CMOS Imaging Technology with Embedded Early Image Processing

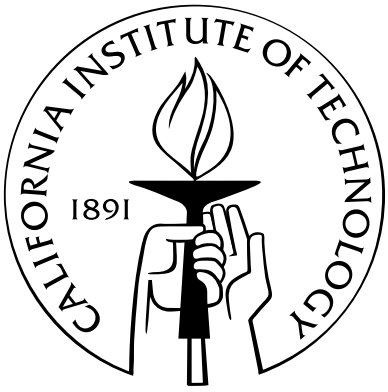
Thesis by

Christophe Jean-Michel Basset

In Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy



California Institute of Technology

Pasadena, California

2007

(Defended April 6, 2007)

A Mimi,

A Françoise, notre Rose.

Acknowledgements

I would like to start by thanking the members of my committee, who have helped and supported me at various stages of my research. Professor Pietro Perona, my academic advisor, Dr. Bedabrata Pain, my research advisor who suggested a partnership with JPL and the CMOS imaging group, Professors Ali Hajimiri, Christof Koch, Alain Martin, and Dr. Bimal Mathur who gave me valuable feedback and perspective on my work.

This adventure would not have been possible without the help of the many people whose path I crossed during my years at Caltech and at JPL. On a scientific level, I would like to express my gratitude to Guang Yang who jump-started my research by teaching me proper design techniques as well as confidence in my work and abilities. Everyone in the Active Pixel Sensors group contributed to a friendly, supportive, and scientifically stimulating environment. Thanks to my long-time officemate, Pavani Peddada, I had a friend to keep me in track when the goal seemed so far away.

When I first arrived at Caltech, I had little more than a vague idea of what was lying ahead and no funding at all. I am indebted to Don Skelton for hiring me as an assistant for his physics freshman and sophomore laboratory courses as soon as I arrived in California. This lead to a partnership with the physics department that lasted until the end of my Ph.D., with the continued help from Frank Porter who always managed to squeeze me into his budget. These years of teaching, the other instructors I worked with, and most of all my students have taught me more about science and human interaction than I ever imagined possible. When Virginio Sannibale took over the supervision of the lab, I found myself with a great supporter. With his unfailing trust and encouragements, he helped me tremendously when I needed the flexibility to organize my teaching duties around my research and my deadlines.

During such a long ordeal, it was a challenge to keep my sanity. When not playing the guitar or the flute to relieve stress, cycling was always very effective at getting rid of

excess negative energy. My time with the cycling club was lots of fun and is full of awesome memories. A special thank you goes to Pierre Moreels who accompanied me riding up countless hills and on (much) longer rides than he ever wanted to do.

Thank you to all my friends. I am blessed to have too many to name them all here. Thank you Kari for your support, kindness, love, and for proofreading this thesis. Finally, thank you to my family who encouraged me the entire time.

Abstract

As imaging technology evolves, so does the need for accurate, low-power and high-data-rate low-level image processing in a variety of computationally intensive vision applications. These applications include optical-flow computation, autonomous navigation, object avoidance or intercept, real-time target tracking, and recognition. To reach this goal, a single chip was developed, which functions as a camera able to preprocess the image in real time. It processes images through a convolution filter with a user-chosen kernel.

One of the particulars of this project is to combine the processing unit with an active pixel sensors (APS) pixel array. This complementary metal-oxide semiconductor (CMOS) technology for building imager chips allows on-focal plane signal processing, as opposed to their charge-coupled device (CCD) counterparts that need to serially output the flow of pixels to an external processing chip. The filtering can therefore be implemented as a fast, low-power analog circuit.

Convolution is achieved by matching a kernel to an image using a computation unit. The chip has an integrated imager array and a digital memory large enough to store a generic, up-loadable kernel. When recognizing or tracking a target, the uploaded kernel represents the template. Other convolution filters are implemented by setting the kernel to the set of parameters corresponding to the desired task. Filtering is performed through a column-parallel architecture of computing units, so real time computation can be achieved.

Several versions of the convolution circuit are investigated. They have been fabricated, fully tested and characterized. A number of important design changes have occurred, either to address issues that could be improved on or to experiment with alternative approaches. Timed and geometrical amplifier controls have also been investigated. By implementing image arrays of different sizes, we also demonstrate the scalability of the architecture in the spatial domain to an arbitrarily sized imager. Test results show the analog convolution chip is a viable solution for highly integrated embedded early image processing.

Contents

Chapter 1

Introduction

1.1 On-chip image processing

While high-speed imagers with varying degrees of performance are being developed [1, 2], and high speed digital processors exist, signal transfer from the imager, and processing of images at a high update rate, as required in autonomous navigation or object-avoidance scenarios, remains a challenge. Existing systems involving CCD or CMOS imager arrays combined with an external computing chip [3, 4] are limited both by the sheer volume of data, as well as by the bottleneck of transferring the data serially from the imager to the processing chip. These limitations only get worse as larger and larger imaging arrays are being released regularly on the market. It is now common to find imaging systems with well over ten million pixels. Transferring such a large amount of data for external processing demands resources capable of handling the information.

On-focal plane systems on a chip [5] benefit from fully parallel computing which simplify the interaction between neighboring pixels but at the cost of reducing greatly the fill factor of the pixels. Communication between non neighboring pixels also becomes an issue. In addition, in-pixel digital or binary systems [6] do not take advantage of the full precision of the signal from the imager, as the space restrictions for keeping a manageable pixel size do not allow the digital precision needed for good-quality imaging. Multichip and digital systems also suffer from large power consumption [2, 6], and they lack the compactness required in some embedded applications.

The new single-chip architecture, which was presented in a previous paper [7], incorporates a layer of analog early-image processing near but separate from the imager array. It is built on an active pixel sensor (APS) architecture that operates in a column-parallel

basis. Due to this semiparallel approach, the data volume and bandwidth to transfer the signals from the chip to a postprocessing unit are vastly reduced without enlarging the size of the pixels and therefore do not compromise the quality of the images. This architecture enables efficient implementation of high-quality, real-time computational imaging systems.

On-chip implementation of a general-purpose convolution filter allows identification in real time of areas of interests within the field of view without compromising signal integrity. On-focal plane integration of image preprocessing allows an efficient implementation of a variety of computationally intensive applications such as autonomous navigation, object avoidance or intercept, and recognition.

1.2 Outline

Optical flow calculation in real-time systems is a computationally intensive task, yet common in vision applications. Fast, low-level execution before the transfer of the image to an external processor alleviates the load on the processor. However, the calculations require the evaluation of spatial as well as temporal gradients which are not computed easily in hardware. An optical flow algorithm that is appropriate for hardware implementation (either analog or digital) is described in the second chapter, also with the benefits from a parallel first stage to reduce the load on digital circuits and produce a more compact design.

At the heart of the optical-flow computation is the evaluation of convolutions with known kernels. The third chapter explains the basics of convolution and why it is a costly operation in circuit development. The term "cost" is defined, and an algorithm taking advantage of the semi-parallel architecture of active pixel sensor imagers is presented. Again, the algorithm is appropriate for both analog and digital implementation. Simulations on real images are shown to illustrate the correctness of the processing.

A fully digital convolution circuit is presented in the fourth chapter. Described in Verilog and synthesized on an FPGA, it serves as a benchmark for performance and allows further validation of the algorithm in a real-time, fully operational system. A layout was generated to illustrate the size of a full-custom chip using this method. The imager is not integrated since the layout is extracted exactly from the description that was tested in an FPGA. The transfer from the pixel array to the computing circuits is a pixel-serial link. An already fabricated imaging APS chip with an integrated, on-chip analog-to-digital converter [8] is

used for the demonstration setup.

To improve on the performance and the cost of the digital implementation, an analog system on a chip is introduced which integrates an active pixel sensor with a real-time convolution system. The convolution follows the algorithm described in chapter 3. The analog circuits needed for the convolution chip are described in detail in chapter 5. This is the first stage of the design process. The main equations used to make design decisions are developed there.

The design phase of the analog convolution chip is described in chapter six. The operating conditions of each of the circuit subblocks is analyzed and the corresponding simulation results are presented. The various interfaces between subblocks, which ensure proper transmission of the information in the entire chip are explained. Influencing the design decisions are the calculations on the derivation of the main noise sources and their propagation throughout the chip. They are therefore also part of this chapter.

The layout of the chip directly impacts the performance and therefore plays an important role in the chain of events to create a circuit. The seventh chapter goes through the specifics of laying out the analog blocks, the floor plan and the choices made to ensure a compact fit as well as good operation due to proper layout techniques ensuring good matching between transistors.

The fabricated computational imager chip was tested to evaluate the performance of both the imaging capabilities and the computation units. Test techniques specific to each circuit element are shown in chapter eight. The result of the tests are also detailed as integrated test structures allow separate testing of all the independent blocks.

The analog convolution chip presented is one possible solution to computing convolution. This approach and the various other ones are summarized in the concluding chapter with their respective positive and negative aspects. Possible extensions of the work are also proposed which would take advantage of scalability properties as well as new fabrication technologies for developing vertical interconnection structures.

Chapter 2

Optical Flow for Hardware Implementation

2.1 Introduction

Determining the optical flow of a video sequence consists of extracting the changes in a series of images. It implies a system capable of finding the direction and velocity of the image at each pixel location. Optical flow determination is a common and fundamental image-processing task. It is used in a variety of vision applications. Examples include robotic vision systems [9], robotics [10], autonomous vehicle navigation, object avoidance or detection, and medical imaging [11]. However, an optical flow algorithm is not trivial to implement and is a computationally intensive tack. This problem can be approached in three different ways [12] depending on the application and the type of implementation: the frequency-based analysis [13] which extracts the frequencies of high energy, matching algorithms [14] where the distance between frame is computed, and the gradient method [14–16]. The gradient method was chosen and is described below because it can yield an algorithm that is well suited for hardware implementation.

2.2 Optical flow derivation

The calculation of the optical flow of a sequence of images requires fast determination of gradients as well as other operations. These operators are not easily implementable in a hardware architecture so they require some adaptation to produce an efficient solution. The derivation below is based on the works of Martin [15] and Horn [16] in which some of the simplifications were removed as they were not needed for this specific implementation.

The optical flow of a sequence of images is the solution to the following constraints equation:

$$\begin{cases} \nabla^2 u = \lambda^2 \left(E_x u + E_y v + E_t\right) E_x, \\ \nabla^2 v = \lambda^2 \left(E_x u + E_y v + E_t\right) E_y, \end{cases} \tag{2.1}$$

where $\begin{pmatrix} u \\ v \end{pmatrix}$ is the optical flow at the current pixel location (i, j).

The variables E_x and E_y represent the spatial gradients of the image in the x and y direction respectively and E_t is the temporal gradient. They are all also evaluated at the current pixel location(i, j).

The factor λ^2 is a smoothness factor that can be chosen depending on the specific application.

To find the optical flow, we are solving equation (2.1) for $\begin{pmatrix} u \\ v \end{pmatrix}$ at every pixel location in the image. We first need to estimate the laplacians $\nabla^2 u$ and $\nabla^2 v$ as functions of the current and previous image frames and solve for u and v.

To compute the laplacian, we introduce $\bar{u}$ and $\bar{v}$ such that:

$$\begin{cases} \nabla^2 u = k(\bar{u} - u), \\ \nabla^2 v = k(\bar{v} - v), \end{cases}$$

where $k = 3$.

The laplacian can now be approximated by applying a discrete kernel to the u and v neighborhoods. [16–18]

$$\bar{u} = \begin{pmatrix} 1/12 & 1/6 & 1/12 \\ 1/6 & 0 & 1/6 \\ 1/12 & 1/6 & 1/12 \end{pmatrix} * \begin{pmatrix} u_{i-1,j-1} & u_{i,j-1} & u_{i+1,j-1} \\ u_{i-1,j} & u_{i,j} & u_{i+1,j} \\ u_{i-1,j+1} & u_{i,j+1} & u_{i+1,j+1} \end{pmatrix}$$

$$\bar{v} = \begin{pmatrix} 1/12 & 1/6 & 1/12 \\ 1/6 & 0 & 1/6 \\ 1/12 & 1/6 & 1/12 \end{pmatrix} * \begin{pmatrix} v_{i-1,j-1} & v_{i,j-1} & v_{i+1,j-1} \\ v_{i-1,j} & v_{i,j} & v_{i+1,j} \\ v_{i-1,j+1} & v_{i,j+1} & v_{i+1,j+1} \end{pmatrix}$$

With some algebra and these variables introduced, the optical flow equation reduces to:

$$\begin{cases} u = \bar{u} - E_x \cdot \frac{E_x\bar{u}+E_y\bar{v}+E_t}{\alpha^2+E_x^2+E_y^2}, \\ v = \bar{v} - E_y \cdot \frac{E_x\bar{u}+E_y\bar{v}+E_t}{\alpha^2+E_x^2+E_y^2}, \end{cases} \tag{2.2}$$

where $\alpha^2 = \frac{k}{\lambda^2}$.

Note that to evaluate the optical flow, it is necessary to already know its value ($\bar{u}$ and $\bar{v}$ depend on u and v). The calculation is therefore an iterative process which is not ideal for real-time hardware implementation. A good approximation is the corresponding $\begin{pmatrix} u \\ v \end{pmatrix}$ from the previous frame. To give a reliable answer, the algorithm requires that the optical flow is assumed to not change much in time from one frame to the next.

To implement the above solution in hardware includes some degree of parallelism to speed up the computation without affecting the accuracy.

The steps to follow to reach a solution for equation 2.2 for two available frames are:

1. Gradient and laplacian are processed in parallel.

   (a) Gradients E_x, E_y: spatial gradients in x and y. Both nearest neighbors would be used for a second-order approximation:

$$\begin{cases} E_x = I_{1,0,0} - I_{-1,0,0}, \\ E_y = I_{0,1,0} - I_{0,-1,0}. \end{cases} \tag{2.3}$$

   E_t: temporal gradient. Only the previous frame is used. It is a first-order approximation so only one previous frame needs to be stored in memory.

$$E_t = I_{0,0,0} - I_{0,0,-1} \tag{2.4}$$

(b) The laplacian is found from the variables $\bar{u}$ and $\bar{v}$:

$$\bar{u} = \begin{pmatrix} 1/12 & 1/6 & 1/12 \\ 1/6 & 0 & 1/6 \\ 1/12 & 1/6 & 1/12 \end{pmatrix} * \begin{pmatrix} u_{i-1,j-1,-1} & u_{i,j-1,-1} & u_{i+1,j-1,-1} \\ u_{i-1,j,-1} & u_{i,j,-1} & u_{i+1,j,-1} \\ u_{i-1,j+1,-1} & u_{i,j+1,-1} & u_{i+1,j+1,-1} \end{pmatrix},$$

$$\bar{v} = \begin{pmatrix} 1/12 & 1/6 & 1/12 \\ 1/6 & 0 & 1/6 \\ 1/12 & 1/6 & 1/12 \end{pmatrix} * \begin{pmatrix} v_{i-1,j-1,-1} & v_{i,j-1,-1} & v_{i+1,j-1,-1} \\ v_{i-1,j,-1} & v_{i,j,-1} & v_{i+1,j,-1} \\ v_{i-1,j+1,-1} & v_{i,j+1,-1} & v_{i+1,j+1,-1} \end{pmatrix}. \tag{2.5}$$

With these elements, obtaining the optical flow is a straight forward process as we follow equation 2.2 to compute new values for the vector $\begin{pmatrix} u \\ v \end{pmatrix}$.

To simplify the hardware implementation, in the next steps we break the equation down and introduce several intermediate variables that eventually lead to the final result:

2.
$$\begin{cases} D = \alpha^2 + E_x^2 + E_y^2 \\ P = E_x\bar{u} + E_y\bar{v} + E_t \end{cases}$$

3.
$$\begin{cases} P_x = E_x \cdot P \\ P_y = E_y \cdot P \end{cases}$$

4.
$$\begin{cases} R_x = \frac{P_x}{D} \\ R_y = \frac{P_y}{D} \end{cases}$$

5.
$$\Rightarrow \begin{cases} u = \bar{u} - R_x \\ v = \bar{v} - R_y \end{cases}$$

2.3 Digital hardware resources

The five steps described above guarantee that each operator has valid inputs when doing its computation. All necessary variables are calculated in the previous step. Real-time timing is achieved through a pipeline architecture. The intermediate calculation for the next pixel

is done while the next operator is still working on the current pixel position.

Assuming an entirely digital implementation, the cost of the algorithm is estimated in terms of resources needed. Resources are defined as being basic digital operators such as adders or multipliers. Depending on the precision sought, they can be implemented as any size words, their complexity growing accordingly. For example, a 3×3 convolution unit is equivalent to nine multipliers and eight adders. Table 2.1 shows the step during which each operator must perform its calculation such that the flow is not perturbed and shows the cost associated with its implementation.

Use of prior knowledge of the convolution coefficients can be an efficient way to reduce the complexity of the circuit but at the cost of making any evolution or modification of the algorithm difficult or not possible. Two aspects of the convolution operators are worth noting.

The matrix operation to calculate both $\bar{u}$ and $\bar{v}$ in equation 2.5 is a standard convolution which allows for the recycling of the convolution hardware. The cost of sharing is the serialization of the process (only one operation can be done at any given time) and the need for a router to guide the data flow for the u and v parameters into the same operator. The kernel used in both convolutions are identical, so even the coefficients storage can be shared directly.

Also, in order to calculate the laplacian, the convolution kernel is fixed and can therefore be hard-wired, sparing the cost of a full, generic multiplier. The middle coefficient being "0", the operator only has to work with eight parameters for a 3×3 window. The trade-off is that no modification of the kernel would then be possible, including to change the window size to smooth the image for estimating the gradients [19] or the use of a different kernel for the laplacian. [18]

The resources described above are sufficient to go through the optical flow computation steps at one pixel location. When generalizing to an entire imager, the resources must either be duplicated for parallel or semi-parallel circuits, or reused for a serial implementation. On-chip wiring becomes significant when dealing with parallel signals. A fully parallel or column parallel circuit would require a data bus per pixel or per column which is unrealistic. A serial implementation is therefore necessary, at the cost of reduced execution speed, which is at least partly compensated by the availability of high speed operators and data transmission for digital circuits.

Table 2.1: Sequence of operators and their cost in term of hardware resources used

Step	Operation	Resources
1.	Laplacian for $\bar{u}$ and $\bar{v}$	Two 3×3 convolutions (9 multiplications and 8 additions)
1.	Gradients for E_x, E_y and E_t	Three differences (adders)
2-3.	D	Two squares and 3-term adder
2.	P	Two multiplications and 3-term adder
3.	P_x, P_y	Two multiplications
4.	R_x, R_y	Two divisions
5.	u, v	Two differences (adders)

In addition to the arithmetic operators listed in table 2.1, the cost of the circuit also includes the use of memory to store the neighboring pixels (or the partial products) for the convolution operator. The algorithm also makes use of the data from the previous frame to estimate the temporal gradient when solving for equation 2.4 so each frame must be stored in a memory for retrieval during processing of the next incoming frame. A FIFO memory of one frame size is very well suited for such purpose. The need for the nearest neighbor in both the x and y direction adds one line to the memory needs.

