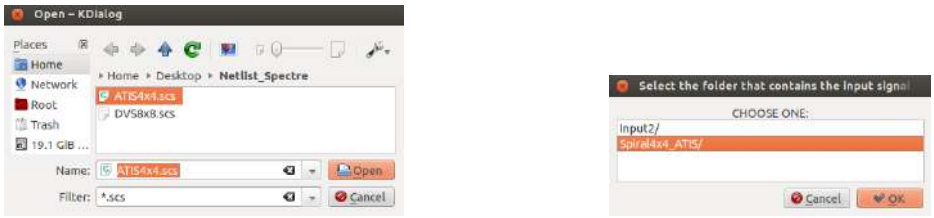


(a) (b) (c)

Figure A.1: Simulation configuration, first step: (a) pixel array type selection, (b) simulation name assignment, and (c) confirmation.

After the netlist selection, the user must specify a folder containing input stimuli (Figure A.2(b)). The script looks, inside the stimuli folder, for a *README.txt* file containing input signal features such as:

```
N 4
M 4
T 1e-01
freq 10 (Hz)
Tdelay 5e-01
```



(a) (b)

Figure A.2: Simulation configuration, second step: (a) netlist selection, and (b) input stimuli folder selection.

where 'N' and 'M' are the numbers of columns and rows of the pixel array. The *README.txt* file contents should be independent of the pixel array type, although they do depend on the array dimensions. The input signal period is 'T'. The input signal frequency is 'freq'. The time instant at which the global reset signal is activated is 'Tdelay'. Other electrical simulation parameters are defined in the same way as in Figures A.1 and A.2. The parameters that are required to start the simulation are shown in Figure A.3 (number of bits in the data bus, V_{ref}, $V_{d,on}$, and so forth). By hitting the 'Cancel' button, the user aborts the configuration process and the simulation is not launched.

(a) (b) (c)

(d) (e) (f)

(g) (h) (i)

Figure A.3: Simulation configuration, third step: electrical simulation parameters.

After all questions were answered, the *exe_sim_DVSs* script executes the following scripts sequentially:

1. *setting_input_netlist_ATIS*: this script modifies the original netlist, which had been created by the Spectre simulator, and it generates a new netlist with the specified features: electrical parameters and input stimulus;

2. *sort_data_ATIS_pixel*: at this point, the simulation output data was generated by Spectre. This script reads simulation output data, and it creates a new

file containing the same information in CSV format for MATLAB. In fact, the script creates four files: i) the CSV data file, which contains all time-domain (transient) simulation results; ii) a text file that indicates which data file columns must be read for executing the DVS analysis; iii) a text file that indicates which data file columns must be read for executing the ATIS analysis; and iv) a text file containing data file signal labels, which is useful for plotting the signals;

3. em ATIS_Model: this script computes the ATIS expected response based on the numerical model that was presented in Section 2.2.3. It uses the same input signals that were used for the corresponding electrical simulations.

The scripts designed for DVS simulations are similar to the ones that were presented, above, for ATIS simulations. For example, the *setting_input_netlist_DVS* script replaces the *setting_input_netlist_ATIS* script. The major difference, in this case, is that DVS and ADMDVS pixels use a single photodiode, so the script modifies the input stimulus for the TCD (i.e. basic DVS) inside the ATIS pixels. In Appendix B, we show the complete codes the implement the functions that were mentioned in the present appendix.

Appendix B

Codes for simulating DVS cameras

In this appendix, we show the complete codes the implement the functions that were mentioned in Appendix A. The *ATIS_Model* is omitted, because it implements the simple model that was presented earlier, in Section 2.2.3.