Similarly, the laplacian of equation 2.5 uses the results from the previous frame for both the $\bar{u}$ and $\bar{v}$ terms. Their values must be retained for an entire frame, with an extra line for the neighborhood, adding to the overall memory requirements:

$$Memory\ needed = 3 \times width \times (height + 1)\,, \tag{2.6}$$

where each memory point is an eight-, twelve- or sixteen-bit word, depending on the analog to digital converter used in the digitization of the pixels.

The first-order approximation of the temporal gradient in equation 2.4 is justified by the memory resources. A second-order approximation would use a sequence of frames, and an extra frame would have to be stored in a buffer memory, increasing significantly the

storage requirements. Large temporal changes in the image or complex video scenes would require more complicated techniques such as a multi-scale approach [20,21]. A second-order approximation would require both nearest neighbors and would therefore require two frames to be stored while introducing a full frame latency.

However, although the implementation from Martín et al. [15] uses a first-order approximation of the derivative for both temporal and spatial gradients, this rough approximation is not so beneficial when dealing with the spatial gradients since the nearest neighbors are readily available in x with a one-pixel latency and in y with a one-row latency as in equation 2.3. The added resources are one row in the memory, and the arithmetic resources are equivalent. A wider windows for the spatial gradient and laplacian results in increasing the memory to accommodate the larger neighborhood and increases the size of the convolution engine.

2.4 Analog hardware resources

The implementation of the image flow calculation in a fully digital circuit as described above suffers from several constraints that impair the circuit and can be improved on by introducing some elements of analog circuitry when they can help the performance, compactness or integration of the circuit.

Before any processing, the signals from the pixel array are analog signals (current or voltage) that can be either processed as is or digitized immediately as required in a fully digital implementation. By delaying the digitization stage, each pixel provides a single-wire interface that can be used for column-parallel readout similar to that presented in chapter 5, or fully parallel when such readout capability is available. An interesting technique for such a system is the use of three-dimensional stacked interconnections [22–24] where a vertical interchip interface is done through thinned wafer and deep via connections. In this configuration, the pixel signal can be vertically sent to a low-level processing unit independently of all other pixels in a completely parallel fashion as illustrated in figure 2.1.

Unlike the in-pixel processing approach which crowds the pixels site, the size of each pixel is not sacrificed and compact, high-fill-factor pixels are not affected. As in standard active pixel sensor technology, the readout and computation circuits are sent away from the imaging array, but thanks to the vertical readout, it is not limited to column-parallel,

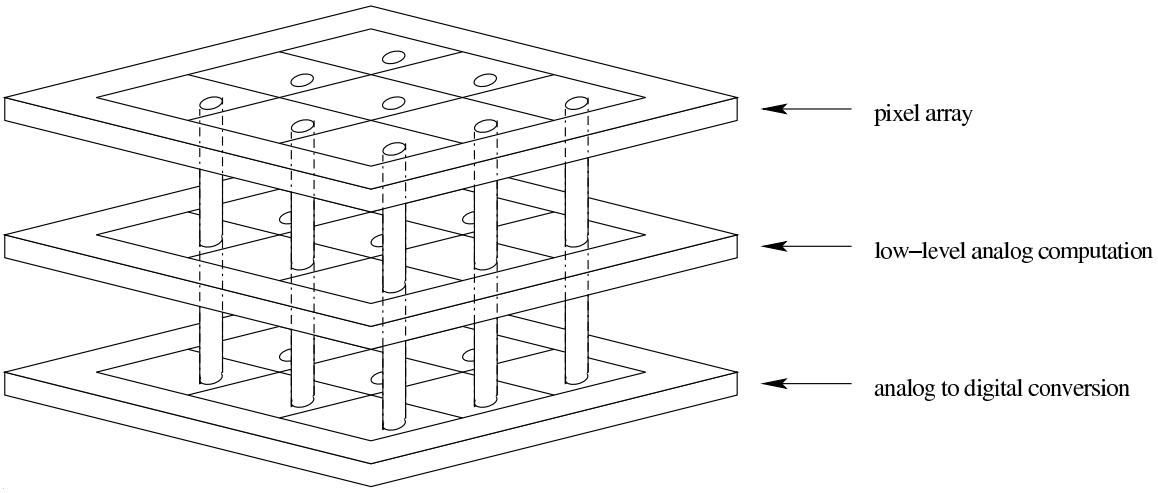


Figure 2.1: Example of a three-layer-stack interconnection: pixel array, analog transformation and analog to digital conversion interconnected with deep vias.

one row at a time access to the image, which can still be used and have the advantage of allowing large circuits without expanding the overall footprint of the chip. Computing circuits that are typically developed inside the pixel site [25–30] can be relocated to a lower layer with no modification.

Although inserting a fully parallel analog layer as the initial low-level computation does not affect the output flow-rate of the chip after digitization, it opens the door to new possibilities that eventually lead to higher performance and eventually to a faster outflow of data. The two most intuitive ways to take advantage of this are to allow insertion of extra processing on the fly and to be able to preselect pixels or regions of interest before digitizing so the serial flow of digital information only handles a smaller amount of data.

Area of the chip layout is an issue for both analog and digital circuits. The cost of producing the chip directly depends on it. It also affects the integration with other hardware where space is limited. Power consumption is also critical in autonomous systems. Comparative studies on laying out basic elements such as multiplier cells and adders [31] show that analog layout is up to forty times smaller than digital counterpart with significantly lower power consumption. The digital convolution circuit presented in chapter 4 can also be used to supplement the analysis. It uses a kernel of size 9×9 pixels, encoded with eight-bit words and was generated as a digital circuit from a Verilog description synthesized with *Mentor Graphics* tools and laid out using standard cells. The layout of its analog counterpart is

Cell	Digital area	Analog area
8-bit multiplier	$120,000\ \mu m^2$	$5740\ \mu m^2$
One-pixel memory	$15,600\ \mu m^2$	$615\ \mu m^2$
9×9 convolution	5.88 mm^2 to 16.8 mm^2	0.1 mm^2

Table 2.2: Area comparison between digital and analog cells

presented in chapter 7.

Note that only the arithmetic units to estimate the convolution are included in this estimate. The area needed to implement the memory used to buffer the previous frame also varies depending on the nature of the storage elements. A single capacitive memory cell is used to store an analog signal. The current memory cell presented in section 5.2.3.1 only occupies $615\mu m^2$ in the layout as shown in the bottom of the pixel analog readout circuit layout in 7.5. The digital memory on the other hand requires one memory cell per bit of data. A D flip-flop as used in the chip presented in the next chapters uses an edge-triggered clock to set the memory. It occupies an area of $1950\mu m^2$ on the layout. Assuming eight-bit analog to digital conversion, each pixel or basic element would be encoded using eight bits. An eight-bit word therefore occupies $8 \times 1950 = 15600\mu m^2$, which is 25 times larger than its analog counterpart. Custom layout geared toward compact optimization (e.g., using non-overlapping clocks schemes) would result in slightly smaller footprint but would not make up for such a difference.

While digital systems have the advantage to offer great flexibility, easy implementation, fast computation speeds and easy to use interface for the chip output, the introduction of an analog layer before digitization makes single chip implementation easier thanks to a compact layout and reduces the power consumed in the first stages. The comparison in term of layout area shown in table 2.2 illustrates the gain when analog cells are used instead of digital ones through the example of several of the most common cells as well as on the whole convolution circuit.

Chapter 3

Convolution

3.1 Image convolution

The convolution of two function f and g, noted $f \otimes g$, is the measure of the overlap between the two signals. In the case of images, it represents the similarity of two patches. Convolution of an image with a smaller kernel is done by repeating the basic operation on the neighborhood of all the pixels of the image. The resulting image has the same size as the original and shows the locations where the kernel is visually similar to the image. Convolution is used as a stand-alone operation in digital filters such as orientation filters, low-pass and smoothing filters, or matched filters for tracking and recognition applications. It is also used as part of a more complex computation like the optical flow estimate described in chapter 2 where it is used to calculate both the spatial and temporal gradients of sequences of images.

Let I be an image of arbitrary size. The general expression for discrete convolution [32, 33] at location (x, y) in the image I with a kernel K of size $n \times n$ is given by the expression:

$$C(x,y) = (I \otimes K)|_{x,y} = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \left(I_{x-\frac{n-1}{2}+i, y-\frac{n-1}{2}+j} \times K_{i,j} \right).$$

Another way to look at convolution is at the pixel level rather than the image level. That is, finding the transformation of each pixel in an independent step and repeat it on each pixel of the image. The concept of image coordinates (x, y) is no longer used, and the convolution core only views two small $n \times n$ image patches I and K on which it performs the sum of pixel-wise multiplications. The resulting expression (equation 3.1) is not only

simpler as it only handles a small amount of data, it is also well suited to the goal of hardware implementation. Each pixel of the convolved image being derived independently of all the others, it implicitly introduces the concept of parallel processing that will be exploited when designing the circuit.

$$I \otimes K = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} (I_{i,j} \times K_{i,j}) \tag{3.1}$$

Equation 3.1 is only a rewrite of the general expression and therefore does not affect the rendering of the output image. Its use is justified by the algorithm development which becomes more intuitive when approached at the pixel level, as detailed in section 3.2 below.

Some examples of image convolution with different kernels are shown in section 3.3 on simulation where a Matlab implementation of the algorithm described in the next section of this chapter is presented.

3.2 Algorithm

3.2.1 Overview

Implementing the convolution algorithm on a chip requires that the circuit be well suited to the system it is going to be integrated in and its operation. A hardware-oriented algorithm had to be developed that would take advantage of both the environment used (analog or digital, system on a chip or external processing system) and the interface (serial, parallel or semi-parallel) with the components of the imaging setup. What is meant by "setup" is the complete system interfacing an imager, either off the shelf or custom CMOS or CCD pixel matrix, with the convolution circuit operating in the digital or analog domain. Depending on the type of imager and mode of operation, the information from the pixels can be handled by a computation unit in three ways:

1. **Serial interface.** A serial interface providing one pixel at a time to the computation unit is used in multiple-chip systems to preserve external resources by minimizing the wiring between chips. [3,34,35] The image is extracted one pixel at a time, prohibiting any kind of parallel processing unless a frame buffer is implemented.

2. **Fully parallel.** Fully parallel architectures add computational circuits and interconnections between the pixels on the photodiode site. [5, 25–30, 36–38] The fill factor is small as most of the pixel site is filled by other elements. Still, limited space remains for interconnections in the pixel matrix so each pixel can only connect to its nearest neighbors.

3. **Semi-parallel.** Active pixel sensor (APS) imagers relocate the computation outside of the pixel matrix. Some computation or memorization can still be done inside the pixel [39]; it has the advantage of being able to keep the fill factor large while a column-parallel readout allow semi-parallel computation. Line buffers or stored partial results are used to use information from neighbors in different rows. [40–42]

The data flow in the convolution circuit is initialized in the pixel and therefore defines the interface to the first stage of the computation element. Since the final application for this project is an integrated fully custom active pixel sensor which operates in column-parallel mode, the algorithm uses this format to take advantage of that scheme rather than buffer the whole frame (or part of the frame) before starting computation as a generic processor. On the contrary, the digital implementation of the convolution chip receives a serial flow of pixels from an external imager. The incoming pixels are buffered until enough information has been read and is ready for use in the calculations. See chapter 4 for the details and specificities of the digital circuit.

In a column-parallel imager architecture, all the pixels of a row are made available at the same time to the readout circuit. They are kept valid for the entire time of the processing, until the next row is addressed. They are then replaced by the new incoming pixels which are in their turn sent to the readout circuit and the computation units. In contrast, the convolution kernel is a constant for a given frame and can be accessed at all time, allowing the realization of a pipeline architecture in the row direction.

The columns can therefore be processed in parallel, given that they are each equipped with a hardware processing chain. The diagram in figure 3.1 shows the resources needed to compute the convolution for one column. The currently processed pixel is shown in gray in the diagram. The complete convolution requires not only one pixel but the neighborhood around it. Therefore, the availability of a window centered on the addressed pixel must be guaranteed. Because it is the same system repeated for each column, only a single

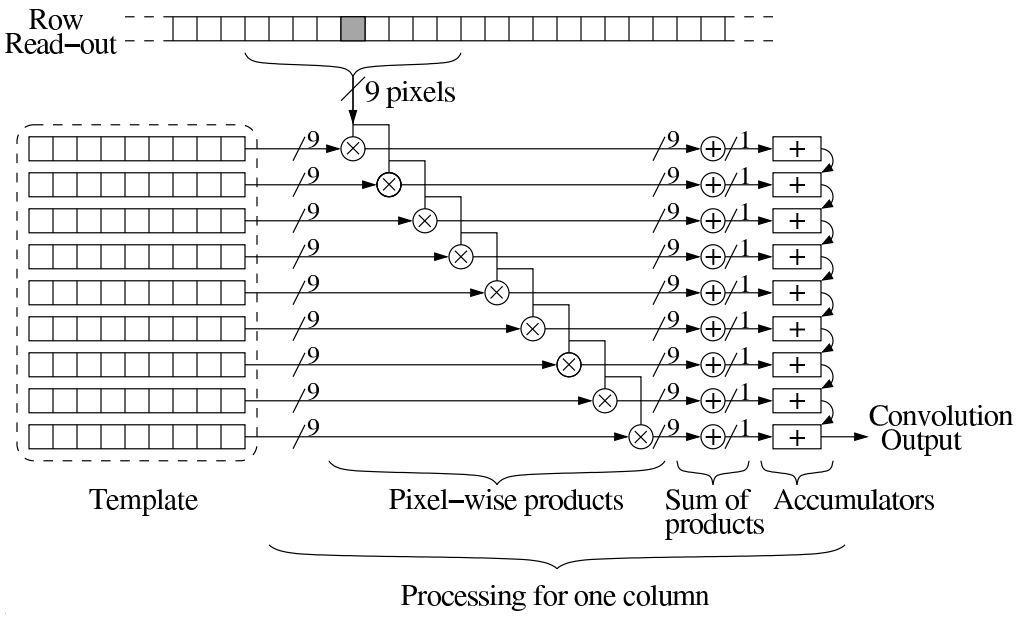


Figure 3.1: Semi-parallel architecture: convolution block diagram

convolution unit is described in detail in the next sections. Identical structures operate in parallel to compute the transformation of every column of the imager.

Neighbors on the same row (horizontal neighborhood) are all read out together along with the entire row. They are used simultaneously for the calculating the convolution in nine columns. In the event of an analog system, care must be taken to not attenuate the signals during this multiple readout sequence. The product with each row of the kernel can occur at this point as well as the inner sum of equation 3.1 which is the sum of the pixel-wise products over the same row. In the accumulators the partial results are combined with those obtained from the rows previously read-out, so the vertical neighborhood is included in the outer sum as in equation 3.1.

In summary, with each incoming row that is read out, two successive operations take place. First, the currently addressed neighborhood of pixels is combined with each row of the kernel separately through a sum of pixel-wise products. Then, the resulting partial products are combined with those from the previous rows for reconstructing the convolution using the neighborhood in the vertical direction. In this pipeline architecture, the first complete convolution is available as soon as a full neighborhood has been processed.

3.2.2 Sum of products

The first step of calculating the convolution mixes the incoming neighborhood of pixels with the entire kernel through a set of multiplications and additions. Each incoming pixel is paired with all the pixels of the kernel of the same column. In a 9×9-pixel window size, each pixel takes part in nine multiplications, yielding a product for each of the nine rows. The complete incoming group of pixels provides as many products as the kernel size.

The products obtained from the same line of the kernel are involved in the same convolution calculation and can be immediately combined by summing them right after the multiplications. This direct sum of product approach keeps memory resources lower than if all products had to be memorized before the sum over the entire window is computed. The nine remaining partial results will be used when reconstructing the convolution for the window centered on different pixels of the same column when enough rows have been read out and the fill window has been multiplied and summed in the same way but against different rows of the kernel.

During readout, the part of the image that is available (one row) is a one-dimensional array, while the convolution operation requires a two-dimensional window to be used. The convolution window is reconstructed with the addressing of subsequent rows of the image. Each row is only read once but still must be utilized for the calculation of all the convolutions in which window it appears. Each row is therefore duplicated several times and combined with each row of the kernel to generate the partial products for each position that will be used. The duplication of the pixels from the imager and the separate processing for each kernel row is shown in the diagram of figure 3.2.

3.2.3 Accumulators

At the end of the first phase, the inner sum of products of equation 3.1 is computed for the current row, the outer sum (sum over different rows) remains to be reconstructed in accumulators by combining some of these partial products with those from previous rows and saving some for use with rows still to come.