Listing B.1: Main program for executing DVSs camera simulations

```
#!/bin/sh
# ------ Creating the folder to the simulation ------
PATH_scriptMatlab=$PWD/
cd $PATH_scriptMatlab
cd ..
PATH_script=$PWD/
PATH_scriptPython=$PATH_script"python"/
cd ..
choice_TypeSim=$(kdialog --menu "Avaible Architectures:" \
        1 "DVS or ADMDVS" 2 "ATIS" \
        --title "Which camera do you want to simulate?")
echo $choice_TypeSim
# ---------- Creating the common folders ----------
namefolder_netlistSpectre="Netlist_Spectre"
mkdir $namefolder_netlistSpectre
PATH_netlist_spectre=$PWD/$namefolder_netlistSpectre/
namefolder_simulation="Cameras_simulation"
mkdir $namefolder_simulation
PATH_simulation=$PWD/$namefolder_simulation/
namefolder_inputs="Inputs"
mkdir $namefolder_inputs
PATH_namefolder_Input=$PWD/$namefolder_inputs/
cd $PATH_simulation
# ---------- Reading user input ----------
name_simulation=$(kdialog \
    --inputbox "Write the name of the simulation");
if [ "$?" -ne 0 ]; then
  kdialog --error "Simulation Aborted"
  cd $PATH_scriptMatlab
  return
fi;
mkdir $name_simulation
# Guarantee that the simulation has a unique name
while [ $? -ne 0 ]
do
  kdialog --yesno \
  "The directory exist  Do you want re-write it?";
  if [ "$?" = 0 ]
  then
    rm -fR $name_simulation
    mkdir $name_simulation
  elif [ "$?" = 1 ]; then
    name_simulation=$(kdialog --inputbox \
      "Write a different name for the simulation");
    if [ "$?" = 0 ]
    then
      mkdir $name_simulation
    else
      return
    fi;
  else
    kdialog --error "Simulation Aborted";
    cd $PATH_scriptMatlab
    return
```

```
55    fi;
56  done
57  cd $PATH_netlist_spectre
58  # Here is chosen the netlist to simulate
59  kdialog --msgbox "Select the netlist that you want simulate"
60  PATH_nameNetlist_spectre=$(kdialog --getopenfilename . "*.scs ");
61  if [ "$?" -ne 0 ]; then
62    kdialog --error "Simulation Aborted"
63    cd $PATH_scriptMatlab
64    return
65  fi;
66  # extract the name of the netlist
67  nameNetlist_spectre=$(echo "$PATH_nameNetlist_spectre" | sed "s/.*\///")
68  nameNetlist_spectre_Orig=$nameNetlist_spectre
69  ## Selecting the folder input
70  cd $PATH_namefolder_Input
71  count=1
72  a=""
73  direc=()
74  for i in $(ls -d */);
75  do
76          a=$a$(echo "$count  ${i%%/} ")
77          direc[$((count-1))]=${i%%/}
78          count=$((count+1))
79  done
80  echo $a
81  # Selecting the folder that contain the input signal
82  choice=$(kdialog --menu "CHOOSE ONE:" $a \
83      --title "Select the folder that contains the input signal desired")
84  echo $choice
85  echo ${direc[$((choice-1))]}
86  name_Signalsinput=${direc[$((choice-1))]}
87  name_Signalsinput=$(echo ${name_Signalsinput///})
88  if [ "$?" = 0 ]; then
89    PATH_inputs=$PATH_namefolder_Input$name_Signalsinput
90  else
91    kdialog --error "Simulation Aborted"
92    cd $PATH_scriptMatlab
93    return
94  fi;
95  cd $PATH_scriptMatlab
96  # Setting of the simulation
97  number_bits=$(kdialog --inputbox "How many bits have the data bus?");
98  if [ "$?" -ne 0 ]; then
99    kdialog --error "Simulation Aborted"
100   cd $PATH_scriptMatlab
101   return
102 fi;
103 N=$(kdialog --inputbox "How many columns have your camera?");
104 if [ "$?" -ne 0 ]; then
105   kdialog --error "Simulation Aborted"
106   cd $PATH_scriptMatlab
107   return
108 fi;
109 M=$(kdialog --inputbox "How many rows have your camera?");
110 if [ "$?" -ne 0 ]; then
111   kdialog --error "Simulation Aborted"
112   cd $PATH_scriptMatlab
113   return
114 fi;
115 Vref=$(kdialog --inputbox "Write the Vref in long format");
116 if [ "$?" -ne 0 ]; then
117   kdialog --error "Simulation Aborted"
118   cd $PATH_scriptMatlab
119   return
120 fi;
121 Vdoff=$(kdialog --inputbox "Write the Vdoff in long format");
122 if [ "$?" -ne 0 ]; then
123   kdialog --error "Simulation Aborted"
124   cd $PATH_scriptMatlab
125   return
126 fi;
127 Vdon=$(kdialog --inputbox "Write the Vdon in long format");
128 if [ "$?" -ne 0 ]; then
129   kdialog --error "Simulation Aborted"
130   cd $PATH_scriptMatlab
131   return
132 fi;
133 # For ATIS
134 if [ "$choice_TypeSim" = 2 ]
135 then
136   Vhigh=$(kdialog --inputbox "Write the Vhigh in long format");
137   if [ "$?" -ne 0 ]; then
138     kdialog --error "Simulation Aborted"
139     cd $PATH_scriptMatlab
140     return
141   fi;
142   Vlow=$(kdialog --inputbox "Write the Vlow in long format");
143   if [ "$?" -ne 0 ]; then
```

```
144     kdialog --error "Simulation Aborted"
145     cd $PATH_scriptMatlab
146     return
147   fi;
148 fi
149 comment_simulation=$(kdialog --inputbox \
150     "Write some comments of your Simulation")
151 if [ "$?" -ne 0 ]; then
152   comment_simulation="No comments!"
153 fi;
154 #--------  CREATING DIRECTORIES  ----------
155 PATH_folder_simulation=$PATH_simulation$name_simulation/
156 # Copy the original netlist to the folder Simulation
157 cp -f $PATH_nameNetlist_spectre $PATH_folder_simulation
158 cd $PATH_folder_simulation
159 cp -f $nameNetlist_spectre "netlist_"$name_simulation".scs"
160 # Write the comment simulation into the README_SIM.txt
161 echo "Original Netlist $nameNetlist_spectre" >> README_SIM.txt
162 echo "==========COMMENT ==========" >> README_SIM.txt
163 echo $comment_simulation >> README_SIM.txt
164 nameNetlist_spectre="netlist_"$name_simulation".scs"
165 name_folder_matlab_output="output_matlab"
166 name_matlab_output="output_matlab_"$name_simulation".csv"
167 name_images="images"
168 mkdir $name_folder_matlab_output
169 mkdir $name_images
170 PATH_sim_output_matlab=$PATH_folder_simulation$name_folder_matlab_output/
171 PATH_folder_input=$PATH_inputs/
172 PATH_folder_images=$PATH_folder_simulation$name_images/
173 name_folder_output_Spectre="netlist_"$name_simulation".raw"
174 cd $PATH_folder_simulation
175 #------------ SAVE VARIABLES ------------ #
176 array_vars=(PATH_nameNetlist_spectre PATH_scriptMatlab \
177 PATH_script PATH_scriptPython \
178 PATH_netlist_spectre PATH_simulation PATH_folder_simulation \
179 PATH_sim_output_matlab PATH_folder_input PATH_folder_images \
180 name_simulation name_Signalsinput name_folder_matlab_output \
181 name_matlab_output name_images name_folder_output_Spectre \
182 number_bits M N Vref Vdoff Vdon Vhigh Vlow \
183 nameNetlist_spectre_Orig nameNetlist_spectre)
184 # Export environment variables
185 for item in ${array_vars[*]}
186 do
187     export $item
188 done
189 # ... Write the file ...
190 echo -n "" > env_var.sh  #Clear the file
191 echo "#!/bin/bash" >> env_var.sh
192 for item in ${array_vars[*]}
193 do
194     printf "%s\n" $item=${!item} >> env_var.sh
195 done
196 for item in ${array_vars[*]}
197 do
198     printf "export %s\n" $item >> env_var.sh
199 done
200 #-----------------------------------------------#
201 # Executing Spectre and Script to processing the simulation response
202
203 case "$choice_TypeSim" in
204   1) #DVS
205     clear
206     echo "Simulating a DVS camera. It could take some \
207     minutes, hours or even days please wait"
208     # Execution of DVS commands
209     cd $PATH_scriptPython
210     python setting_input_netlist_UNIX_DVS2.py
211     cd $PATH_folder_simulation
212     spectre +mt ++aps -format psfascii $nameNetlist_spectre
213     if [ "$?" = 0 ]
214     then
215       sleep 50
216       cd $PATH_scriptPython
217       python sort_data_DVS_pixel_UNIX.py
218       if [ "$?" = 0 ]
219     then
220       cd $PATH_scriptMatlab
221       matlab -nodesktop -nosplash -r Model_CamDVS
222       matlab -nodesktop -nosplash -r plotTran_DVS
223     else
224       kdialog --error "Errors found in the Python script. Please verify it"
225       fi
226     else
227       kdialog --error \
228     "Please verify if the Spectre environment variables are properly configured"
229       cd $PATH_scriptMatlab
230       return
231     fi
232     ;;
```

```
  2) #ATIS
    clear
    echo "Simulating an ATIS camera wait it could take some minutes, hours or days"
    cd $PATH_scriptPython
    python setting_input_netlist_UNIX_ATIS.py
    cd $PATH_folder_simulation
    spectre +mt ++aps -format psfascii $nameNetlist_spectre
    # Analysis for ATIS Pixel
    if [ "$?" = 0 ]
    then
      sleep 50
      cd $PATH_scriptPython
      python sort_data_ATIS_pixel_UNIX.py
      if [ "$?" = 0 ]
      then
    cd $PATH_scriptMatlab
    matlab -nodesktop -nosplash -r ATIS_Model
    matlab -nodesktop -nosplash -r plotTran_ATIS
      else
    kdialog --error "Errors found in the Python script. Please verify it"
      fi
    else
      kdialog --error \
    "Please verify if the Spectre environment variables are properly configured"
      cd $PATH_scriptMatlab
      return
    fi
esac
cd $PATH_scriptMatlab
```

Listing B.2: Modifying the ATIS netlist

```
import os
import sys
import re
# ========          Declarations          ======== #
PATH_nameNetlist_spectre = os.environ['PATH_nameNetlist_spectre']
PATH_folder_simulation = os.environ['PATH_folder_simulation']
nameNetlist_spectre_Orig = os.environ['nameNetlist_spectre_Orig']
nameNetlist_spectre = os.environ['nameNetlist_spectre'] # netlist final
PATH_folder_input = os.environ['PATH_folder_input']
name_Signalsinput = os.environ['name_Signalsinput']
Vdoff = float(os.environ['Vdoff'])
Vdon = float(os.environ['Vdon'])
Vhigh = float(os.environ['Vhigh'])
Vlow = float(os.environ['Vlow'])
Vref = float(os.environ['Vref'])
ext_input = '.csv'
N = int(os.environ['N'])
M = int(os.environ['M'])
quant_pixels = N*M
T_Rst=1e-3;
os.chdir(PATH_folder_simulation)
name_n_VoltageDVS = 'V_pdD'
name_n_VoltageATIS = 'V_pdA'
name_n_CurrentDVS = 'I_pdD'
name_n_CurrentATIS = 'I_pdA'
# ========          Variables          ======== #
l_nodevoltageNames = []   # Save the name of the node voltage DVS
l_nodecurrentNames = []   # Save the name of the nodes current name DVS
l_nodevoltageNames_A = [] # Save the name of the node voltage ATIS
l_nodecurrentNames_A = [] # Save the name of the nodes current name ATIS
# ========      Open and Write Files      ======== #
f = open(nameNetlist_spectre_Orig,'r')
f_readme=open(PATH_folder_input+'README.txt','r') # Read the features of the input signal
f_netlist = open(nameNetlist_spectre,'w')
l_readme=list(f_readme.readlines())
l_readme=[w.replace('\n','') for w in l_readme]
l_netlist = list(f.readlines())
l_netlist = [w.replace('\n','') for w in l_netlist] # Stores the Netlist in a list
len_netlist = len(l_netlist)
f.close()
f_readme.close()
# ========= Extracting the Period Signal ======== #
regex=re.compile(".*(T )")
sub_list = filter(regex.match, l_readme)
sub_list = sub_list[0].split(' ')
T=sub_list[2]
# ========      Open and Write Files      ======== #
l_nodevoltageNames=[name_n_VoltageDVS+str(i) for i in range(0,quant_pixels)]
l_nodecurrentNames=[name_n_CurrentDVS+str(i) for i in range(0,quant_pixels)]
l_nodevoltageNames_A=[name_n_VoltageATIS+str(i) for i in range(0,quant_pixels)]
l_nodecurrentNames_A=[name_n_CurrentATIS+str(i) for i in range(0,quant_pixels)]
# ========      Open and Write Files      ======== #
f_netlist.seek(0,0)
# Modify the parameters
regex=re.compile("(parameters)")
```

```
sub_list = filter(regex.match, l_netlist);i=l_netlist.index(sub_list[0])
new_line = ('parameters Vdon=%1.3f'%Vdon+' Vdoff=%1.3f'%Vdoff+' T=%1.10f'%float(T)+
            ' T_Rst=%1.10f'%T_Rst+' Vhigh=%1.3f'%float(Vhigh)+ ' Vlow=%1.3f'%float(Vlow)+
            ' Vref=%1.3f'%Vref)
l_netlist[i]=new_line
# Modify the total time simulation
regex=re.compile("(tran tran)")
sub_list = filter(regex.match, l_netlist);i=l_netlist.index(sub_list[0])
sub_list = sub_list[0].split(' ')
regex=re.compile("(stop)")
string = filter(regex.match, sub_list); j = sub_list.index(string[0])
new_line =  'stop=%1.10f'%(T_Rst+float(T))
sub_list[j]= new_line
l_netlist[i]=' '.join(sub_list)
# Setting up the input signal and change the
# all sources by intuitives names
for i in range(0,quant_pixels):
    # DVS
    regex=re.compile(".*(pdD"+str(i)+" 0)")
    sub_list = filter(regex.match, l_netlist)
    pos_ini = l_netlist.index(sub_list[0])
    j = pos_ini + 1
    while j <len_netlist:
        string=l_netlist[j]
        lst_tmp=string.split(' ')
        if lst_tmp[0] != '':
            nxt_ins=j
            j=len_netlist
        else:
            j=j+1
    del l_netlist[pos_ini:nxt_ins]
    line1 = l_nodecurrentNames[i]+' ('+l_nodevoltageNames[i]+' 0) isource \\'
    line2 = '       file="+'"'+ PATH_folder_input+name_SignalsinputT'_'+str(i)+ext_input+'"'+' \\'
    line3 = '       type=pwl delay=T_Rst edgetype=halfsine scale=1 stretch=1 pwlperiod=T'
    l_netlist.insert(pos_ini,line1)
    l_netlist.insert(pos_ini+1,line2)
    l_netlist.insert(pos_ini+2,line3)
    # ATIS
    regex=re.compile(".*(pdA"+str(i)+" 0)")
    sub_list = filter(regex.match, l_netlist)
    pos_ini = l_netlist.index(sub_list[0])
    j = pos_ini + 1
    while j < len_netlist:
        string=l_netlist[j]
        lst_tmp=string.split(' ')
        if lst_tmp[0] != '':
            nxt_ins=j
            j=len_netlist
        else:
            j=j+1
    del l_netlist[pos_ini:nxt_ins]
    line1 = l_nodecurrentNames_A[i]+' ('+l_nodevoltageNames_A[i]+' 0) isource \\'
    line2 = '       file="+'"'+ PATH_folder_input+name_SignalsinputT'_'+str(i)+ext_input+'"'+' \\'
    line3 = '       type=pwl delay=T_Rst edgetype=halfsine scale=1 stretch=1 pwlperiod=T'
    l_netlist.insert(pos_ini,line1)
    l_netlist.insert(pos_ini+1,line2)
    l_netlist.insert(pos_ini+2,line3)
# ========      Write netlist     ======== #
x = 0
len_netlist = len(l_netlist) # calculates the length of the list 'l_netlist'
while x < len_netlist:
    f_netlist.write(l_netlist[x])
    f_netlist.write('\n')
    x = x + 1
f_netlist.close()
```