A pipeline architecture for the accumulator was designed to preserve the rate of pixels from the imager and to not introduce unnecessary delays. Therefore a real-time operation is possible. After an 8-row-time latency, the output flow rate is identical to that of the

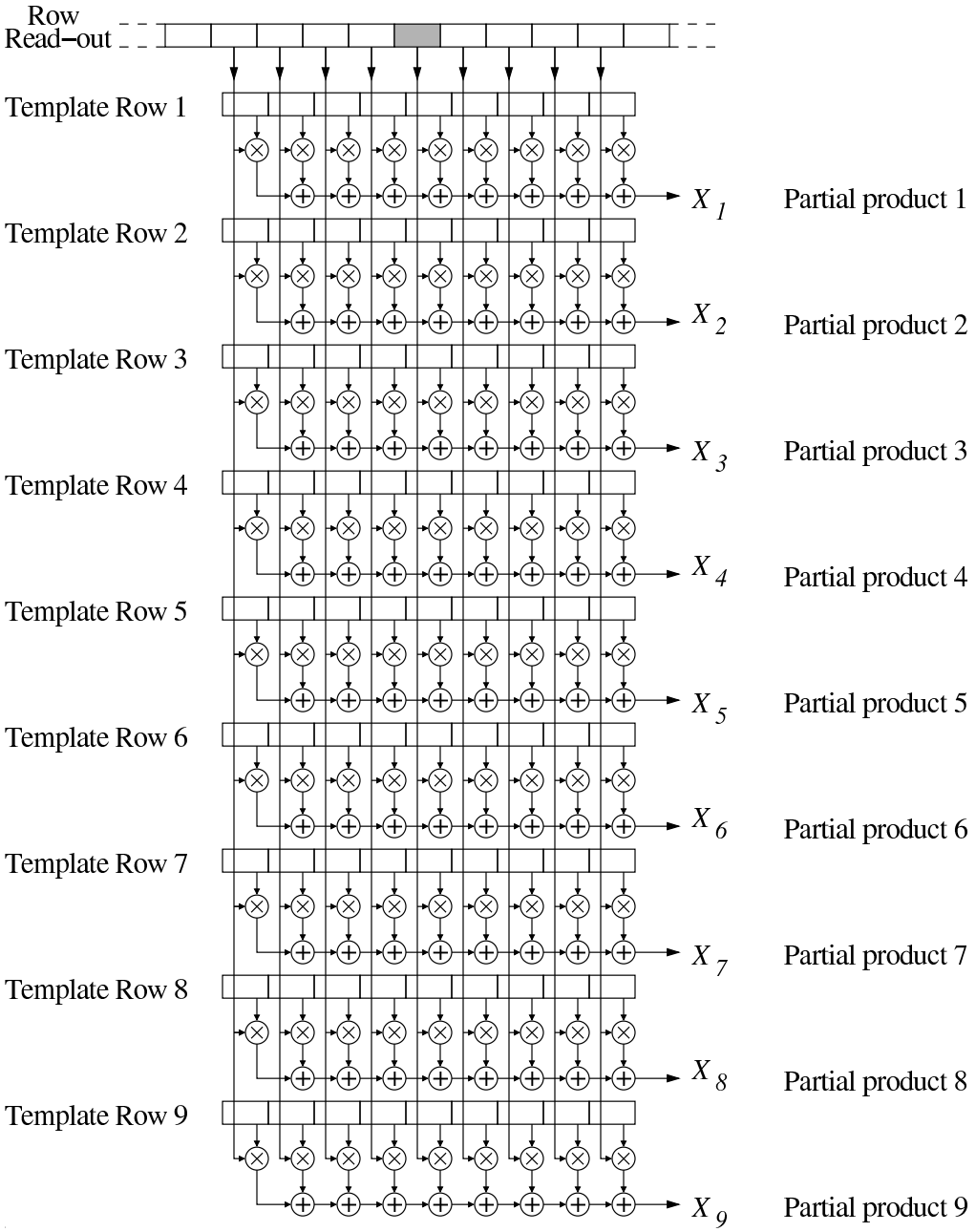


Figure 3.2: Semi-parallel architecture: convolution block diagram

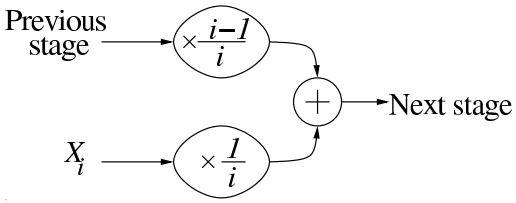


Figure 3.3: Stage i of the pipeline

imager.

Another important consideration is the need to handle a large signal range. When doing arithmetic in either the digital or the analog domain, signals are limited at both ends of the range. In the digital domain, information gets lost when small signals are rounded to a digital binary number and large values get clipped to the maximum number allowed for the allocated bit space. In the analog world, the noise level and saturation as well as non-linearity also restrict the range that can be used for transmitting information. The increase in signal strength is limited at each stage by averaging the result while preserving the same weight on each input.

Let X_j the output of the j^{th} partial product from the multiplication and inner sum stage, created when processing the j^{th} row of the window on which the convolution is performed. X_j is also input to the accumulator. Equation 3.2 summarizes the actual calculation that is performed, where n is the number of rows used. (The kernel is an $n \times n$ array.)

$$\frac{I \otimes K}{n} = \frac{1}{n} \sum_{j=1}^{n+1} X_j \tag{3.2}$$

To reconstruct this equation in a pipeline form, each stage incorporates its input from the sum of products from its corresponding kernel row into the average calculated so far. A weight correction, shown in equation 3.3, is done to preserve the equal influence of each parameter and to achieve averaging by the number of rows. When expanding it fully, the average form of equation 3.2 appears.

$$\frac{\sum_{i=1}^{j} X_i}{j} = \left(\frac{j-1}{j}\right) \cdot \underbrace{\frac{\sum_{i=1}^{j-1} X_i}{j-1}}_{from\ previous\ stage} + \left(\frac{1}{j}\right) \cdot X_j \tag{3.3}$$

Since each input comes into the pipeline at different levels, their contribution does not

ripple through the same number of stages. While the first one, labeled X_0 in the figures, is scaled and transfered through all nine elements, the last one is only processed once. Although it makes no difference in the equation, the noise and uncertainties from all the operators will not affect the various inputs in the same way. The last one will have a greater influence on the final result. The robustness of the algorithm to this asymmetry is looked into in section 3.3.2. The noise analysis of each analog computational element and propagation of errors inside the accumulator are studied in section 6.7.

The simplified block diagram of the complete pipeline, figure 3.4, uses the same variables as the sum of products of figure 3.2, which interconnects with the pipeline. These two diagrams summarize the full architecture for hardware implementation of the convolution on one column. Further details on the sub-blocks of the design depend on the type of circuit being developed. Software simulation of this system is described in section 3.3.

Actual hardware implementations of the convolution algorithm have also been fabricated and tested: a stand-alone digital system is presented in chapter 4, and an analog circuit is studied in detail in chapter 5 and following.

3.3 Simulations

High level system simulation were run to illustrate the effects of various convolution filters on images. The proposed algorithm was implemented using Matlab, rather than the embedded convolution function. This not only validated the algorithm through simulation, but also performed various tests on the robustness of the algorithm and analyzed the propagation of uncertainties from stage to stage.

In the first part of this section, the convolution of an image with various different templates is shown. No noise is added to the system, so the ideal case is portrayed, and the effect of various filters is shown. Then, random and systematic uncertainties are inserted at various stages of the algorithm simulation, and their effect on the final output is shown with emphasis on the propagation of the errors and the robustness of the system.

3.3.1 System simulation

The algorithm is shown to work through a Matlab implementation run on real images with several filter kernels. Because no noise is being added at any step of the computation, the

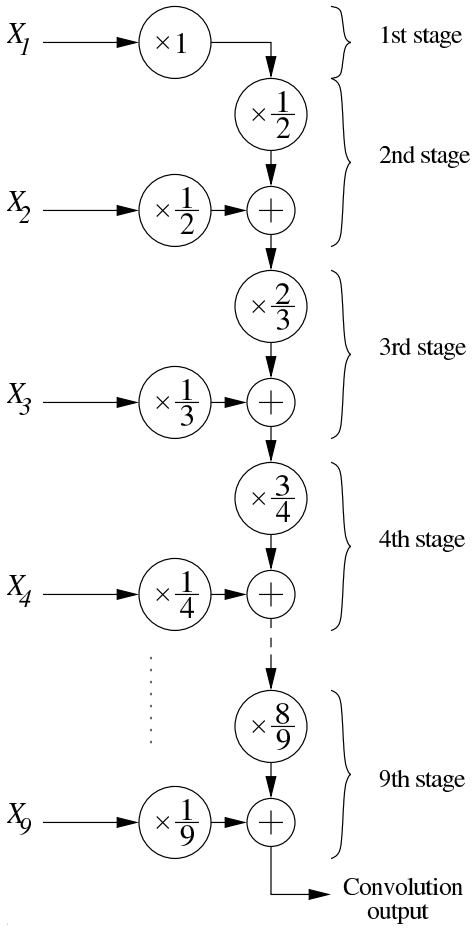


Figure 3.4: Pipeline accumulator

simulations show the transformation of the input image when applying a kernel with the convolution algorithm described in this chapter. The Matlab source code used to generate these results is shown in appendix A.

The first validation test is to verify the efficiency of the neutral element of the convolution and to obtain a filtered image identical to the input. When only one of the pixels of the kernel is non-zero, the image is not modified through convolution. It is only scaled and shifted, depending on the value and position of the remaining pixel. The identity kernel used in figure 3.5 uses the central pixel, so no shifting occurs, and its value compensates for the scaling taking place in the accumulator. The filtered image, shown in 3.5(b) is indeed identical to the original image.

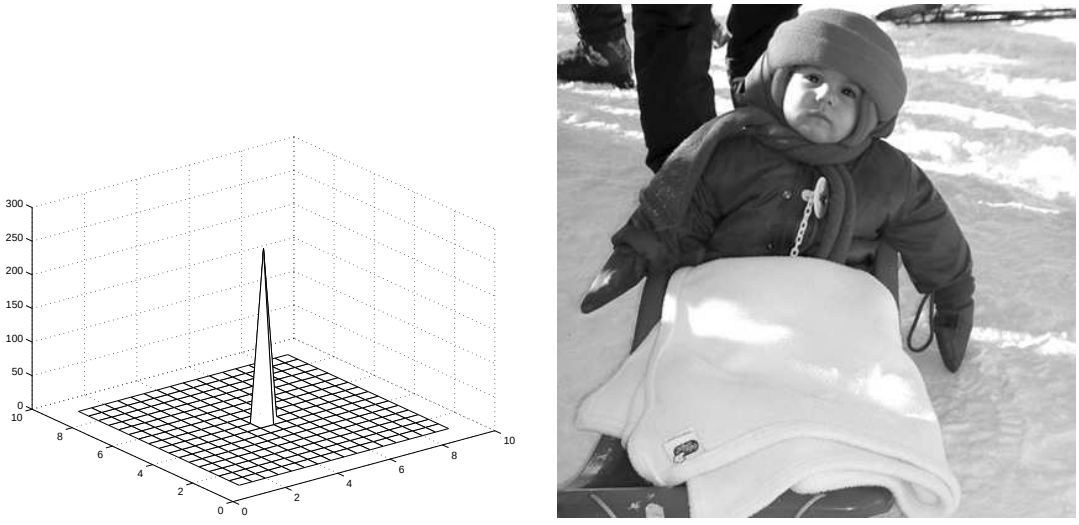
Blurring can be achieved with a uniform kernel as in figure 3.6(a) where all the pixels of a 9×9 neighborhood are averaged by the filter. Gaussian kernels allow for a more subtle smoothing of the image, depending on the standard deviation σ used to generate them. Figures 3.7 and 3.8 show examples of such smoothing with two values for σ. As the value used for σ increases, the Gaussian kernel becomes less sharp and the images get more and more blurry.

Although the hardware implementation presented in chapters 4 (programmable digital system) and 5 (analog circuit) do not use signed operators, subtraction is a minor extension to the circuits and is already planned for in the pipeline algorithm which makes no assumption regarding the sign of the kernel parameters. Two examples of filters using signed operators are shown in figures 3.9 and 3.10 with first-order derivatives in the horizontal and vertical directions.

3.3.2 Accuracy tolerance

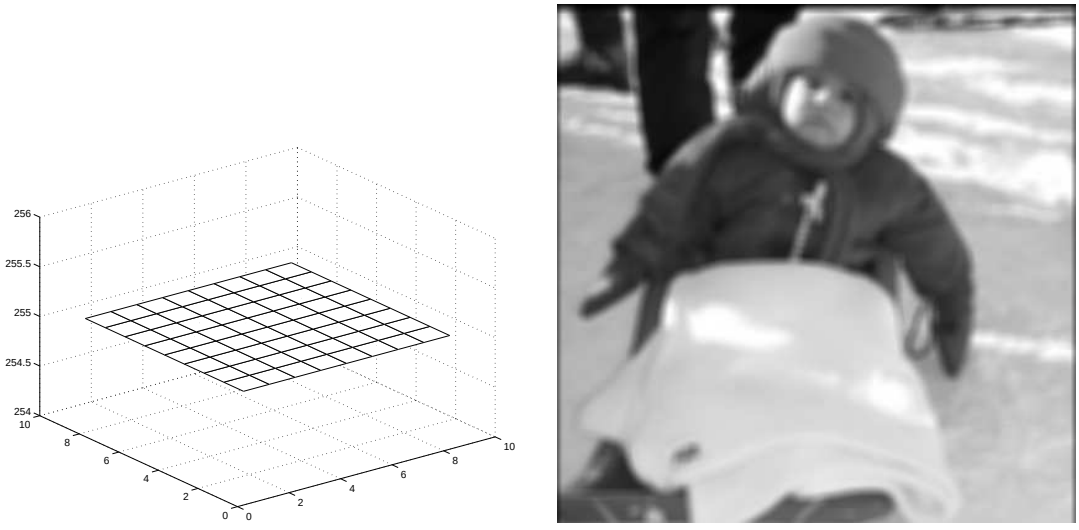
The reliability of the system depends on how well it is able to handle undesired variations from the ideal scenario. When dealing with arithmetic circuits, uncertainties may appear at any stage of the computation, so it is important to be able to predict their influence on the final results.

Digital circuits have the advantage of being predictable and unaffected by signal noise during normal operation. Unfortunately, floating point arithmetics is not easily accessible because of the complexity of implementation and the large size of the circuit. Digital operators are therefore prone to rounding errors from dividing integer operands. While



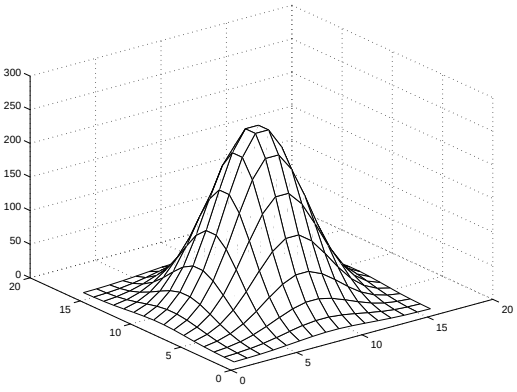
(a) Kernel (b) Output image

Figure 3.5: Identity filter: output image identical to the original



(a) Kernel (b) Output image

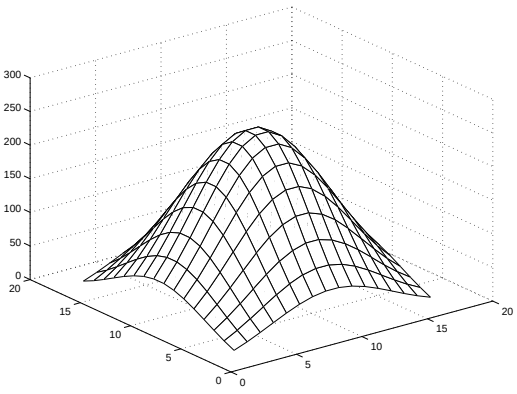
Figure 3.6: Uniform filter. (9×9 window averaging)



(a) Kernel

(b) Output image

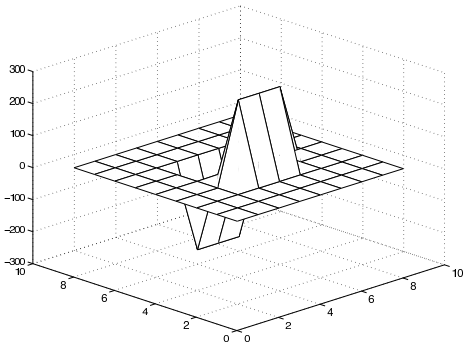
Figure 3.7: Gaussian filter: $\sigma = 1.5$ pixels



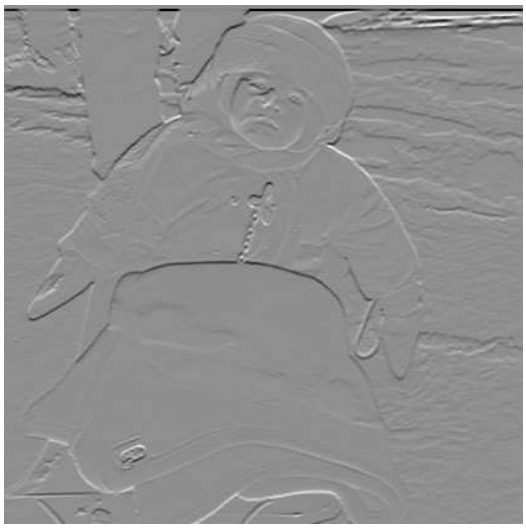
(a) Kernel

(b) Output image

Figure 3.8: Gaussian filter: $\sigma = 2.5$ pixels

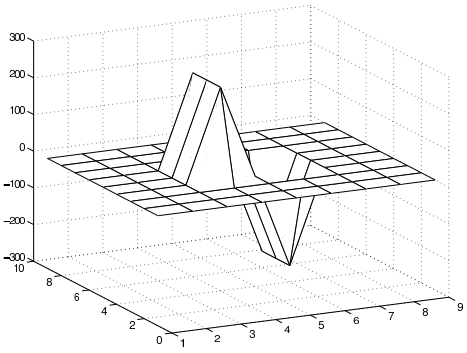


(a) Kernel

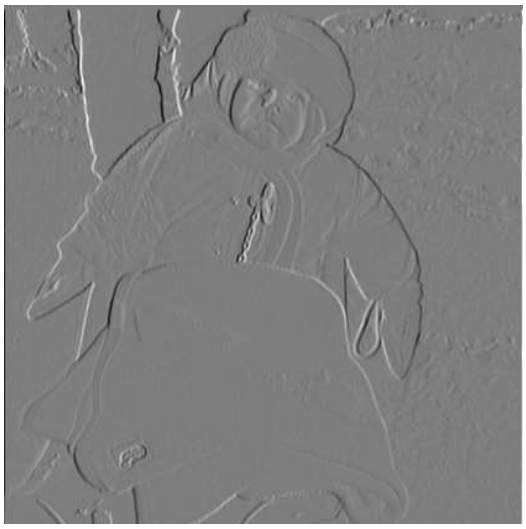


(b) Output image

Figure 3.9: Horizontal edge detector



(a) Kernel



(b) Output image

Figure 3.10: Vertical edge detector

such operators are available and commonly used in microprocessors, they are complex and yield large designs. [43–46] To minimize rounding errors, the division at each step of the accumulator from equation 3.3 is not performed until the end of the pipeline where averaging occurs for all steps at once. Overflow in the pipeline is prevented by an adequately sized internal data bus. Section 4.3 describes the trade-offs specific to the digital implementation of the convolution algorithm.

The range of signal in the analog circuit is not expendable as easily in analog circuits. Although the bus size is constant, the level has to remain inside the linear region of operation of the operators. This is where scaling at each stage becomes critical to keep the analog signal from growing into saturation. The rolling averaging of equation 3.3 solves this problem but amplifies the asymmetry of the error contribution inside the pipeline accumulator.

A full analysis of the noise in analog circuit and how it propagates through the pipeline is presented for the convolution chip in section 6.7.2.

Chapter 4

Digital Stand-Alone Implementation

4.1 Introduction

A general-purpose filter can be built on a variety of circuits, each tailored to specific applications. A fully analog design is presented in the next chapters that incorporates the filter on the imager chip. An implementation of the same algorithm onto a fully digital circuit uses well-defined elements that are easy to set up. It provides a reliable testbench for validation of the algorithm as well as grounds for comparison of the various systems.

Microprocessor-based computers allow fast integration that is valuable for test and validation of the algorithm. With the use of a high-level programming language, they are easy to program, fully reconfigurable and operate at speeds that make them the best choice for many applications. Because of their physical size, however, they are ill suited for miniature environments and cannot meet power restrictions of many embedded systems.

The more practical implementations for compact, real-time image processing units call for more specialized components. Digital signal processors have been a platform of choice for many imaging systems. [47, 48] As general-purpose processors, they are also programmable and only require software-level development, making their turn around and reconfiguration very fast. They are also often equipped with specialized interfaces for data transfer which the full microprocessors lack. This is vital in real-time image processing due to the amount of data that needs to be transferred to and from the DSP.

Dedicated circuits can only perform one task and are the most difficult to work with. They are versatile in the sense that there are no restrictions in term of what function

can be implemented and no resource limitations, but unlike processors they cannot be programmed or reconfigured easily. The development time is much longer and the price much greater. The finished product is however the best tuned for the application since it carries no overhead or unused circuitry. Algorithms for convolution on digital VLSI chips have been developed and tuned using pipeline architectures, parallel computations or systolic arrays. [49, 50]

Field programmable gate arrays offer an attractive trade-off between cost, ease of development and speed. Although they are made of predefined logic cells, they have few high-level functions and do not suffer from large overhead as DSPs and micro-processors do. Since they are designed in a similar way to full-custom circuits, they are a platform of choice for research and development before fabrication of integrated circuits. This is why algorithms for FPGAs have been studied extensively, either as proofs of concept or as final products. [35, 51–54]

Digital designs using standard CMOS logic can be implemented on the imager chip which uses the same technology. However, the test setup being developed would not be actually fabricated on an integrated circuit. The design is therefore made using the Verilog hardware description language which is first simulated as a behavioral model. It is then compiled and uploaded onto a reconfigurable FPGA for fast and efficient validation of the code and of the algorithm in a real test environment. The final step for a full-custom circuit is to synthesize the Verilog program to produce a full layout of the chip, ready for fabrication. Each step of the design flow adds new information that is used in simulation such as timing restrictions and resources allocated. While the FPGA implementation validates the accuracy of the algorithm, it is the final layout that shows the actual resources used if it were to be fabricated.

An FPGA digital implementation was chosen to perform the algorithm comparison. For testing purposes, an existing, available imager chip with a 512×512-pixel array equipped with an on-chip 8-bit analog to digital converter [8] was wired to a Xilinx Spartan-3 FPGA. The output of the FPGA was in turn interfaced to a digital data acquisition system for visualization. This simulates the interaction of the imaging array with the computing unit, the main operating difference being that the pixels coming from the imager have to be sent in series, preventing parallel processing.

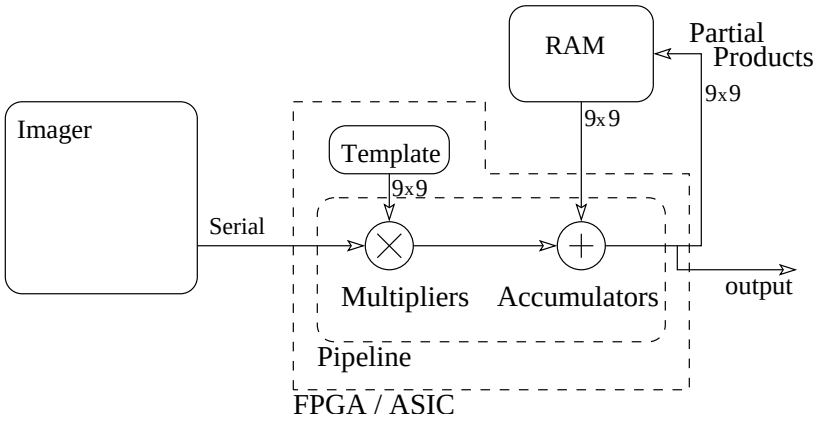


Figure 4.1: Digital implementation block diagram

4.2 Algorithm

The convolution algorithm described in section 3.2 was developed for the purpose of hardware implementation. No assumptions regarding readily available building blocks were made and the drive was to minimize the cost of the algorithm in terms of resources while maximizing the parallelism of the data flow. The design process also did not assume which type of hardware implementation would be used. As a first approximation, the same factors have to be taken into account whether a digital circuit or an analog circuit is to be produced and the same trade-offs have to be considered in both cases. Therefore the same algorithm is implemented in both circuits. The specific implementation details of each approach do vary however. The trade-offs to be considered are different and will be discussed separately in section 4.3.

The parallelism capability is an important part of on-focal plane designs. An on-focal plane digital architecture with an analog to digital converter embedded in each column would preserve this capability but it is lost when creating a two-chip system as done here. The test setup should not however hide the design goal which is still an integrated circuit, so it is important to preserve the structure of the architecture. This way, the same algorithm can be applied and the produced circuit and layout are a realistic implementation of the fast, fully integrated system. The imager sends pixels serially to its output interface. A line buffer was implemented on the receiving end so the data of the full 9-pixel width of the kernel is always available. It is completed by another 8-line buffer so the nine-line buffer used in the analog chip to store partial products of the convolution is also accurately

rendered.

The resources needed are not limited to the memory cells used to store the partial products which grows linearly as a function of the imager width and the kernel height, but also include arithmetic units for the actual convolution to take place. For each column computing unit, a choice had to be made between a fully parallel structure or a semi-parallel one. With the fully parallel approach, as many multipliers as there are pixels in the kernel are necessary. (81 8-bit multipliers for a 9×9-pixel kernel) This is the most efficient design in terms of speed of execution but expectedly produces large layouts due to the redundancy of the hardware produced. In the semi-parallel implementation, the resources are re-used constantly in the processing of the columns. It is the width of the kernel only that determines the number of multipliers, not the product of the width by the length. Only one multiplier per column in the kernel is needed and provides nine partial products for each convolution computation.

Both options implement the same algorithm and therefore share the same structure as shown on the block diagram, figure 4.1. The multipliers and accumulators are wrapped in a pipeline implemented as a state machine. The difference between the two implementations are in the number of multipliers used and the latency introduced to ensure availability of the products at the stage they are needed in the pipeline. The sequence of events is common to both implementations and is made of four steps that are repeated for each pixel provided by the imager.

Multiplications. $\sum_{row}(K \cdot I)$. The pixel neighborhood is multiplied pixel-wise with each row of the kernel. The results of the same rows are added to each other, producing nine sums of products that will each be used in a stage of the accumulators.

Read from RAM. The contribution from the previously calculated partial products that correspond to the accessed column is needed so they need to be extracted from the memory.

Accumulation. Once both inputs of the accumulators are available, they are processed and the pipeline running average is executed. A new set of partial products is calculated, the last stage (the output of the pipeline) is made available for readout as it is the complete convolution for the current neighborhood.

Write to RAM. The output of the accumulator is sent back to the RAM so it can be stored for use when the next column is addressed again with the pixels from the next row. The last row is not saved as it is already complete and will not be needed again.

4.3 Trade-offs

Decisions during the implementation process of the circuit are made at every stage of the design. A recurring issue touching digital computational units is the width of the data bus that needs to grow with each operation to avoid overflows and loss of information. A worst case scenario approach was taken for the internal buses so the width at the output of each multiplier or adder can accommodate any result. An internal bus of up to 20 bits guarantees that no overflow should occur during normal operation. The input range of 8 bits is restored for the output by post-computation division. Although it would keep the bus width constant and largely reduce the internal wiring, scaling is not done on the fly to avoid handling small numbers that can round to zero when using integer-based operators rather than floating-point divisors. On the algorithm level, the averaging factor of equation 3.2 is not done at each step of the accumulator as shown in the block diagram of figure 3.4 but rather at once after the complete accumulation has occurred. This is the only major difference with the analog implementation which required the signal range to remain in the linear region for every element, as is described in chapter 6 on the design requirements.

The area reduction by using a semi-parallel architecture consists in not implementing all 81 multiplier cells needed for the 9×9-pixel kernel. Instead, only one row of such multipliers are implemented, and each is used 9 times to compensate for the lack of resources. A pipeline setup maintains the high throughput, but a latency is introduced at the beginning of each pixel computation. This design reducing technique can be enforced further by only using one multiplier cell used 81 times for each incoming pixel. The trade-off to consider is that between resources used (which translates directly in silicon area when fabricating a full-custom ASIC) and computation time. Because less cells operate at once, the power is reduced when decreasing the number of multipliers, but the energy consumed during the full convolution operation remains the same, as all 81 multiplications are eventually performed.

It is worth noting that the discussion so far has focused on the resources needed for implementing the computation cells for one column. A fully parallel architecture would

Frame rate (Hz)	Row Clk (kHz)	Pixel Clk (MHz)	Digital Circuit Clock: Col-parallel (kHz)	Digital Circuit Clock: Full Circuit (MHz)	Digital Circuit Clock: 16-column Blocks (MHz)
1	1	1	30	30	0.48
10	10	10	300	300	4.80
30	30	30	900	900	14.4
60	60	60	1800	1800	28.8
100	100	100	3000	3000	48.0

Table 4.1: Clocks for a megapixel imager (1024 rows and 1024 columns): full circuit with one convolution unit per column, one for the whole chip and one per block of 16 columns.

duplicate these resources into each column and obtain a much more efficient design in terms of speed of operation. The size of the circuit, however, becomes unmanageable as the whole arithmetic unit (which includes the multipliers as well as the accumulators) is multiplied by the number of columns in the imager.

The operating speed of the digital circuit does not justify the use of so much hardware. The flow of pixel from the imager is regulated by a pixel clock which is itself a function of the frame rate: $CLK_{pix} = \frac{CLK_{row}}{height} = \frac{CLK_{frame}}{(width \times height)}$. The frame rate can easily be changed but for practical reasons, only small variations are possible. When the frame time decreases, less light is integrated in the imager so the image gets darker and the quality is affected. It can be compensated by increasing the light level and opening the aperture, but only on a small scale.

The information summarized in table 4.1 shows the two extreme cases and the suggested trade-off in terms of operating speed of the digital circuit. The data shown assume a 1k×1k-pixel imager is used and attached to the convolution units as described here while operating at various frame rates. Processing each pixel requires eight memory accesses in read mode, and an extra 13 cycles delay for the pipeline dividers and nine memory accesses in write mode. The total of 30 cycles is therefore the minimum computation time for this algorithm. The memory modules used in the prototype use a two-cycle read and write scheme that brings the computation time to 54 cycles. The optimal case of 30 cycles is used as a reference in table 4.1 as one-cycle memories are readily available.

The fully column-parallel architecture expectedly allows high frame rate while keeping a low internal clock for the digital circuits (up to $3MHz$). It is there that the waste of

resources becomes evident. As discussed above, digital circuits can operate at much higher speeds and such resource allocation is unnecessary. The other extreme consists in only implementing one convolution unit for the entire imager. In this configuration, the only cell must be used by every pixel of each row. The system becomes fully serial and must the speed the digital circuit must operate at just to keep up with the frame rate grows dramatically as we reach normal operating frequencies. Operating speeds of $30fps$ to $100fps$ are very common and require the digital circuit to run with a clock of $900MHz$ to $3GHz$ which is not practical for this type of circuit. As a reference, currently available FPGAs only allow up to a few hundred mega-hertz. More specifically, the Spartan-3 chip for Xilinx that is used in the prototype is rated for an internal maximum speed of $300MHz$. Digital signal processing chips can operate at a faster rate but would eventually face the same issue with fast frame rates ($\geq$$100fps$) or larger imager arrays.

The suggested trade-off for hardware implementation is to use a block-parallel architecture where the output of the imager is split. Several convolution units are implemented, each in charge of handling the data from 16 columns. Each convolution units operate in parallel, and the serialization only includes 16 pixels. The operating frequency remains in an easily manageable range for normal imaging operation while using significantly less resources.

4.4 Layout

A description of a single convolution unit was written in the Verilog HDL so a prototype could be built. The program was synthesized with the Xilinx library to produce a working setup on a Spartan-3 FPGA and also synthesized with the Mentor Graphics tools to generate a netlist and a standard cells-based layout. The FPGA uses predefined cells that perform a function when properly interconnected. The resulting implementation is sub-optimal and only serves the purpose of validating the code and the algorithm while quickly producing a working prototype. The netlist and layout, however, give a more realistic estimate of the resources needed in terms of number of logical gates and flip-flops and in terms of silicon area for fabrication.

Due to the limited resources of the FPGA, only one convolution cell was described in Verilog. Although this approach does not allow high-speed processing of the images, it

validates both the algorithm and the digital implementation approach. It was discussed in section 4.3 on the trade-offs faced during the design phase that the possibility of parallelism is important for the viability of the system. This property is preserved with the designed cell as it is trivial to add more processing units in parallel in this configuration: the same computation cell needs to be arrayed and wired to various sub-sections of the imager. This re-wiring is, of course, only possible in the case of a system on a chip where the imager and the processor are on the same chip. In the case of multi-chip systems in which the imager provides the pixels as a serial flow of data, a pipeline is needed with a controller so processing starts on the first pixels of the row before the whole row is made available.

4.4.1 Memory

The memory requirements discussed above already included all that is needed for full frame processing and therefore do not change when parallel processing is used. However, the organization of the memory changes slightly. It is split in separate blocks wired to the various computing units, as each of these units needs to address its corresponding memory space simultaneously.

In an FPGA, basic operators (such as flip-flops, full adders, and multipliers) are already implemented as elementary building blocks and need not be redefined. The design makes extensive use of these blocks as they are optimized for the device. The same goes for the random access memory for which a core is available for on-chip implementation. Another option, had the on-chip memory capacity not been sufficient, was to use an external memory chip. In both cases, the Verilog description focuses on the computation and assumes a proper memory already instantiated with a known interface as in figure 4.2.

The kernel however, is significantly smaller (648 bits as opposed to 256 Kbits) and can easily be implemented as a single register. By connecting the most significant bit to a pad, the register can be initialized serially as a FIFO while providing random access to the saved data. This register is unique for the whole chip and need not be replicated in each instance of the circuit in the case of parallel processing. It is therefore not part of the actual convolution cell which is meant to be fully arrayable.

The interface between the kernel and the convolution unit is a fully parallel connection, which is easily done internally but would require as many input/output pads as bits in the kernel to use an external memory. Because of the limited number of pads in the device, the

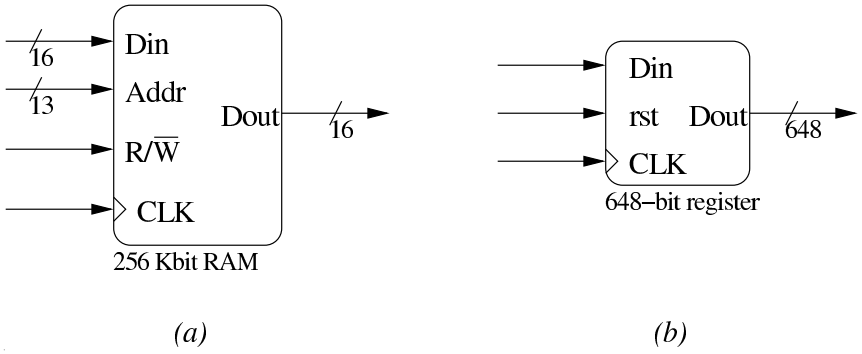


Figure 4.2: Memory interfaces. (a) Partial products RAM and (b) kernel serial-in / parallel-out register

kernel has to connect to the convolution unit internally. Only the serial interface used to load and change the kernel makes use of the pads (clock, serial data line and reset signals).

The Verilog description used to produce a full custom ASIC does not include any shared elements, so the arrayability of the convolution cell is not compromised. The partial products are stored in a RAM module that can be implemented on the same chip with one of the well-defined memory designs available. [55]

The final layout is made of three interfaced blocks: the kernel memory, the partial products memory and the convolution cell.

4.4.2 Convolution unit

The single convolution unit is the building block of the circuit guaranteeing scalability for any size imager. Like its analog counterpart presented in chapter 5, it is meant to be integrated with an imager, along with the memory cells in the same chip. The block-diagram of figure 4.1 shows how the convolution unit integrated in either a FPGA or an ASIC fits into the complete system. The resources used in the layout of this cell and its physical size are the only growing elements of the computation part of the design when modifying the imager size and the amount of data to process. It therefore represents the factor by which the complexity of the circuit increases linearly and is consequently the cell for which a layout was generated.

Automatic layout generation uses a selection of standard cells already available for the process that is used as a reference. For this design, we selected a process with $0.5\mu m$ minimum feature size, three metal layers and two polysilicon layers. It is similar to the

one used for laying out the analog chip, so area comparison between the two methods are relevant.

A custom standard cells library was used to create the layout. It includes layout and simulation models for all the usual logic gates: two-, three- and four-input nand and nor, xor, inverters, buffers, tri-states as well as D-type flip-flops, latches, multiplexers, etc. These allow the generation of all logical and computational elements from the Verilog source code. A clear improvement on this library, useful for this design, would have been optimized multipliers. Such cells are available in the FPGAs and would improve the ASIC in terms of area and timing performance.

To illustrate one of the trade-offs pointed out in section 4.3, two versions of the layout were produced. It was shown there that with a small increase of the computation time, the area of the laid-out circuit could be greatly reduced. In addition to quantifying the resources necessary for each implementation, the two generated layouts illustrate the improvement.

The Verilog code used to produce the prototype implemented in a FPGA was also used for the layout. Although different synthesis tools had to be used, introducing various timing uncertainties in the design, the two designs are guaranteed to be functionally identical. Still, post-routing simulation of the final layout is necessary before fabrication to ascertain all the timing requirements are met as well when using the standard cells.

A size comparison is shown figure 4.3 where both complete layouts are displayed on the same scale.

4.5 Testing

The Verilog code used for the synthesis to generate the layout used in the analysis above, was also used to create a working prototype of a real-time convolution imaging system. An existing imaging APS chip with integrated analog to digital converters was interfaced to a Xilinx Spartan-3 FPGA evaluation board on which the synthesized Verilog was uploaded. The transformed images could then be displayed on a computer monitor through a digital acquisition system.