Listing B.3: Reading the data simulation from Spectre simulator

```
from getIndex_desiredSignals import getIndex_desiredSignals
import os
# ========          Global variables        ======== #
PATH_sim_output_matlab = os.environ['PATH_sim_output_matlab']
PATH_folder_simulation = os.environ['PATH_folder_simulation']
name_simulation = os.environ['name_simulation']
name_folder_output_Spectre = os.environ['name_folder_output_Spectre']
number_bits = int(os.environ['number_bits'])
name_tran = 'tran.tran'
l_signals = ['time']                    #by default if the simulation was transient
string_start = 'VALUE'                  #default
string_stop = 'END'                     #default
string_index_file = 'index_data'        #DVS
string_index_file_A = 'index_data_A'    #ATIS
string_data = 'data_'+name_simulation
# ========      Defining desired signals    ======== #
# DVS
lst_desired_signals = ['data','En_Read_Row','En_Read_pixel','Global_rst']
lst_bus_data = ['data']
```

```
# ATIS
lst_desired_signals_A = ['data_A','En_Read_Row_A','En_Read_pixel_A','Global_rst','Req_fr']
lst_bus_data_A = ['data_A']
# ========= Sorting output simulation file  ========#
File = open(PATH_folder_simulation+name_folder_output_Spectre+'/'+name_tran,'r')
file_data = open(PATH_sim_output_matlab+string_data+'.csv','w')       #Output readable for Matlab
path_file_index = PATH_sim_output_matlab+string_index_file+'.csv'     #Index desired DVS signal
path_file_index_A = PATH_sim_output_matlab+string_index_file_A+'.csv' #Index desired ATIS signal
file_header = open(PATH_sim_output_matlab+'header.txt','w')           #file_data headers
l_output = list(File.readlines())
l_output = [w.replace('\n','') for w in l_output]   # removes the '\n' string
len_output = len(l_output)                          # counts the total elements in l_output
line_TRACE = l_output.index('TRACE')                # since the keyword TRACE starts the signals
    obtained from sim.
line_VALUE = l_output.index('VALUE')                # Until the keyword VALUE ends the signals
    obtained from sim.
x = line_TRACE+1
# === obtaining the signals names from simulation ===#
while x < line_VALUE:
    value = l_output[x].split()
    if ( len(value) == 1 ):
        # It is not a valid signal
        x = x + 1
    else:
        string = value[1]
        if ( string == '"A"'):
            # It is not a valid signal
            x = x + 1
        else:
            # It is a valid signal
            string = value[0]
            string = string.replace('"','')
            l_signals.append(string)
            x = x + 1
# === Calling external function to create the indexes ===#
#DVS
getIndex_desiredSignals(l_signals,lst_desired_signals,
                        lst_bus_data,number_bits,path_file_index)
#ATIS
getIndex_desiredSignals(l_signals,lst_desired_signals_A,
                        lst_bus_data_A,number_bits,path_file_index_A)
# ==========    Writing the output files      ==========#
len_signals = len(l_signals) # counts the elements in l_signals
i_start = line_VALUE + 1     # sets up the index where the for-loop needs to start.
x = i_start
file_data.seek(0,0)
string_header = ','.join(l_signals)
file_header.write(string_header) # writes the headers
file_header.close()
while x < len_output:
    if (l_output[x] == string_stop):
        break
    else:
        for i in range(0,len_signals):
            value = l_output[x+i].split()
            value = value[1]
            file_data.write(value)
            if ( i == len_signals-1):
                file_data.write('\n')
            else:
                file_data.write(',')
    x = x + len_signals
file_data.close()
```

Listing B.4: Processing the ATIS data

```
% Este script plotea la salida de una camara DVS
clear all; clc; close all;
pwd_current = pwd;
tic; % It is for measuring the elapsed time
%% ==========  Gets global variables ==========%%
PATH_sim_output_matlab = getenv('PATH_sim_output_matlab');
name_simulation = getenv('name_simulation');
PATH_folder_images = getenv('PATH_folder_images');
number_bits = str2num(getenv('number_bits'));
N = str2num(getenv('N'));
M = str2num(getenv('M'));
Vhigh = str2num(getenv('Vhigh'));
Vlow = str2num(getenv('Vlow'));
PATH_input = getenv('PATH_folder_input');
name_signal = getenv('name_Signalsinput');
T_Rst = 1e-3;
cd(PATH_sim_output_matlab)
%% ========== Reading the data files ==========%%
string_data = strcat('data_',name_simulation,'.csv');% Name simulation file
string_index_file = 'index_data.csv';                % Name indexes DVS signals
string_index_file_A = 'index_data_A.csv';            % Name indexes ATIS signals
```

```
22 data_Sim = importdata(string_data);                    % Saves simulation in data_Sim
23 middle_point = 0.9;                                    % It defines from which value is high or low
24 %% ========== Processing of DVS data ========== %%
25 % Struct = {'time' 'data<0:up(log2(N*M)+1)>','En_Read_Row','En_Read_pixel','Global_rst'}
26 index_desired = importdata(string_index_file);
27 len_index = length(index_desired);
28 len_row_data_Sim = length(data_Sim);
29 digitalSignal = zeros(len_row_data_Sim,len_index);
30 vec_pixels = zeros(len_row_data_Sim,1);
31 time = data_Sim(:,1);
32 % ADC converting process
33 for i=1:len_index
34     signalX= data_Sim(:,index_desired(i)+1); % Plus 1 to include the time signal
35     index_ONE= find(signalX >= middle_point);
36     index_ZERO= find(signalX <middle_point);
37     signalX(index_ONE) = 1;
38     signalX(index_ZERO) = 0;
39     if (i <= number_bits-1)
40         signalX = (2^(i-1))*signalX;
41         vec_pixels = vec_pixels + signalX;
42     end
43     digitalSignal(:,i)=signalX;
44 end
45 % Defining signals of intereset
46 index_Threshold    = number_bits;
47 index_En_ReadRow   = number_bits+1;
48 index_En_ReadPixel = number_bits+2;
49 index_Globalrst    = number_bits + 3;
50 kindEvent = digitalSignal(:,index_Threshold); % Kind event Threshold ON / OFF
51 EnReadRow = digitalSignal(:,index_En_ReadRow); %en read row
52 EnReadPix = digitalSignal(:,index_En_ReadPixel); %en read pixel
53 GlobalRst = digitalSignal(:,index_Globalrst); %global reset
54 % Defining when an event occurred. It can be obtained seeing the EnReadPix
55 % signal when its value be equal to one we have a valid data.
56 state = 0;
57 ON_events = {[]};
58 OFF_events = {[]};
59 ind_ON=1;
60 ind_OFF = 1;
61 i = find(time > T_Rst,1); % Sets up the index i after the time reset defined
62 while (i<=len_row_data_Sim)
63     if (state == 0)
64         % Finds when EnReadPix signal rises
65         ind_En_RdPix = find(EnReadPix(i:len_row_data_Sim) == 1,1) + i - 1;
66         if isempty(ind_En_RdPix)
67             % Exit loop
68             i = len_row_data_Sim + 1;
69         else
70             % Verifies which kind of event occurred
71             if kindEvent(ind_En_RdPix) == 0
72                 % ON EVENT
73                 t = time(ind_En_RdPix);
74                 pixel = vec_pixels(ind_En_RdPix);
75                 vec_time_pix = [t pixel];
76                 ON_events{ind_ON} = vec_time_pix;
77                 ind_ON = ind_ON + 1;
78                 state = 1;
79             else
80                 % OFF Event
81                 t = time(ind_En_RdPix);
82                 pixel = vec_pixels(ind_En_RdPix);
83                 vec_time_pix = [t pixel];
84                 OFF_events{ind_OFF} = vec_time_pix;
85                 ind_OFF = ind_OFF + 1;
86                 state=1;
87             end
88         end
89     else
90         ind_En_RdPix=find(EnReadPix(ind_En_RdPix:len_row_data_Sim)==0,1)+ind_En_RdPix - 1;
91         i = ind_En_RdPix;
92         state = 0;
93     end
94 end
95 cd(pwd_current)
96 % Plotting DVS response
97 plot3dDVS_fn(ON_events,OFF_events,'SIMULATED')
98 %% ========== Processing ATIS data ========== %%
99 cd(PATH_sim_output_matlab)
100 % Struct={'time','data_A','En_Read_Row_A','En_Read_pixel_A','Global_rst','Req_fr'}
101 index_desired_A = importdata(string_index_file_A);
102 len_index_A = length(index_desired_A);
103 digitalSignal_A = zeros(len_row_data_Sim,len_index_A);
104 vec_pixels = zeros(len_row_data_Sim,1);
105 time = data_Sim(:,1);
106 resol = 256;
107 Clim = [0 255];
108 cd(pwd_current)
109 % CodingGrayScale is a function that return a vector with size equal to resol
110 % with this information we can determine to classify the brightness into
```

```
% grayscale.
[vec_color, vec_COD_TIME] = CodingGrayScale(Vhigh,Vlow,resol,Clim);
% ADC converting process
for i=1:len_index_A
    signalX = data_Sim(:,index_desired_A(i)+1); % Se suma 1 para contar el vector tiempo
    index_ONE = find(signalX >= middle_point);
    index_ZERO = find(signalX <middle_point);
    signalX(index_ONE) = 1;
    signalX(index_ZERO) = 0;
    if (i <= number_bits-1)
        signalX = (2^(i-1))*signalX;
        vec_pixels = vec_pixels + signalX;
    end
    digitalSignal_A(:,i)=signalX;
end
% Defining signals of intereset
index_Threshold = number_bits;
index_En_ReadRow = number_bits+1;
index_En_ReadPixel = number_bits+2;
index_GlobalRst  = number_bits + 3;
index_Req_fr  = number_bits + 4;
time;
vec_pixels;
kindEvent = digitalSignal_A(:,index_Threshold); % Kind event Vhigh or Vlow
EnReadRow = digitalSignal_A(:,index_En_ReadRow); %en read row
EnReadPix = digitalSignal_A(:,index_En_ReadPixel); %en read pixel
GlobalRst = digitalSignal_A(:,index_GlobalRst); %global reset
Req_fr = digitalSignal_A(:,index_Req_fr); % Req_fr
% Finite state machine to brigthness encoding since simulation response
state = 0;
Matrix_time_pix_colour = zeros(1,3);
Matrix_time_high_low = zeros(N*M,2);
cross_High = zeros(N*M,2);
cross_High(:,1) = [0:N*M-1]';
Matrix_time_high_low(:,1) = [0:N*M-1]';
ind_TPC=1;
i = find(time > T_Rst,1);
while (i<=len_row_data_Sim)
    if (state == 0)
        % Verifying if EnReadPix signal rises
        ind_En_RdPix = find(EnReadPix(i:len_row_data_Sim) == 1,1) + i - 1;
        if isempty(ind_En_RdPix)
            i = len_row_data_Sim + 1;
        else
            pixel = vec_pixels(ind_En_RdPix);
            % Determining the type of crossing
            if kindEvent(ind_En_RdPix) == 0
                % Crossing by Vhigh
                t_high = time(ind_En_RdPix);
                Matrix_time_high_low(pixel+1,1) = t_high;
                state = 1;
                cross_High(pixel+1) = 1;
            else
                %Crossing by Vlow
                if cross_High(pixel+1) == 1
                    t_low = time(ind_En_RdPix);
                    Matrix_time_high_low(pixel+1,2) = t_low;
                    t_high = Matrix_time_high_low(pixel+1,1);
                    T_int = t_low - t_high;
                    ind_table = find(vec_COD_TIME <= T_int , 1);
                    Color = vec_color(ind_table);
                    Matrix_time_pix_colour(ind_TPC,1) = t_low;
                    Matrix_time_pix_colour(ind_TPC,2) = pixel;
                    Matrix_time_pix_colour(ind_TPC,3) = Color;
                    ind_TPC = ind_TPC + 1;
                    state = 1;
                    i = ind_En_RdPix;
                    cross_High(pixel+1) = 0;
                end
            end
        end
    else
        % Finding the next rising edge of En_Read_Pixel
        ind_En_RdPix = find(EnReadPix(ind_En_RdPix:len_row_data_Sim) == 0,1) + ind_En_RdPix - 1;
        i = ind_En_RdPix;
        state = 0;
    end
end
cd(pwd_current)
% Plotting the ATIS response in a bidimensional matrix
plot2dATIS(Matrix_time_pix_colour,'SIMULATED')
cd(pwd_current)
toc
```