Unlike the block diagram of figure 4.1, no external RAM is used. The internal memory space of the FPGA is sufficiently large to accommodate the memory needs to store the convolution partial products. Instead, a simple two-stage setup was built with only the

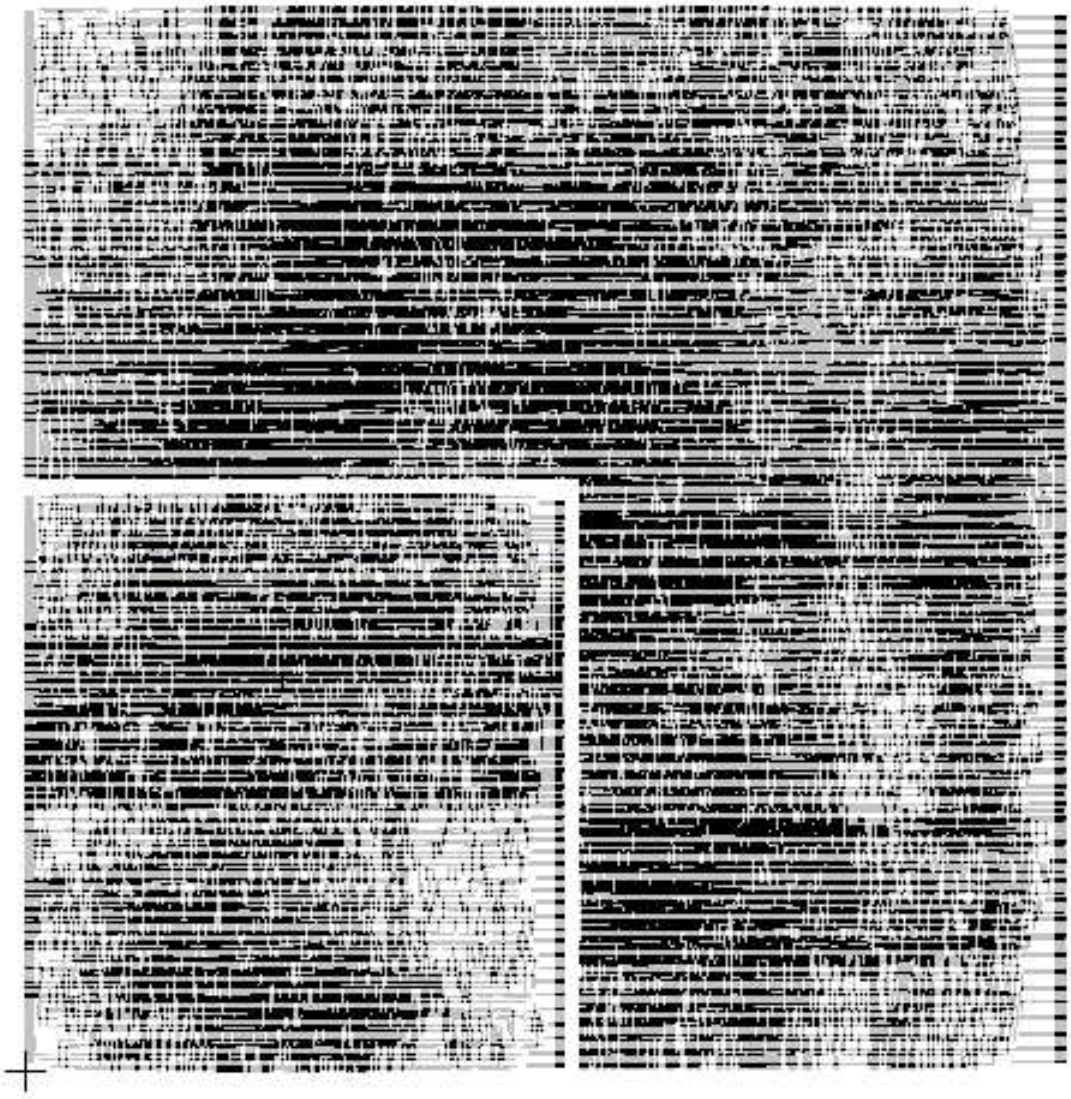


Figure 4.3: Layout trade-off: 9-multiplier implementation ($2.35mm \times 2.5mm$) and 81-multiplier implementation ($4.2mm \times 4.0mm$)

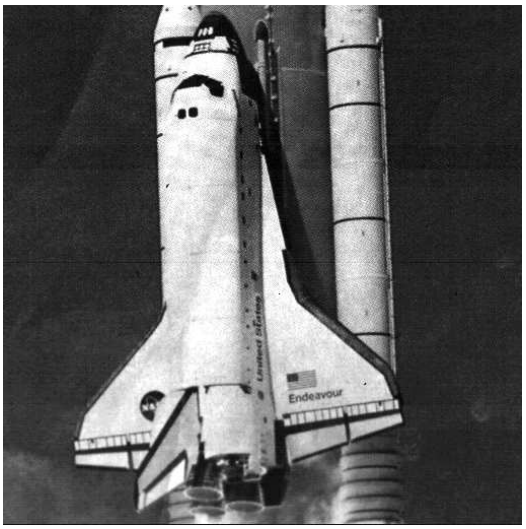
imager custom interface board and the FPGA off the shelf evaluation board.

The imager chosen for the task was an already available APS chip with a 512×512 pixel image sensor and an integrated 10-bit analog to digital converter [8]. Since the computing circuit expects 8-bit digital encoding of the pixels, the precision of the ADC was reduced to eight bits.

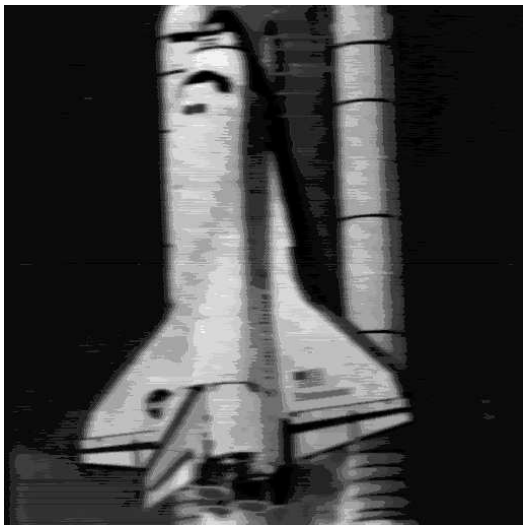
Real-time video rate of 30fps was achieved, demonstrating the effectiveness of the algorithm. Images taken with kernels encoding various types of filter are shown as examples below. For the test, kernels similar to those used in the software simulations of figures 3.5 to 3.8 were used on a poster filmed by the camera while the output is acquired in real time.

Low-pass filters are shown along with the original image in figure 4.4. The granularity of the unfiltered image (a) is an artifact of the printed image used as a target. It helps to show the smoothing that occurs when flat or Gaussian kernels are applied: the granularity disappears, leaving a smooth background instead.

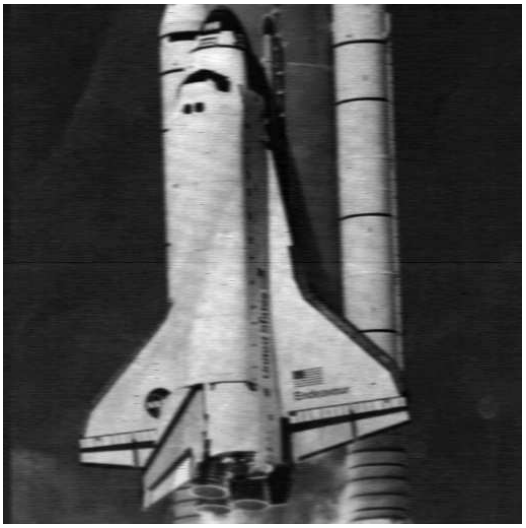
Figure 4.5 uses a kernel with a single non-zero element in it. It effectively encodes a gain in the image with no distortion. The differences seen between the original image and image (a) especially in the background, illustrates rounding errors and clipping in the division operations mentioned above in section 4.3. When too large of a gain is applied, the opposite problem occurs and saturation appears in the bright areas, as shown in (b). In this event, the dark background still suffers from division rounding errors so the resulting image looks binary as the gain increases and the bright pixels saturate.



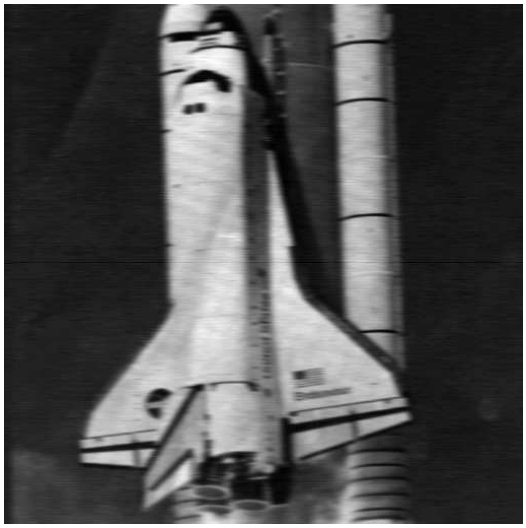
(a) Original image



(b) Flat kernel

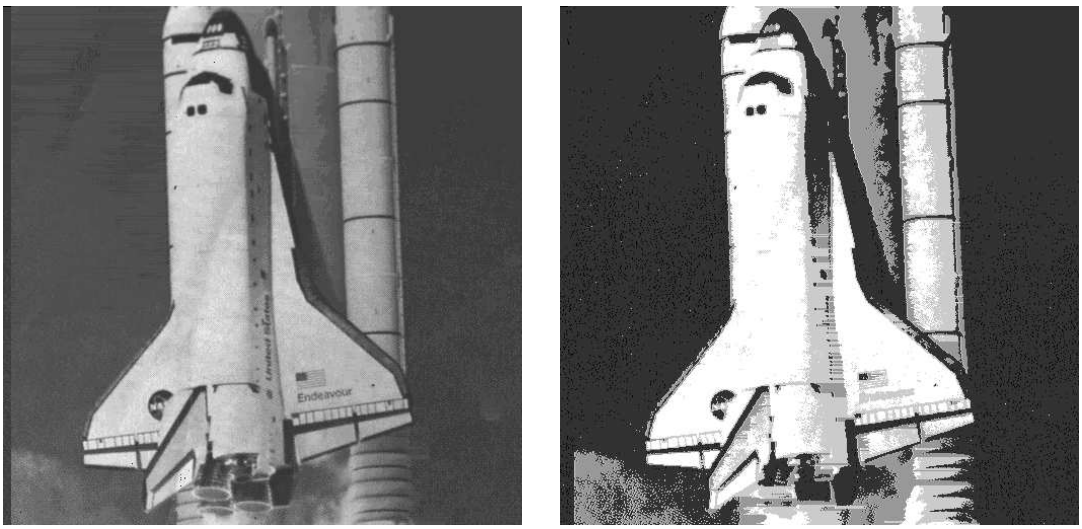


(c) Gaussian $\sigma = 1.5$ pixels



(d) Gaussian $\sigma = 2.5$ pixels

Figure 4.4: Original image and filter responses



(a) Gain with no distortion

(b) Gain inducing saturation

Figure 4.5: Gain applied to the image with no distortion (a) and saturating the output (b)

Chapter 5

On-Focal Plane Implementation: Current-Mode Computational Imager

5.1 Introduction

Several version of the convolution chip were designed, based on different principles for implementing the analog arithmetic operators (adders, multipliers, memory cells, etc.) necessary for calculating the convolution. In the first version, the charge-mode convolution chip, a current-based pixel array was implemented. The arithmetic operators were relying on transfer of charges to create voltage levels proportional to the operation result. For example, accumulation was achieved by flowing a current into a capacitor for an equal period for each input. The second version, presented in this chapter, manipulates current flows for the calculations.

The charge-mode convolution chip was fabricated and tested. It gave encouraging results for the computation but also pointed out some issues that needed to be addressed to improve the image quality and to develop a seamless interaction between the entire pixel matrix and the convolution core.

In the next generation of the circuit, which was fabricated and tested, the overall concept remains the same. However, the actual design was greatly modified to take into account the results of the previous version. Since the multiplying DAC showed accurate calculations, it was used again with only minor modifications. The accumulators, although they also performed well, were entirely redesigned to save space on the chip (large capacitor banks and their triangular structure required some awkward re-arrangement to be scaled up and

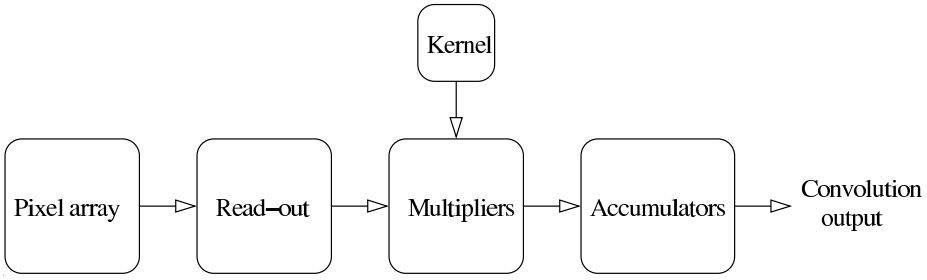


Figure 5.1: Signal flow block diagram

accommodate a full-imager convolution) and to test another approach that had shown encouraging results in simulations.

The purpose of this second generation chip was twofold. First, to demonstrate the ease or arraying the various elements so that the entire imager could be scanned at every frame and the convolution performed in a semi-parallel fashion. Second, to integrate a new convolution accumulation algorithm, saving space on chip and improving the accuracy of the calculations.

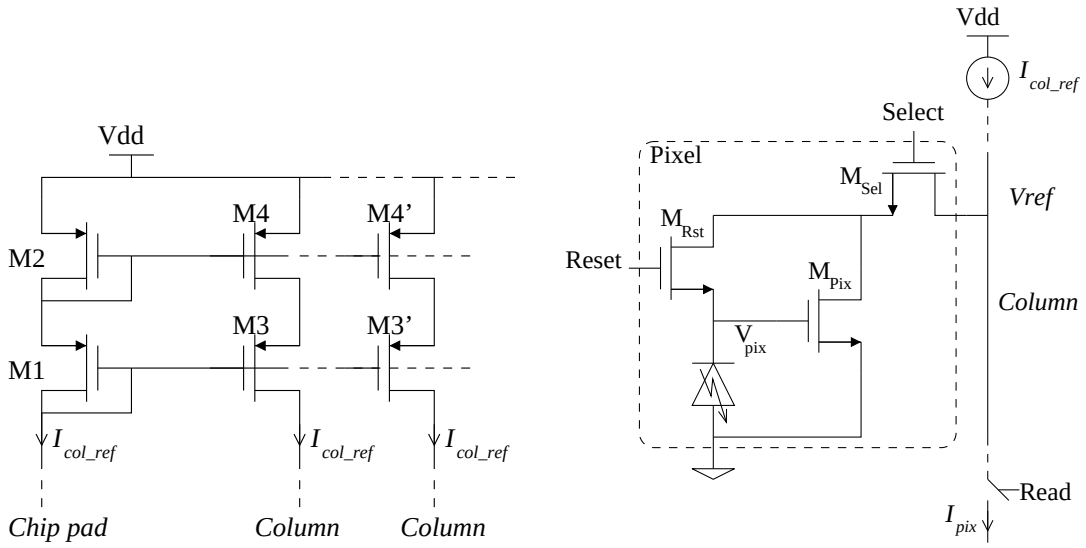
This chapter goes through the architecture of the convolution chip by following the natural flow of information. Following the diagram of figure 5.1, it starts with the pixel capturing the visual information all the way through the computing elements, to the final result of the convolution.

5.2 Imager

The imaging part of the chip is made of two entities: the pixel matrix containing the photo-sensitive elements and the pixel readout circuits, whose task is to transfer and transform the information for the pixels to the processing unit in an acceptable format.

Several types of pixel designs were studied, and two of them were implemented on different chips. A current-mode pixel was used in the first designs to minimize the complexity of the downstream processing. Because current-based pixel designs are prone to large fixed pattern column noise [56], a mode conventional voltage-mode pixel was used in the later designs. Thanks to a voltage to current converter added in the readout stage, the current interface with the downstream processing remains unchanged.

Regardless of the pixel design, an appropriate readout is necessary so a current can



(a) Self-biased current sources

(b) Current mode pixel

Figure 5.2: Column in the current-mode imager

be provided to the low impedance load of the convolution circuit. The current type pixel already provides the proper type of signal so the only additional functions needed aim at selecting the right columns that will be processed while scaling the currents to the expected range. The voltage type requires an additional conversion so the computation blocks receive a well-defined current flow. Fixed pattern noise reduction is also performed on the fly at that level.

Both options, along with the corresponding readout methodology, are described in this section. Emphasis is, however, placed on the voltage mode pixel paired with a voltage to current converter which was chosen for the final design.

5.2.1 Pixel implementation

5.2.1.1 Current mode pixel

In the configuration of figure 5.2, a constant current source provides a reference current down columns of the imager [56]. It is realized by providing a single reference to the chip and mirroring it for each column. A set of self-biased cascode current mirrors was used in this design. [57]

The transformation from light shining onto the pixel array to currents is a three-step process. First, the cell is reset so all potentials are initialized to a known value. Then, as light shines on the photodiode, the current source in the pixel gets biased and allows more current to flow through. And finally, the column is connected to a load where the difference between the reference current and the one flowing into the pixel is read.

Reset. Although it initiates the sequence, the reset is actually done last so the integration phase can benefit from the time needed to access the entire imager to gather light. This avoids wasting time as the imager is never idle so the frame rate is maximized.

$$V_{pix\_rst} = V_{ref} - V_{DS_{sel}} - V_{DS_{rst}},$$

where V_{ref} is the voltage drop across the current source at the top of the column:

$$V_{ref} = Vdd - V_{tp_3} - \sqrt{\frac{2I_{ref}}{\mu_p C_{ox}}} \cdot \left(\sqrt{\frac{1}{\left(\frac{W}{L}\right)_3}} + \sqrt{\frac{1}{\left(\frac{W}{L}\right)_4}} \right).$$

Integration The longest step of all. Light is gathered on the photodiode during the other rows' readout as well as during the idle time between frames.

$$V_{pix} = V_{pix\_rst} - \frac{1}{C} \int_0^T i_{diode} dt,$$

where i_{diode} is the photodiode current which is a function of the light intensity, and C is the junction capacitance of the diode added to the gate capacitance of M_{pix}.

Readout. The pixel transistor, biased by V_{pix} allows a current to flow through it. The purpose of the readout circuit will be to extract this information and process it.

$$I_{pix} = \frac{1}{2} \mu_n C_{ox} \left(\frac{W}{L} \right)_{pix} \left(V_{pix} - V_{tn_{pix}} \right)^2$$

The column is connected through a switch to a load which receives the difference between the reference current and the pixel current:

$$I_{out} = I_{ref} - I_{pix}.$$

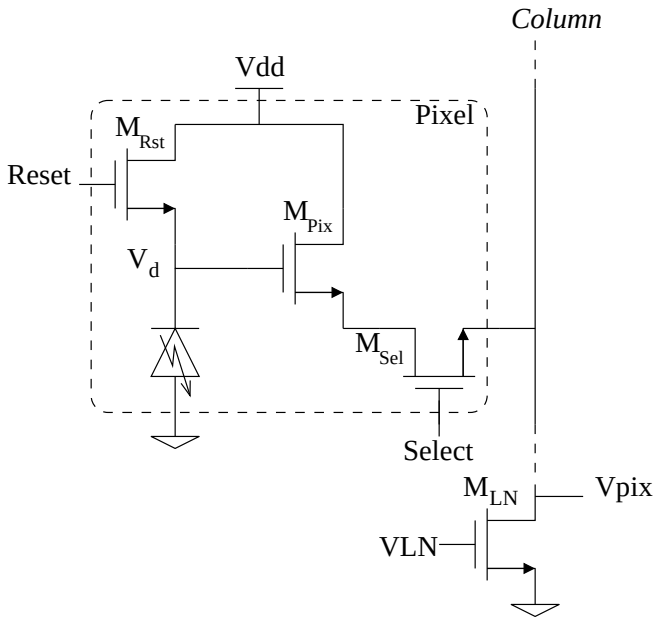


Figure 5.3: Voltage mode pixel

5.2.1.2 Voltage mode pixel

In this configuration, the current flowing down the pixel column is set by an externally biased current sink, M_{LN}, shown in figure 5.3. However, the potential of that line is set by the selected pixel. It is this voltage, $Vpix$ that is sent to the readout circuit described in section 5.2.2.1.