Listing B.5: Plot DVS response in 3D

```
function [] = plot3dDVS_fn(ON_events,OFF_events,string)
```

```
% ========== Global variables ==========%
N = str2num(getenv('N'));
M = str2num(getenv('M'));
PATH_folder_images = getenv('PATH_folder_images');
cd(PATH_folder_images)
X = 0:2*N-1;
Y = 0:2*M-1;
timeScale=1e3;
struct_limsX = {[]};
struct_limsY = {[]};
% ======= Creating the labels to plots =======%
for x=1:2*N
    if rem(x,2) == 1
        struct_limsX{x} = '';
    else
        struct_limsX{x} = num2str(x/2 - 1);
    end
end
for x=1:2*M
    if rem(x,2) == 1
        struct_limsY{x} = '';
    else
        struct_limsY{x} = num2str(x/2 - 1);
    end
end
% Plotting the ON channel
z  = zeros(2*M,2*N);
len_ON_events = length(ON_events);
i = 0;
if ( len_ON_events > 1)
    fig_ON = figure('Visible','off','units','normalized');
    colormap(fig_ON,'gray')
    while i < len_ON_events
        vec_time_pix = ON_events{i+1};
        t = vec_time_pix(1);
        pixel = vec_time_pix(2);
        row = fix(pixel/N);
        col = rem(pixel,N);
        y1  = row+1;
        y2  = y1+1;
        x1  = col+1;
        x2  = x1+1;
        z(:,:) = NaN; % Avoids that Matlab create lines no desired
        z([y1 y2],[x1 x2]) = timeScale*t;
        surf(X,Y,z)
        hold on
        grid on
        i = i+1;
    end
    % adjusting apperance
    set(gca,'xtick',X);
    set(gca,'ytick',Y);
    set(gca,'Ydir','reverse')
    xlim([0 N])
    ylim([0 M])
    xlabel('COLUMNS')
    ylabel('ROWS')
    zlabel('Time ms')
    name_title = ['DVS ON EVENTS ',string];
    title(name_title)
    set(fig_ON,'PaperPositionMode','auto')
    print('-depsc2',['DVS_ON_',string,'.eps'])
    print('-dpng',['DVS_ON_',string,'.png'])
    saveas(gcf,['DVS_ON_',string],'fig');
    saveas(gcf,['DVS_ON_',string],'svg');
end
% Plotting the OFF channel
len_OFF_events = length(OFF_events);
z  = zeros(2*M,2*N);
i = 0;
if ( len_OFF_events >1)
    fig_OFF = figure('Visible','off');
    colormap(fig_OFF,'gray')
    while i < len_OFF_events
        vec_time_pix = OFF_events{i+1};
        t = vec_time_pix(1);
        pixel = vec_time_pix(2);
        row = fix(pixel/N);
        col = rem(pixel,N);
        y1  = row+1;
        y2  = y1+1;
        x1  = col+1;
        x2  = x1+1;
        z(:,:) = NaN; % Avoids that Matlab creates lines no desired
        z([y1 y2],[x1 x2]) = timeScale*t;
        surf(X,Y,z)
        hold on
        grid on
        i = i+1;
```

```
91      end
92      % Adjusting apperance and printing to file the plots
93      set(gca,'Ydir','reverse')
94      set(gca,'xtick',X);
95      set(gca,'ytick',Y);
96      xlim([0 N])
97      ylim([0 M])
98      xlabel('COLUMNS')
99      ylabel('ROWS')
100     zlabel('Time ms')
101     name_title = ['DVS OFF EVENTS ',string];
102     title(name_title)
103     set(fig_OFF,'PaperPositionMode','auto')
104     print('-depsc2',['DVS_OFF_',string,'.eps'])
105     print('-dpng',['DVS_OFF_',string,'.png'])
106     saveas(gcf,['DVS_OFF_',string],'fig');
107     saveas(gcf,['DVS_OFF_',string],'svg');
108 end
```

Listing B.6: Plot ATIS response in 2D

```
1  function [] = plot2dATIS(Matrix_time_pix_colour,string)
2  %================== Global variables ========================%
3  N = str2num(getenv('N'));
4  M = str2num(getenv('M'));
5  PATH_folder_images = getenv('PATH_folder_images');
6  Matrix2print = sortrows(Matrix_time_pix_colour,1);
7  [r c] = size(Matrix2print);
8  len_Matrix2print = r;
9  Struct_Frames = {[]};
10 vec_time_pix_colour_tmp = N*M*ones(1,3);
11 ind_struct = 1;
12 ind_Matrix_tmp = 1;
13 struct_lims = {[]};
14 %======== Creating the labels of each subfigure or frame ========%
15 for x=0:N-1
16     struct_lims{x+1} = num2str(x);
17 end
18 %=================== Building the frames ====================%
19 for i=1:len_Matrix2print
20     time   = Matrix2print(i,1);
21     pixel  = Matrix2print(i,2);
22     colour = Matrix2print(i,3);
23     if isempty(find(vec_time_pix_colour_tmp(:,2) == pixel,1))
24         vec_time_pix_colour_tmp(ind_Matrix_tmp,1) = time;
25         vec_time_pix_colour_tmp(ind_Matrix_tmp,2) = pixel;
26         vec_time_pix_colour_tmp(ind_Matrix_tmp,3) = colour;
27         ind_Matrix_tmp = ind_Matrix_tmp + 1;
28     else
29         Struct_Frames{ind_struct} = vec_time_pix_colour_tmp;
30         vec_time_pix_colour_tmp = N*M*ones(1,3);
31         ind_Matrix_tmp = 1;
32         vec_time_pix_colour_tmp(ind_Matrix_tmp,1) = time;
33         vec_time_pix_colour_tmp(ind_Matrix_tmp,2) = pixel;
34         vec_time_pix_colour_tmp(ind_Matrix_tmp,3) = colour;
35         ind_struct = ind_struct + 1;
36         ind_Matrix_tmp = ind_Matrix_tmp + 1;
37     end
38     if i == len_Matrix2print
39         Struct_Frames{ind_struct} = vec_time_pix_colour_tmp;
40     end
41 end
42 max_subfig = 16;    % Defines the maximum number of frames per plot
43 ind_subfig = 1;
44 ind_nameFig = 1;
45 % Defining the counts subfigure number to improve the visual
46 frames_maxsubfig = ceil(length(Struct_Frames)/max_subfig);
47 elements_fig = ceil(length(Struct_Frames)/frames_maxsubfig);
48 max_col = ceil(sqrt(elements_fig));
49 max_rows = max_col;
50 % Plotting each frame
51 h=figure('Visible','off','units','normalized','outerposition',[0 0 1 1]);
52 for i=1:length(Struct_Frames)
53     vec_time_pix_colour_tmp = Struct_Frames{i};
54     len_vec = length(vec_time_pix_colour_tmp(:,1));
55     Matrix_paint = zeros(M,N);
56     Matrix_paint(:,:) = NaN;
57     for j=1:len_vec
58         pixel = vec_time_pix_colour_tmp(j,2);
59         colour = vec_time_pix_colour_tmp(j,3);
60         indx = fix((pixel)/M)+1;indy = rem(pixel,N)+1;
61         Matrix_paint(indx,indy) = colour;
62     end
63     c_min = uint8(min(vec_time_pix_colour_tmp(:,3)));
64     c_max = uint8(max(vec_time_pix_colour_tmp(:,3)));
65     CMAP = uint8(unique(vec_time_pix_colour_tmp(:,3)));
66     subplot(max_rows,max_col,ind_subfig)
```

```
67      imagesc(uint8(Matrix_paint),[0 255])
68      colormap(gray)
69      if c_min ~= c_max
70          if length(CMAP) >10
71              ind_CMAP = floor(linspace(1,length(CMAP),10));
72              colorbar('Ylim',[c_min c_max],'YTick',CMAP(ind_CMAP));
73          else
74              colorbar('Ylim',[c_min c_max],'YTick',CMAP);
75          end
76      else
77          colorbar('YTick',CMAP);
78      end
79      % Finding the NaN value to Mark it. the NE label indicates which pixels
80      % had not event
81      [rows columns] = find(isnan(Matrix_paint));
82      text(columns,rows,'\color{white}NE','HorizontalAlignment','center', ...
83          'FontSize',10)
84      % Creating the title
85      title(strcat('Time = [ ',num2str(min(vec_time_pix_colour_tmp(:,1))*1e3), ...
86          ' - ', num2str(max(vec_time_pix_colour_tmp(:,1))*1e3),'] ms'))
87      % Creating lines for improving the visual
88      vc_lineX = linspace(0,N+1,200);
89      vc_lineY = ones(1,length(vc_lineX))/2;
90      for x=1:N
91          for y=1:M
92              hold on;
93              plot(vc_lineX,vc_lineY+y,'--','Color',[0.7 0.7 0.7]);
94          end
95          hold on
96          line([x+0.5 x+0.5],[0 M+1],'LineStyle','--','Color',[0.7 0.7 0.7])
97      end
98      % Changing the labels axis
99      xlabel(['Columns',' ','(',char(i+96),') '])
100     ylabel('Rows')
101     set(gca,'XTick',[1:N])
102     set(gca,'YTick',[1:M])
103     set(gca,'XTickLabel',struct_lims)
104     set(gca,'YTickLabel',struct_lims)
105     if (ind_subfig == elements_fig)
106         ind_subfig = 1;
107         cd(PATH_folder_images)
108         % Printing to file the ATIS response
109         set(gcf,'PaperPositionMode','auto')
110         print('-depsc2', ['Output_',string,'_ATIS',num2str(ind_nameFig),'.eps'])
111         print('-dpng', ['Output_',string,'_ATIS',num2str(ind_nameFig),'.png'])
112         saveas(gcf,['Output_',string,'_ATIS',num2str(ind_nameFig)],'fig');
113         saveas(gcf,['Output_',string,'_ATIS',num2str(ind_nameFig)],'svg');
114         close all;
115         if i ~= length(Struct_Frames)
116             %Avoids creating a figure without data
117             h=figure('Visible','off','units','normalized','outerposition',[0 0 1 1]);
118             ind_nameFig = ind_nameFig + 1;
119             cont_plot = 1; %Flag to indicates if is necessary plot the last fig.
120         else
121             cont_plot = 0;
122         end
123     else
124         ind_subfig = ind_subfig + 1;
125     end
126 end
127 if cont_plot
128     % Print to file the las fig
129     cd(PATH_folder_images)
130     set(gcf,'PaperPositionMode','auto')
131     print('-depsc2', ['Output_',string,'_ATIS',num2str(ind_nameFig),'.eps'])
132     print('-dpng', ['Output_',string,'_ATIS',num2str(ind_nameFig),'.png'])
133     saveas(gcf,['Output_',string,'_ATIS',num2str(ind_nameFig)],'fig');
134     saveas(gcf,['Output_',string,'_ATIS',num2str(ind_nameFig)],'svg');
135 end
```