The controls are identical to those of the current mode pixel, so the sequence is the same (reset, exposure, readout). Two noteworthy differences are that the reset is done directly without transit through the select transistor and instead of being tied to ground, the signal is extracted from the gate of M_{pix}.

Reset. The NMOS reset switch holds the photodiode voltage V_d at one threshold voltage below the power supply:

$$V_{d_{RST}} = V_{dd} - V_{tn_{RST}}.$$

Integration. The photodiode current generated by the energy brought by light discharges the photo-diode node:

$$V_d = V_{d_{RST}} - \frac{1}{C}\int_0^T i_{diode} dt.$$

Readout. The signal is sampled at time $t = T$. The transistor M_{pix} follows the saturation

equation:

$$I_{LN} = \frac{1}{2}\mu_n \left(\frac{W}{L}\right)_{pix} \left(V_d - V_{pix} - V_{tn_{pix}}\right)^2,$$

where

$$I_{LN} = \frac{1}{2}\mu_n \left(\frac{W}{L}\right)_{LN} (V_{LN} - V_{tn_{LN}})^2 = constant,$$

and

$$V_d = V_{d_{RST}} - \frac{i_{diode} \cdot T}{C}.$$

The voltage read out is then:

$$V_{pix} = V_{d_{RST}} - \frac{i_{diode} \cdot T}{C} - \sqrt{\frac{I_{LN}}{\frac{1}{2}\mu_n \left(\frac{W}{L}\right)_{pix}}}$$

$$\Rightarrow V_{pix} = V_{d_{RST}} - \frac{i_{diode} \cdot T}{C} - (V_{LN} - V_{tn_{LN}}) \sqrt{\frac{\left(\frac{W}{L}\right)_{LN}}{\left(\frac{W}{L}\right)_{pix}}} \tag{5.1}$$

5.2.2 Readout circuit

The pixel designs proposed in sections 5.2.1.1 and 5.2.1.2 differ in the type of load that they are intended to be connect to. The first design expects a low impedance interface such as that of the arithmetic circuits of the chip. (Section 5.3 shows that the input of the multiplier is a low impedance current mirror.) It can therefore be used as is, and a simple switch is all that is needed for a readout circuit.

The voltage mode implementation, on the contrary, is designed to communicate with a high impedance load which contradicts the design specifications of the downstream arithmetic circuits. To match the impedance of the two modules, a voltage to current converter interface was designed, with the added functionality of reducing the fixed pattern noise caused by physical mismatches in the pixels and columns.

5.2.2.1 Voltage to current converter

The conversion of the pixel information from voltage to current is the very first step before any processing is done. Each column is equipped with a converter so processing can occur in parallel for the entire width of the imager. A compact circuit as proposed in figure 5.5(a) fits easily in the layout pitch imposed by the physical size of the pixel column ($10\mu m$ in

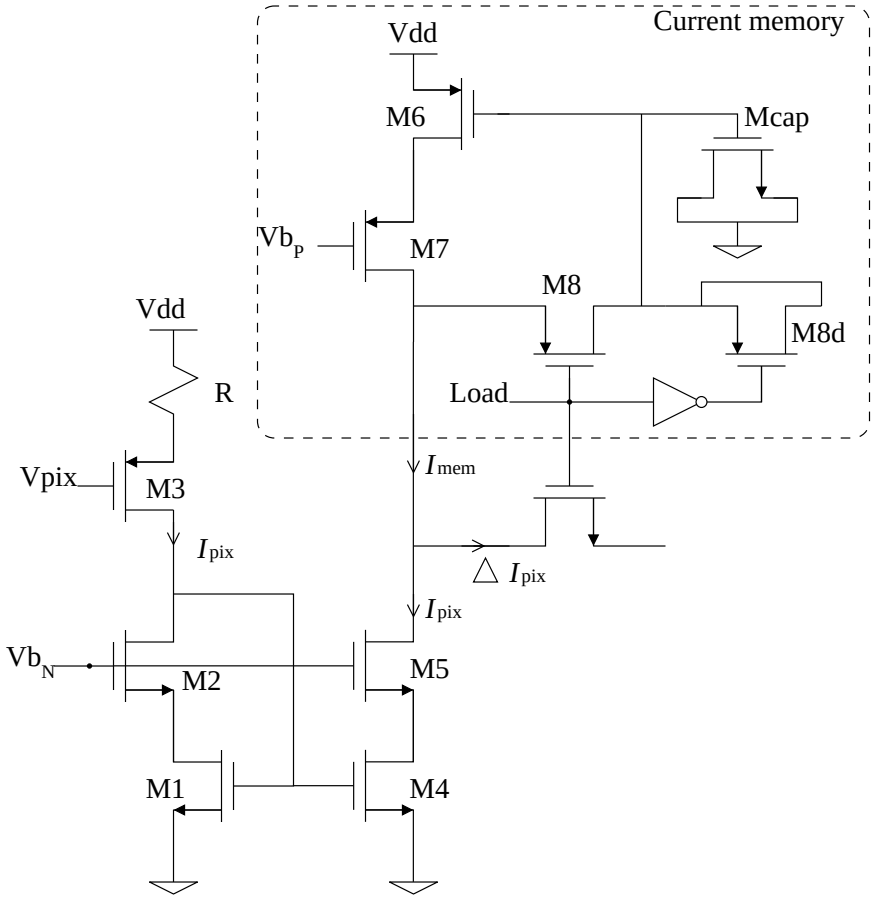


Figure 5.4: Voltage to current conversion and fixed pattern noise reduction

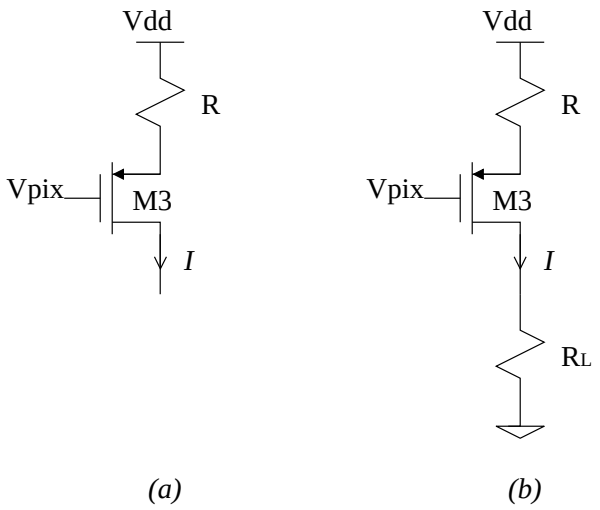


Figure 5.5: (a) Voltage to current conversion stage and (b) with a resistive load attached

the case of the convolution chip) while drawing less power than more bulky transimpedance amplifiers. The cost of the compact circuit is a reduced linearity which is studied in this section.

Assuming M_3 is in saturation (we will verify this hypothesis later),

$$I = \frac{1}{2}\mu_p C_{ox} \left(\frac{W}{L}\right)_3 \cdot (V_{SG_3} - V_{tp_3})^2 .$$

Let $K_3' = \mu_p C_{ox} \left(\frac{W}{L}\right)_3$.
Then, $I = \frac{K_3'}{2} (V_{SG_3} - V_{tp_3})^2$,

$$\Rightarrow V_{SG_3} = V_{tp_3} + \sqrt{\left(\frac{2I}{K_3'}\right)}.$$

If we look at the voltage drop across the resistor, $V_R = V_{dd} - (V_{pix} + V_{SG_3})$, we obtain another expression for the current I:

$$I = \frac{V_{dd} - (V_{pix} + V_{SG_3})}{R}.$$

Combining these expressions yields:

$$I = \underbrace{\frac{Vdd - V_{tp_3}}{R}}_{constant} - \underbrace{\frac{V_{pix}}{R}}_{linear} - \underbrace{\frac{\sqrt{\frac{2I}{K'_3}}}{R}}_{non\ linear} .$$

If we re-arrange this equation, we obtain:

$$R \cdot I + \sqrt{\frac{2I}{K'_3}} - (V_{dd} - (V_{pix} + V_{SG_3})) = 0.$$

Solving for I, we get:

$$I = \frac{1 + (V_{dd} - (V_{pix} + V_{SG_3}))\, K'_3 \cdot R + \sqrt{1 + 2\,(V_{dd} - (V_{pix} + V_{SG_3}))\, K'_3 \cdot R}}{K'_3 \cdot R^2},$$

or, more concisely, with the notation $V_R = V_{dd} - (V_{pix} + V_{SG_3})$,

$$I = \frac{1 + V_R \cdot K'_3 \cdot R + \sqrt{1 + 2V_R \cdot K'_3 \cdot R}}{K'_3 \cdot R^2}. \tag{5.2}$$

In order to plot the voltage to current conversion equation above, we use the parameters obtained from the chip fabrication:

$K'_3 = 37.2\mu A \cdot V^{-2} \times \left(\frac{W}{L}\right)_3 = 37.2\mu A \cdot V^{-2} \times \frac{1.2\mu m}{1.8\mu m} \Rightarrow K'_3 = 24.8 \times 10^{-6} A \cdot V^{-2}$

PMOS transistor threshold voltage: $V_{tp_3} = 0.96V$

Power supply voltage: $V_{dd} = 5V$

Resistor R: designed to be $R = 65.2 \times 10^3 \Omega$

Pixel voltage: measured to be in the range $V_{pix} \in [0; 1.5V]$

Using these parameters, we can plot equation 5.2, as shown in figure 5.6, with a linear fit of the region of interest showing the expected conversion.

Is M_3 always in saturation?

The equation derived for the voltage to current conversion is only as valid as the assumptions made. In this case, the transistor M_3 of the circuit schematic in figure 5.5(a) was assumed

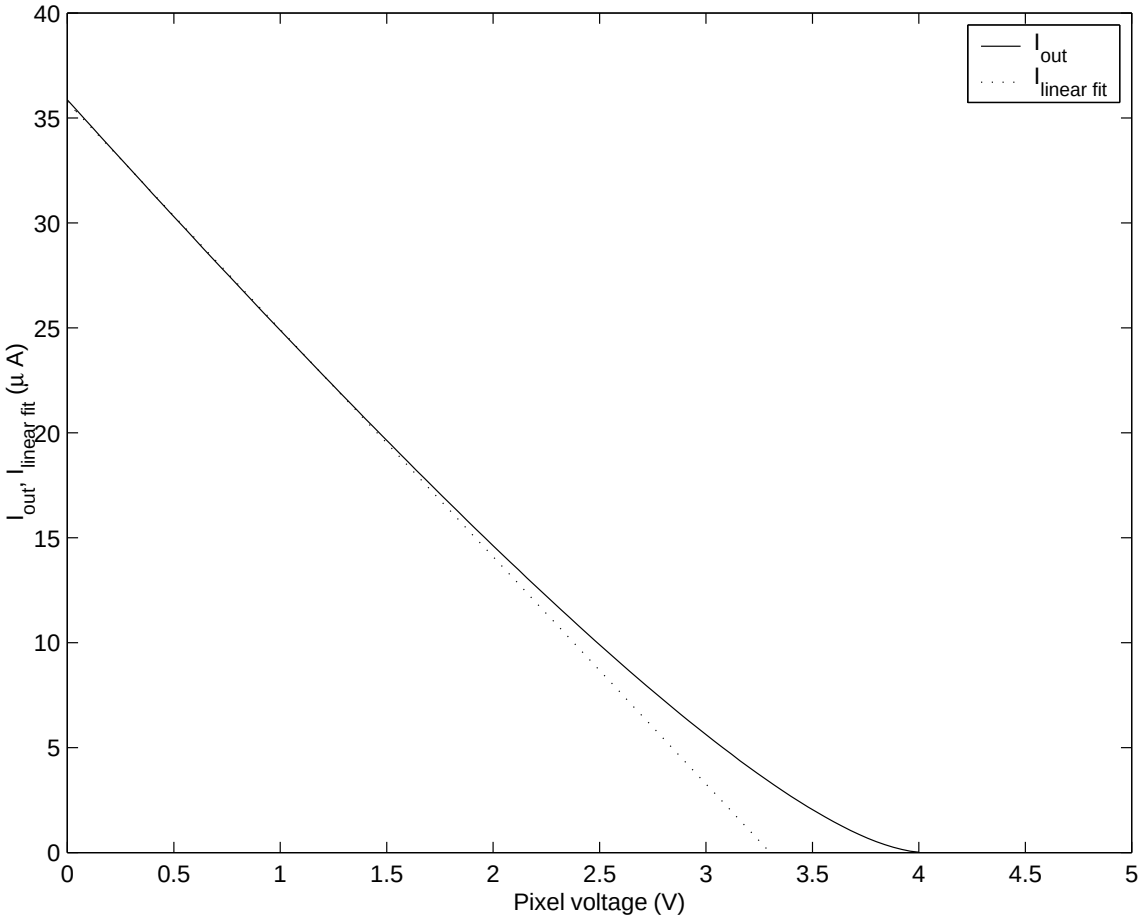


Figure 5.6: Matlab simulation of the voltage to current conversion with a linear fit (dashed line) on the region of interest $[0 : 1.5V]$. Slope of the fit: $-10.82\mu A/V$.

to be in saturation. To verify this, we take a closer look at that same circuit.

$M3$ is in saturation when $V_{SD_3} > V_{SG_3} - V_{tp_3} \Rightarrow V_{D_3} > V_{G_3} + V_{tp_3}$.

As a first approximation, $V_{D_3} \simeq V_{tn_1}$, and $V_{G_3} = V_{pix}$, the condition for saturation is:

$$V_{pix} \leq \underbrace{V_{tn_1} - V_{tp_3}}_{\simeq 0}.$$

The transistor $M3$ therefore remains in saturation in the operating range of the circuit.

For a further investigation, we re-draw schematic with a load resistor R_L as shown in figure 5.5(b).

M_3 is in saturation if $V_{SD_3} > V_{SG_3} - V_{tp_3}$, which is true when:

$$V_{dd} - RI - R_L I > V_{dd} - RI - V_{pix} - V_{tp_3}$$

$$\Rightarrow V_{pix} > R_L I - V_{tp_3}. \tag{5.3}$$

The actual load for the circuit will eventually be the input of a cascode current mirror. Its resistance is determined in equation 5.6. and the current I will be at most a few microamperes. It follows that the product $R_L I$ will always be smaller than V_{tp_3}, and $R_L I - V_{tp_3}$ will always be negative. The input voltage V_{pix} is, on the contrary, always positive. The inequality above is therefore always true and M_3 is indeed in saturation, validating the expression of the voltage to current relation of equation 5.2. [58]

5.2.2.2 Cascode load

The V-I converter described is connected to a cascode configuration as a load. Figure 5.7 shows the cascode stage. For this system to function correctly, proper bias voltage V_b has to be provided to this cascode structure. V_b is determined by looking at the relationship between the voltage and the current of this cell and identifying the condition for saturation of the transistors.

Assuming M_1 and M_2 are in saturation, first look at the current to voltage relationship in M_1:

$$I = \frac{1}{2}\mu_n C_{ox} \left(\frac{W}{L}\right)_1 (V_X - V_{tn_1})$$

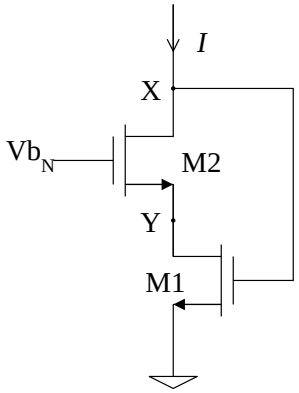


Figure 5.7: Cascode readout

$$\Rightarrow V_X = V_{tn_1} + \sqrt{\frac{2I}{\mu_n C_{ox}\left(\frac{W}{L}\right)_1}}. \tag{5.4}$$

The saturation of M_1 and M_2 is guaranteed by the setting of the bias voltage applied to M_2.

For minimal headroom consumption,

$$V_Y = V_{GS_1} - V_{tn_1}$$

$$\Rightarrow V_{b_n} = V_{GS_2} + (V_{GS_1} - V_{tn_1})$$

Similarly, the voltage to current relationship in M_2 is:

$$V_{GS_2} = V_{tn_2} + \sqrt{\frac{2I}{\mu_n C_{ox}\left(\frac{W}{L}\right)_2}}.$$

Substituting with equation 5.4, we obtain:

$$V_{b_n} = V_{tn_2} + \sqrt{\frac{2I}{\mu_n C_{ox}\left(\frac{W}{L}\right)_2}} + \sqrt{\frac{2I}{\mu_n C_{ox}\left(\frac{W}{L}\right)_1}}.$$

The condition of saturation of M_1 and M_2 is therefore:

$$V_{b_n} \geq V_{tn_2} + \sqrt{\frac{2I}{\mu_n C_{ox}}\left(\left(\frac{L}{W}\right)_1 + \left(\frac{L}{W}\right)_2\right)}. \tag{5.5}$$

With this relation, we can apply a bias voltage that guarantees saturation in M_1 and

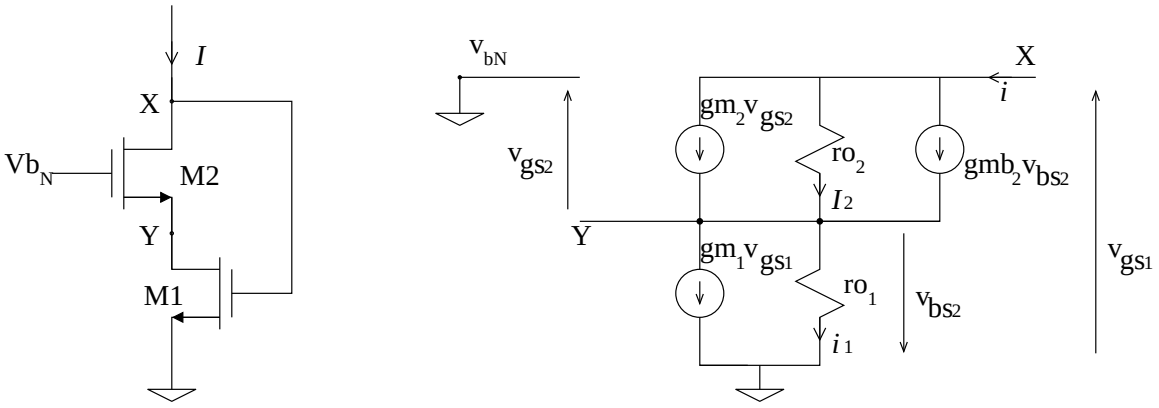


Figure 5.8: Cascode readout and small-signal equivalent circuit

M_2 for the range of current it is intended to receive. The current is set by the conversion from the pixel voltage and follows equation 5.2.