Listing B.7: ATIS Model

```
1  close all;clc;clear all;
2  curr_pwd = pwd;
3  tic;
4  % Get the enviroment variables of simulation
5  PATH_input = getenv('PATH_folder_input');
6  PATH_folder_images = getenv('PATH_folder_images');
7  name_signal = getenv('name_Signalsinput');
8  N = str2num(getenv('N'));
9  M = str2num(getenv('M'));
10 V_p = str2num(getenv('Vdon'));
11 V_n = str2num(getenv('Vdoff'));
12 Vhigh = str2num(getenv('Vhigh'));
13 Vlow = str2num(getenv('Vlow'));
14 Vref = str2num(getenv('Vref'));
15 %% Transistor's parameters
```

```
nn = 1.334;
np = 1.369;
Vtn = 359.2e-3;
Vtp = 387e-3;
Kn = 227.1e-6;
Kp = 48.1e-6;
fi = 25.8e-3;
Vos_comp = 12e-3;          % Voltage offset comparador.
Vos_opamp = 10e-3;         % Voltage offset op-amp.
A = 20;                    % Gain closed loop from differencing circuit.
VdiffON = V_p - Vref + (Vos_comp+Vos_opamp);
VdiffOFF= V_n - Vref + (Vos_comp+Vos_opamp);
%% ========================    TCD    ============================ %%
name_input = strcat(PATH_input,name_signal,'_0.csv');
input_signal = importdata(name_input);
t = input_signal(:,1);
len_t = length(t);
quant_pixel = N*M;
Vdiff=zeros(len_t,quant_pixel);
Vdiff_ind = zeros(len_t,1); %TMP vector to save Vdiff signal of each pixel.
% Data Structures to save on and off Events
ON_events = {[]}; ON_events2TC = zeros(1,2);
OFF_events = {[]}; OFF_events2TC = zeros(1,2);
Events = {[]};
ind_ON = 1;ind_OFF = 1;
cd(PATH_input) % Goes to folder input signal.
for i=0:quant_pixel-1
    % Step 1. Finding Vdiff signal at TCD.
    name_input = strcat(name_signal,'_',num2str(i),'.csv');
    input_signal = importdata(name_input);
    Iph = input_signal(:,2);
    log_Iph = log(Iph);
    Vdiff(:,i+1) = -nn*fi*A*log_Iph;
    Vdiff_ind = Vdiff(:,i+1);
    Vdiff_max = max(Vdiff_ind);          %used to normalized the level signal
    Vdiff_ind = Vdiff_ind - Vdiff_max;  %used to normalized the level signal
    % Step 3. Finding the events produced by the pixels.
    ind_event = 1;
    Event_pix = struct;
    for j=1:len_t
        value = Vdiff_ind(j);
        if (value <= VdiffON)
            Vdiff_ind(j:len_t) = Vdiff_ind(j:len_t) + abs(value); %It sets the differencing circuit
                output up to Vref.
            vec_time_pix = [t(j)+T_Rst i];
            ON_events{ind_ON} = vec_time_pix;        % It is used to plot the TCD response
            Event_pix.value(ind_event) = t(j)+T_Rst; % It saves the instant of time when occurred a
                event at pixel i-th.
            ind_ON = ind_ON + 1;
            ind_event=ind_event+1;
        else
            if ( value >= VdiffOFF)
                Vdiff_ind(j:len_t) = Vdiff_ind(j:len_t)-abs(value); %It sets the differencing circuit
                    output up to Vref.
                vec_time_pix = [t(j)+T_Rst i];
                OFF_events{ind_OFF} = vec_time_pix;      % It is used to plot the TCD response
                Event_pix.value(ind_event) = t(j)+T_Rst; % It saves the instant of time when occurred
                    a event at pixel i-th.
                ind_OFF = ind_OFF + 1;
                ind_event=ind_event+1;
            else
                continue
            end
        end
    end
    Vdiff(:,i+1) = Vdiff_ind;
    Events{i+1} = Event_pix; % It saves
end
% Step 3. Plotting in 3D graph the TCD response.
plot3dDVS_fn(ON_events,OFF_events,'MODEL')

%% ================ Exposure Measurement ================== %%
Vint = zeros(len_t,quant_pixel);  %
C = 30e-15;
resol = 255;  % It defines the grayscale resolution i.e. 255 colors.
Clim = [0 255]; % It defines the range color.
Matrix_Color = {[]};
Matrix_time_pix_colour = zeros(1,3);
i = 0;
ack_Rst = 0;
ack_Vhigh = 0;
vec_Times_events_pixels = zeros(1,quant_pixel);
cd(curr_pwd)
% Call the function CodingGrayScaleto obtain the look-up table based on
% Vhigh and Vlow thresholds.
[vec_color, vec_COD_TIME] = CodingGrayScale(Vhigh,Vlow,resol,Clim);
cd(PATH_input)
Event_pix = struct;
ind_TPC = 1;
```

```
% Step 4. Brightness encoding cycle.
for i=0:quant_pixel-1
    name_input = strcat(name_signal,'_',num2str(i),'.csv');
    input_signal = importdata(name_input);
    t = input_signal(:,1) + T_Rst; % It is adjusted the time delay secified at electrical simulation
    Iph = input_signal(:,2);
    Vo = 0;  % Initial condition of photocurrent integration
    Event_pix = Events{i+1};
    time_events = Event_pix.value;
    ind_events = 1;
    Vint(1,i+1) = Vo;
    Color_pix = struct;
    ack_Rst = 0;
    ack_Vhigh = 0;
    for j=1:len_t-1
        if ~(isempty(find(time_events == t(j),1)))
            Vo = 1.8;
            Vint(j+1,i+1) = Vo;
            ack_Rst = 1;
        else
            % EM model.
            Vint(j+1,i+1) = -1/C*Iph(j)*(t(j+1)-t(j)) + Vint(j,i+1);
            if Vint(j+1,i+1) < 0

                Vint(j+1,i+1) = 0;
            end
            V = Vint(j+1,i+1);
            if V <= Vhigh && ack_Rst
                t_high = t(j+1);
                ack_Vhigh = 1;
                ack_Rst = 0;
            elseif V <= Vlow && ack_Vhigh
                t_low = t(j+1);
                T_int = t_low - t_high;
                ind_table = find(vec_COD_TIME <= T_int , 1);
                if isempty(ind_table)
                    % time integration was less than the minimum time at look-up table.
                    ind_table = resol;
                end
                % Assign the pixel color based on look-up table built from Spectre simulation.
                Color = vec_color(ind_table);
                Color_pix.vec_color(ind_events) = Color;
                Color_pix.vec_time(ind_events,:) = t_low;
                Matrix_time_pix_colour(ind_TPC,1) = t_low;
                Matrix_time_pix_colour(ind_TPC,2) = i;
                Matrix_time_pix_colour(ind_TPC,3) = Color;
                ind_events = ind_events + 1;
                ind_TPC = ind_TPC + 1;
                ack_Vhigh = 0;
                ack_Rst = 0;
            end
        end
    end
    Matrix_Color{i+1} = Color_pix;
end
close all;
cd(curr_pwd)
% Step 5. Plotting the 2-D graphs to verify the ATIS operation.
plot2dATIS(Matrix_time_pix_colour,'MODEL')
toc   % Returns the total elapsed time
cd(curr_pwd)
```