5.2.2.3 Resistive load of the voltage to current converter

When determining the voltage to current conversion, we assumed in equation 5.3 that the load resistance seen by the output of the converter circuit was small and we could count on the relationship $R_L I < V_{tp_3}$ to be true. To validate this assumption, we look at the small-signal equivalent circuit, figure 5.8, to calculate the input resistance of the cascode circuit.

$$i_2 = i - g_{m_2} v_{GS_2} - g_{mb_2} v_{BS_2},$$

$$i_1 = i - g_{m_1} v_{GS_1},$$

where $v_{GS_2} = -r_{o_1} i_1$; $v_{BS_2} = -r_{o_1} i_1$ and $v_{GS_1} = v_X$.

$$v_X = r_{o_1} i_1 + r_{o_2} i_2$$

Substituting,

$$v_X = r_{o_1} \left(i - g_{m_1} v_X\right) + r_{o_2} \left(i + g_{m_2} r_{o_1} I_1 + g_{mb_2} r_{o_1} i_1\right).$$

$$\Rightarrow v_X \left(1 + r_{o_1} g_{m_1} \left(1 + r_{o_2} \left(g_{m_2} + g_{mb_2}\right)\right)\right) = i \left(r_{o_2} + r_{o_1} \left(1 + r_{o_2} \left(g_{m_2} + g_{mb_2}\right)\right)\right)$$

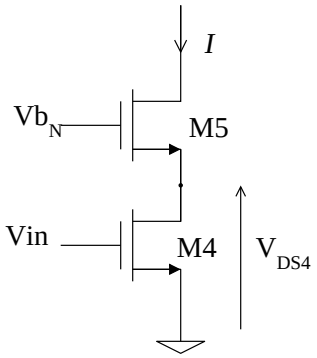


Figure 5.9: Output of the cascode readout current mirror

The input resistance is therefore:

$$R_{in} = \frac{r_{o_2} + r_{o_1}\left(1 + r_{o_2}\left(g_{m_2} + g_{mb_2}\right)\right)}{1 + r_{o_1} g_{m_1}\left(1 + r_{o_2}\left(g_{m_2} + g_{mb_2}\right)\right)}. \tag{5.6}$$

This expression is greatly simplified by taking into account the relationships $g_m \gg 1/r_o$ and $g_m > g_{mb}$. It reduces to:

$$R_{in} = \frac{1}{g_{m_1}}.$$

When using this resistance as the load resistance in equation 5.3, we find that the assumption was indeed valid for the range of currents that flow in the circuit and the transistor $M3$ is indeed in saturation as assumed in section 5.4.

5.2.2.4 Output of the readout current mirror

The saturation of the transistors forming the output of the readout current memory, shown in figure 5.9, can be verified by studying the response of $M4$ and $M5$ with changes of the bias voltage V_{b_n}.

$$I = \frac{1}{2}\mu_n C_{ox}\left(\frac{W}{L}\right)_5 \left(V_{b_n} - V_{DS_4} - V_{tn_5}\right)^2$$

$$I = \frac{1}{2}\mu_n C_{ox}\left(\frac{W}{L}\right)_4 \left(V_i n - V_{tn_4}\right)^2$$

$$\Rightarrow V_{DS_4} = \left(V_{b_n} - V_{tn_5}\right) - \sqrt{\frac{2I}{\mu_n C_{ox}\left(\frac{W}{L}\right)_5}}$$

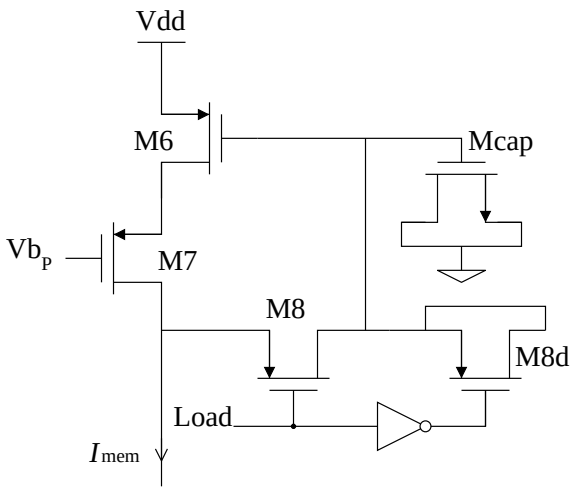


Figure 5.10: Current memory

The transistor M_5 is indeed in saturation if $V_{DS_4} \geq V_{in} - V_{tn_4}$:

$$V_{b_n} \geq V_{tn_5} + \sqrt{\frac{2I}{\mu_n C_{ox}} \left(\left(\frac{L}{W} \right)_5 + \left(\frac{L}{W} \right)_4 \right)}.$$

This relationship is similar to that found while studying the input of the cascode stage in equation 5.5. It includes parameters determined through trade-offs, such as the transistor geometries, and is set by the interface requirements, such as the current I flowing through the cell. The compromise is discussed in section 6.2 on the design decisions for the circuit building blocks.

5.2.3 Fixed pattern noise reduction

Fixed pattern noise (FPN) in the imager appears at both the pixel level and the column level. The mismatch in pixel elements creates pixel level artifacts. The mismatch of the elements of the readout circuitry which are used when reading out of all the pixels of one same column, as described in section 5.2.2, creates column-based offsets. Both effects can be reduced by a two-step sampling and difference circuit. In this scheme, it is the difference between the pixel being read out and an image of the fixed pattern at that location that is sent to the output of the imager. [59]

5.2.3.1 Current memory

The ability to duplicate and hold a current is essential for the operation of several functions in the chip. This section focuses on improving the image quality by reducing the spatial noise in the imager as shown in section 5.2.3.2. Such memory cells are also extensively used in the pipeline accumulator used to reconstruct the processed image, as described in section 5.4.

Figure 5.10 shows the current memory cell that was implemented for fixed pattern noise reduction. The current corresponding to the exposed pixel is memorized and later subtracted from the reset value of the same pixel. The subtraction is used to remove the spatial component of the noise accumulated up to that point, as detailed in section 5.2.3.2.

A cascode PMOS current mirror structured is used, where the same two transistors serve both on the input and the output side. [60, 61]

During memorization, the load phase, a switch connects the gate of the mirror transistor M_6 to the input. A capacitor on that node charges up to the voltage corresponding to the current flowing through it:

$$V_{cap} = V_{GS_6} = V_{tp_6} + \sqrt{\frac{2I_{mem}}{\mu_p C_{ox} \left(\frac{W}{L}\right)_6}}. \tag{5.7}$$

When the switch opens (the read phase), the voltage level at the gate of the mirror transistor M_6 is held by the capacitor at the same level V_{cap} from equation 5.7. The drain current of M_6 remains constant, equal to the current that was flowing during the load phase.

In lieu of implementing an actual capacitor (double-poly capacitors were available for the fabrication process used) which would be quite large, a NMOS transistor, M_{cap} was configured as a capacitor by connecting both its source and drain to ground. A double-poly capacitor of $0.5\mu F$ would require an area of $562\mu m^2$ while the the NMOS capacitor only occupies $202\mu m^2$. The extra clearance required around the layout of a double-poly capacitor further increases the difference.

Since $V_{DS_{cap}} = 0$, the transistor operates in the deep triode region and its gate to source and gate to drain capacitances are equal:

$$C_{GS} = C_{GD} \Rightarrow C_{mem} = WLC_{ox} + 2WC_{ov},$$

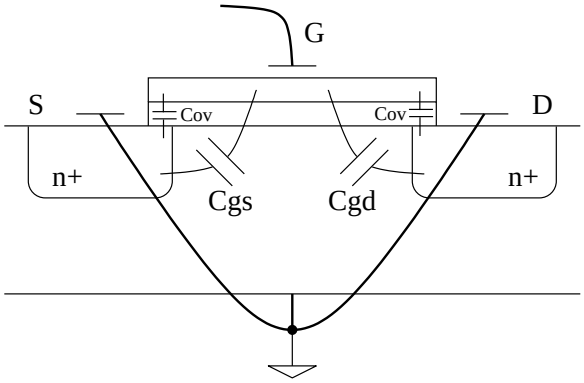


Figure 5.11: MOS capacitor: cross-section

where C_{ov} is the capacitance per unit width due to the overlap of the drain or the source under the transistor gate oxide. The overlap effect is not only small compared to C_{GS} and C_{GD}, but it does not affect the memory, as the voltage is properly retained regardless.

5.2.3.2 Difference circuit

With the current memory memorizing the incoming pixel, the fixed pattern noise at that location can be removed by sampling the difference between the saved signal and the pixel reset level which only contains the noise information.

The value of the exposed pixel is sampled on a current memory as in section 5.2.3.1, then the pixel is reset so the image information is removed from the signal and only the fixed noise features of the pixel remain. The difference of the two signals is then sent to the output, cleaned from the fixed pattern noise accumulated until that point.

Figure 5.12 shows the simplified circuit and the chronogram describing the sequence to create the difference of the signals:

1. During the memorization period ($Load = 1$), the output switch is open so $\Delta I = 0$. The current memory is storing the value of the exposed pixel: $I_{mem} = I_{pix}$.

2. During readout ($Load = 0$), the addressed pixel is being reset. The current memory sources the current memorized in the previous step but only the current corresponding to the reset level. The difference is sent to the branch connected to the next stage where it will be either processed or read out.

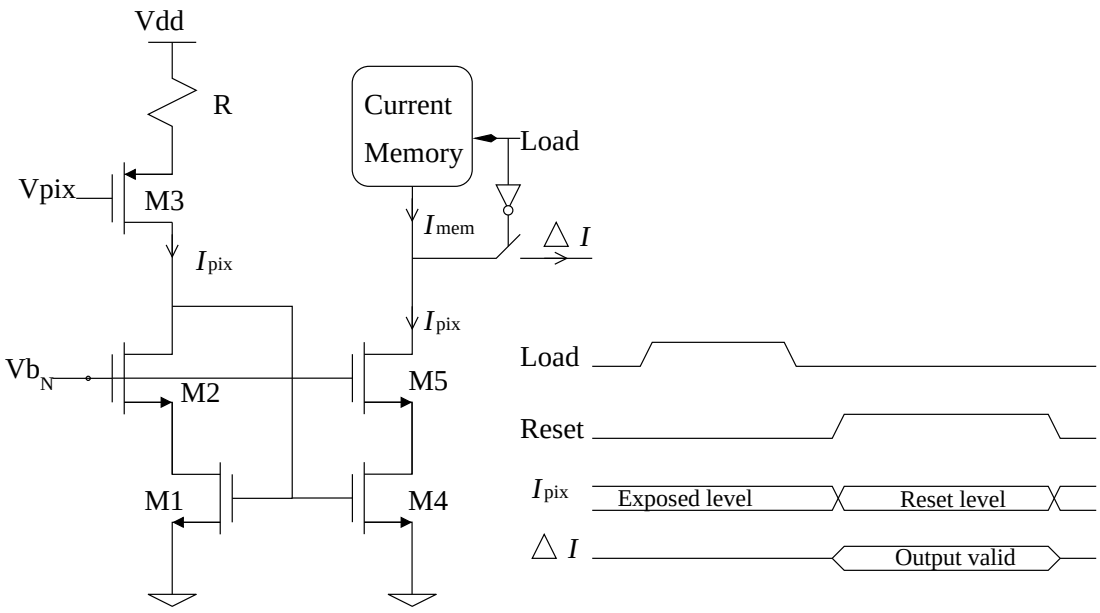


Figure 5.12: Current difference circuit and chronogram

$$\Delta I = I_{mem} - I_{pix}$$

It is this difference ΔI that represents the image and becomes the input of the computing circuits that calculates the convolution. It is combined for this purpose with the digital template in a multiplying DAC, described in section 5.3. The difference can alternatively be sent to an output to monitor the raw image and characterize the imaging performance as described in section 8.2.

5.3 Multiplying DAC

The convolution operation requires the pixel-wise multiplication of the kernel with the neighborhood of the pixel that is being processed. For this, a multiplier unit was implemented in this work that accepts operands from the imager and from the template. The signal provided by the imager is analog, as seen in the previous sections. During design, the template could have been chosen to be either analog or digital as it is meant to be uploaded by the user.

The three choices for designing the multiplier were therefore to built an analog multiplier, a digital multiplier or a mixed-signal multiplier. The decision on the nature of the input

signals was closely linked to this choice. Analog multipliers using CMOS of several types have been available, starting with the Gilbert cell [62], modified for MOS transistors [63,64]. Various designs operating in either the transistor saturation or linear region allow four-quadrant or one-quadrant multiplication of analog signals. [65–69]

The typical analog multiplication schemes [70] and circuits [71] share the same overall advantages and disadvantages. The circuits are typically small in terms of transistor counts compared to their digital counterparts and operate very quickly. (The unclocked analog multipliers are limited by the speed of the transient response of the transistors which is much faster than is needed in this application) Digital multipliers require more hardware but produce a noiseless result when sufficient bus width is provided to avoid overflows. [72–76] Floating-point operations also suffer from round-off noise [77] but are not necessary for the convolution since all operands are integers.

Since the operands are analog for the image and digital for the template, an entirely analog or digital multiplier scheme requires converting one of the signals through an analog to digital [78–81] or digital to analog converter [82–85], which unnecessarily increases the complexity of the circuit. A mixed-signal multiplication scheme [40–42] that accommodates the nature of both operands was therefore chosen for this task.

The multiplier accepts operands of a different type and performs the multiplication in a non-clocked scheme. The output is exclusively a function of the two operands, no control signal or clock is provided in the final version. An early implementation of the multiplier described here uses a clocked output as part of the calculation.

The first operand is provided as an analog current, output of the pixel readout circuit described in section 5.2.2, while the other operand is stored as a digital signal in a serial-input, parallel-output memory. Although the digital signal is changeable by the user, when computing the convolution over a single frame, it is fixed and available at all times. Each pixel of the kernel is stored in the form of an 8-bit unsigned integer.

Since the nature of the signal from the imager is to be kept in the entire signal chain, the multiplying cell was designed to output the result as a current that can be read out in the same way as the signal from the imager.

Therefore the digital template will be converted to an analog signal through a modified digital to analog converter operating in the current domain where the reference current is the one provided by the imager.

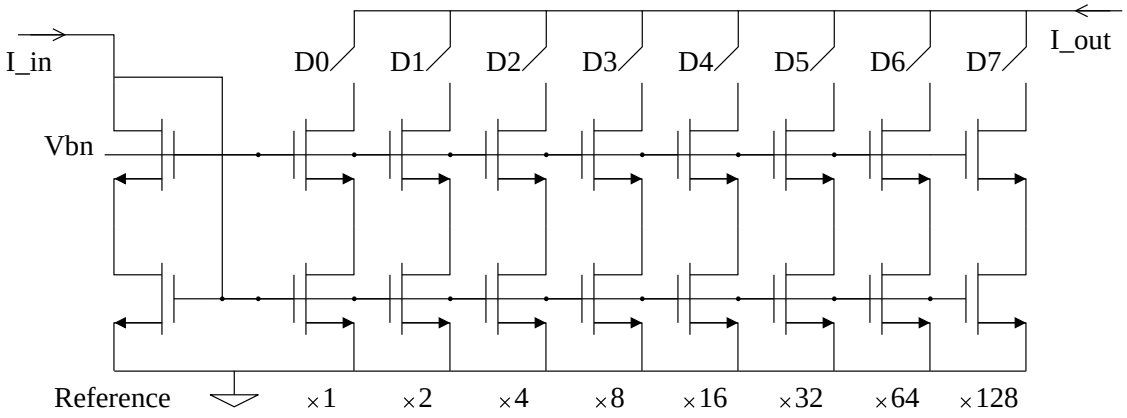


Figure 5.13: Multiplying DAC current ladder

5.3.1 Binary-scaled ladder

The multiplying DAC converts the digital kernel into a current, using the input from the pixel as reference. The output vary linearly with respect to this reference and to the kernel, producing the multiplication of the two. The converter build is based on an array of binary-scaled amplifiers controlled by the kernel bits:

Let K be the number corresponding to the digital operand and I be the analog current input, the second operand. If K is a natural integer which has been stored as an n-bit digital number, K_i is the binary value of the i^{th} bit of K. The validity of the multiplying DAC lies on the associative property:

$$K = \sum_{i=0}^{n-1}(K_i \times 2^i) \Rightarrow I_{out} = K \times I_{in} = \sum_{i=0}^{n-1}(K_i \times I_{in} \times 2^i). \tag{5.8}$$

The input current is duplicated and amplified through a number of current mirrors corresponding to the number of bits in the digital signal. The amplification takes the gain corresponding to the position of the mirror when laid out in the form of a binary-scaled ladder. That is, the first mirror has unity gain, the second a gain of two, then four, eight, sixteen, up to 128. [40, 86]

Each bit of K controls a switch allowing the amplifier current to flow or not. All allowed currents are added, yielding the desired result.

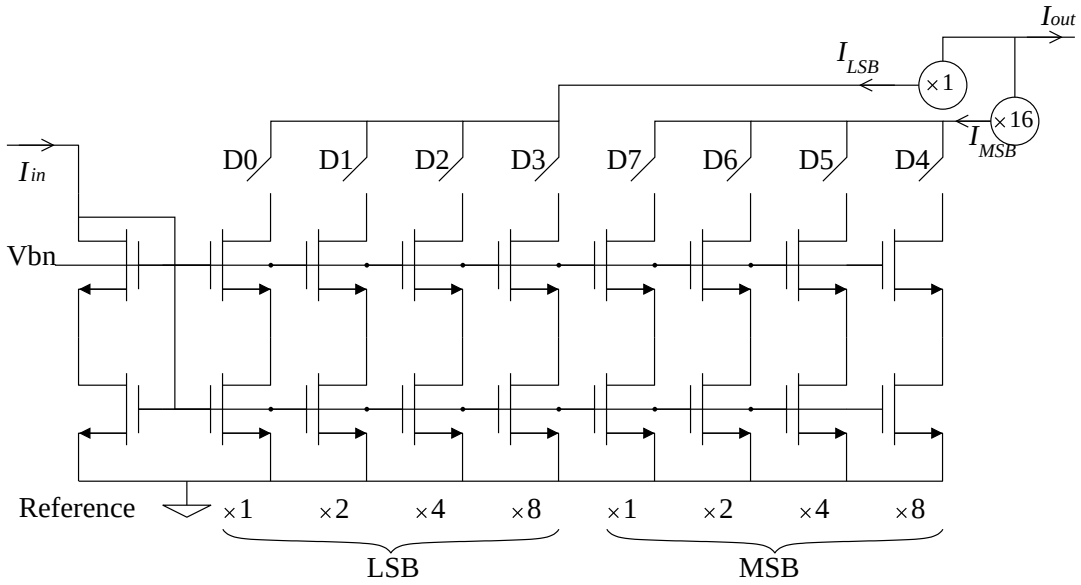


Figure 5.14: Multiplying DAC current ladder with separate lsb and msb

5.3.2 Output scaling

Achieving 8-bit multiplication using this scheme requires laying out a large number or transistors. The reference transistors (replicating the pixel current at the input of the current mirror) is duplicated 255 times, which takes up excessive space on the chip. To implement this cell, we had to divide it into two identical blocks corresponding to the template's four most- and least- significant bits respectively. The output of each of the LSB blocks is simply read out while the ones from the MSB are amplified.

Two methods were studied and implemented to reconstruct the full signal. One uses time-controlled switches which regulate the number of charges that flow from each block and are integrated onto a capacitor. The other uses a $\left(\frac{W}{L}\right)$ geometrical scaling of the current transferred in a series of current memories.

5.3.2.1 Time scaling

In the case of capacitors used as accumulators, it is not only the current that will determine the voltage across the capacitor, but also the time the charges are allowed to flow into it. Although the multiplication cell generates a current proportional to the product of the two operands, the result will be read as the voltage across the readout capacitor, proportional to the number of accumulated charges: $V_{out} = \frac{Q}{C} = \frac{1}{C}\int_0^T Idt = \frac{I \cdot T}{C}$. The ×16 amplification

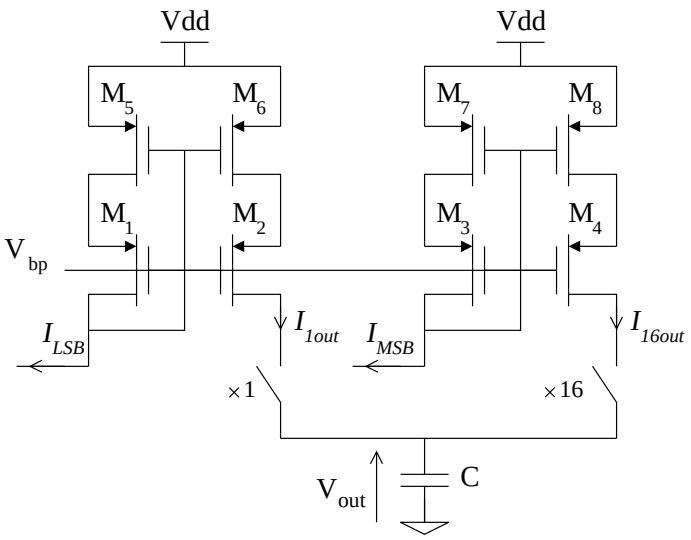


Figure 5.15: Multiplying DAC time-scaled output simplified circuit

of the most-significant bits is achieved by controlling a switch that remains closed sixteen times longer than that on the least-significant bits output. The charges flow for a longer time in the capacitor and the resulting voltage scales linearly with time:

$$V_{out} = \frac{1}{C}\left(\int_0^T I_1 dt + \int_0^{16T} I_{16} dt\right) = \frac{(I_1 + 16 \cdot I_{16}) \cdot T}{C}.$$

A very accurate clock division is required when using time as an input to such an arithmetic operation. If an accurate, high-speed master clock is not available, errors in the correspondence between least- and most-significant bits will occur, which can translate either as inaccuracies or actual arithmetical errors such as when the monotonicity of the multiplication is compromised, i.e., when $(15 \times I) > (16 \times I)$ [87].

5.3.2.2 Geometric scaling

When using a capacitor as the readout element of the multiplier, as in the circuit described in section 5.3.2.1, the current flow is regulated with accurately controlled clock signals to give more weight to the most significant template bits. These clock signals allow to scale the multiplication output as needed. However, using a series of current memories and current-mirror amplifiers, as in the circuit in section 5.4, is more appropriate to maintain the nature of the signal.

To interface such a circuit, the outputs of two halves of the multiplier are amplified using

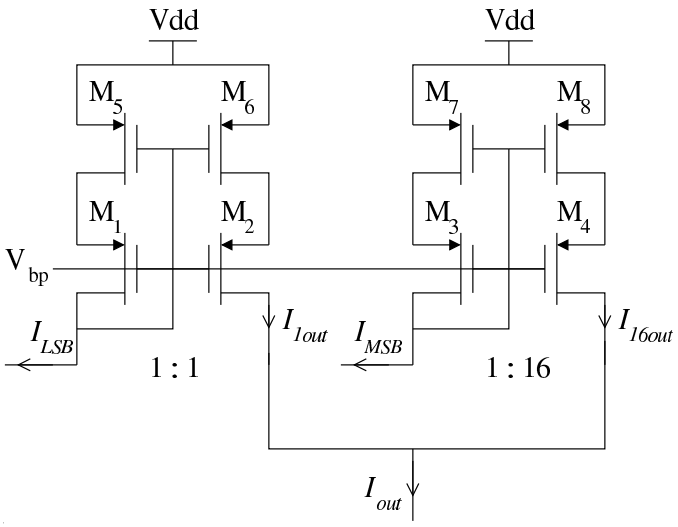


Figure 5.16: Multiplying DAC geometric-scaled output simplified circuit

cascode current mirrors, in the same way the multiplying DAC was implemented. Figure 5.16 shows a simplified schematic of the output circuit where the output current is:

$$I_{out} = I_1 + 16 \cdot I_{16}.$$

Using this method for the output stage, the multiplying DAC does not require any control signal or timing information to operate. The output is entirely a function of its two operands, and it only changes when a new pixel is made available.

5.4 Accumulators

The consequence of the row-wise mode of operation of the APS imager is that computation over a neighborhood of pixels can not take place in a fully parallel fashion. The information from all the pixels taking part in the convolution needs first to be released by the imager. The 9×9-pixel convolution requires nine rows to be sent from the imager to the processing unit before the first results can be computed. With a pipeline architecture for the accumulator, the flow of pixels is not slowed down once it starts after the nine-row latency.

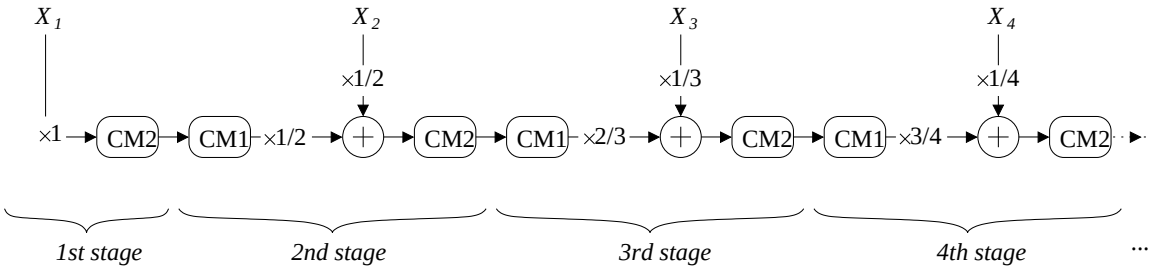


Figure 5.17: Operation of the pipeline accumulator; CM = Current Memory

$$I \otimes K = \sum_{j=1}^{9} \underbrace{\left(\sum_{i=1}^{9} (I_{i,j} \cdot K_{i,j}) \right)}_{row\ partial\ products} \tag{5.9}$$

5.4.1 Pipeline

The *row partial products* in equation 5.9 are the outputs of the multipliers. Nine neighboring pixels were multiplied pixel-wise with the nine kernel rows and added row-wise with each other, yielding the nine partial products. Those are noted X_i in the diagram of the pipeline accumulator, figure 5.17. The input of each stage of the pipeline comes from the multiplier controlled by the corresponding kernel row, i.e., $\sum_{i=1}^{9} (I_{i,j} \cdot K_{i,j})$ goes to the X_j input. When a new row is addressed in the imager, the kernel multiplications are added to the previous states of the pipeline and shifted until the ninth stage where the final result is read out.

The states of the pipeline represent the partially constructed sum and are stored on current memories similar to the one in figure 5.10, used in the pixel readout circuit. A two-step operation (transfer the input into a first current memory, then the sum into a second current memory) prevents an open flow through all the stages which would cause a loss of the stored information.

The output of a simple adder grows linearly with the number of inputs. Therefore, the accumulator has to work with a range of amplitude ninefold larger than that of the input current without distortion or saturation. This demand on the design requires the current memories and mirrors to operate over a wide range while consuming significantly more power than if working with the same amplitude range as the inputs of the accumulator. A solution to this problem is to scale the partial products in each stage of the pipeline

to ensure a constant range of operating current. The inputs are scaled down by a factor matching their position in the pipeline and each partial product receives the complement so the resulting sum has a constant weight of one.

$$\frac{\sum_{i=1}^{9} X_i}{9} = \frac{8}{9}\left(\frac{7}{8}\left(\frac{6}{7}\left(\frac{5}{6}\left(\frac{4}{5}\left(\frac{3}{4}\left(\frac{2}{3}\left(\frac{1}{2}\left(X_1\right)+\frac{1}{2}X_2\right)+\frac{1}{3}X_3\right)+\cdots+\ \frac{1}{8}X_8\right)+\frac{1}{9}X_9 \right. \right. \right. \right. \right. \right. \tag{5.10}$$

The rolling sum and averaging is constructed in nine steps by introducing the partial product values from the new rows in each stage of the pipeline. Equation 5.11 shows the partial product at stage j, constructed with the j first inputs coming from the sum of pixel-wise multiplications of the j first rows of the kernel with the currently addressed row of the imager array. This is similar to equation 3.3 which appeared in the algorithm study in chapter 3.

$$\frac{\sum_{i=1}^{j} X_i}{j} = \left(\frac{j-1}{j}\right) \cdot \underbrace{\frac{\sum_{i=1}^{j-1} X_i}{j-1}}_{previous\ stage} + \left(\frac{1}{j}\right) \cdot X_j \tag{5.11}$$

The output of stage 9 expands to the same expression as equation 5.10. It is the output of the accumulator and therefore the result of the convolution. The accumulator introduces a scaling factor to preserve the swing of the signals:

$$X_i = \sum_{i=1}^{9} \left(I_{i,j} \cdot K_{i,j}\right) \Rightarrow \left.\frac{\sum_{i=1}^{j} X_i}{j}\right|_{j=9} = I \otimes K. \tag{5.12}$$

The system simulation of the nine-stage pipeline accumulator was presented in section 3.3.1 where the algorithm was first introduced. The Matlab code encoding the complete accumulator, and used on the images figures 3.5 to 3.10 shown as examples can be found in section A.1.2.

5.4.2 Single-cell structure

The first stage of the pipeline accumulator does not use a previously saved partial product. Its purpose is to initiate the pipeline with the first kernel row. For the algorithm to work properly, the input current X_1 needs to be available for transfer to the second stage when needed. This happens when all the inputs of the other stages have been scaled and transfered

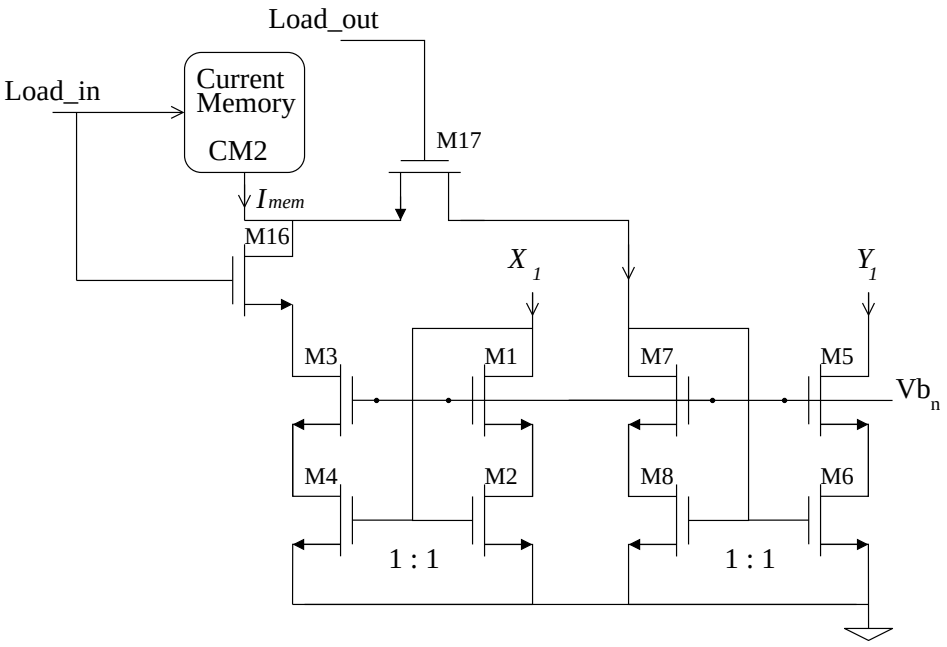


Figure 5.18: One-step cell in the pipeline: first stage

into the $CM2$ memory cells of figure 5.17. To maintain the robustness of the pipeline, a $CM2$ memory cell also appears into the first stage. It is synchronized with those of the entire pipeline, guaranteeing the availability of the signal even if the input has already been withdrawn.

The structure of the initial stage, shown in figure 5.18, is similar to but simpler than its multi-input counterparts. The single current memory saves the current input X_1 on its load mode and releases it on its read mode to the output Y_1. Since no scaling occurs in the first stage, the current mirrors all have a ratio $\frac{\left(\frac{W}{L}\right)_{in}}{\left(\frac{W}{L}\right)_{out}} = 1$.

Load (Load_in). $I_{mem} = X_1$.

The X_1 current input flows through $M16$ into the current memory which is in load mode. The $M17$ transistor acts as an open switch and prevents loss of current to the output.

Read (Load_out). $Y_1 = I_{mem}$.

$M16$ is open and the current memory is in read mode. The memorized current, corresponding to what the X_1 input was equal to (it is no longer assumed valid at this time), flows though $M17$ to the output.

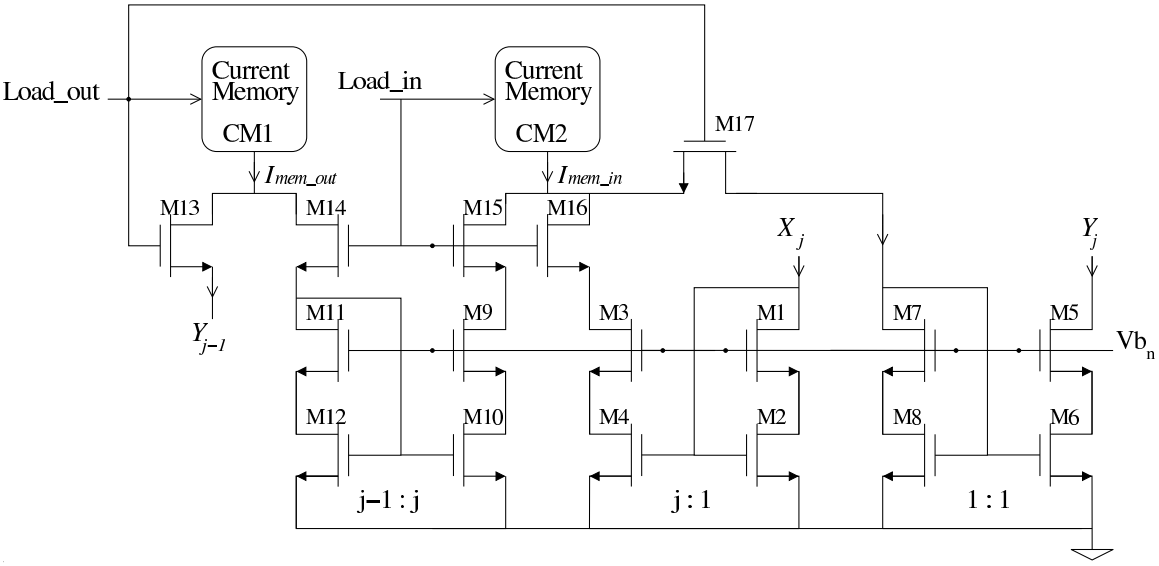


Figure 5.19: Two-step cell in the pipeline: j^{th} stage

Note that the *Load_in* and *Load_out* control signals are non overlapping so that leakage of the saved current to the output during the load phase is minimized.

5.4.3 Double-cell structure

With the exception of the first stage which only has to work with one input, all the stages of the pipelines share the same structure. The two inputs are scaled, combined and transfered to the next level through two current memory cells, $CM1$ and $CM2$ on the simplified schematic of figure 5.19.

Computationally, each of the double-cell stage implements the transformation of equation 5.11. The two inputs are the current from the multiplier X_j which is getting inserted into the pipeline at that stage, and the connection with the previous stage which is a partial product that has been built up in the pipeline up to that point.

$$Y_j = \left(\frac{j-1}{j}\right) \cdot Y_{j-1} + \left(\frac{1}{j}\right) \cdot X_j$$

All the double-cell structures and the one single-cell structure share the same two control signals. Their operation is therefore very similar. The charging of the memory $CM2$ obeys the same timing, ruled by the *Load_in* signal while that of the memory $CM1$ happens when the cells share outputs on *Load_out*.

Load (Load_in). $I_{mem\_in} = \left(\frac{1}{j}\right) X_j + \left(\frac{j-1}{j}\right) Y_{j-1}$.

The X_j current input is scaled down by $\left(\frac{1}{j}\right)$ while the current previously saved in $CM1$ (now in read mode), I_{mem_out} is scaled down by $\left(\frac{j-1}{j}\right)$. They flow through $M16$ and $M14/M15$ respectively into the $CM2$ current memory which is in load mode and memorizes the sum of the two. The $M13$ and $M17$ transistor act as open switches and prevent exchange of information with the other stages of the pipeline.

Read (Load_out). $Y_j = I_{mem\_in}$ and $I_{mem\_out} = Y_{j-1}$.

The transmission of the partial products between stages occurs on the *Load_out* signal. The transistors $M14$, $M15$ and $M16$ act as open switches. The $CM2$ current memory is in read mode and the $CM1$ memory in load mode. The I_{mem_in} current from $CM2$ is sent to the output Y_j while the current output of the previous stage Y_{j-1} flows through $M13$ into $CM1$ where it is memorized. It is this current that will be used in the next sequence with the next incoming X_j from the multiplier.

Chapter 6

Design and Simulation

6.1 Introduction

The complete calculation of the convolution between the acquired image and a kernel involves arithmetic operators such as multiplications and additions. This results in an amplification of the signal. There is, therefore, a need for a way to handle a wide dynamic range. For the digital implementation of chapter 4, this was dealt with by assuming the worst-case scenario (uniform kernel at maximum value with a saturated incoming image) and by increasing the width of the internal data bus accordingly. When working in the analog world, other solutions must be found to address the challenges brought by the long signal chain and to insure linearity, prevent clipping and maximize the signal to noise ratio.

Because of the computational nature of the system, it is necessary to insure that the accuracy of each stage can be guaranteed over the entire range. Closely related to this is the noise introduced by each element that must be addressed to minimize its effects on the final result. The absolute value of the current flowing in the signal chain is also of importance as the power consumption of the chip directly depends on it.

It is these considerations, and others that are specific to the function of each circuit, that have guided the many choices made during the design phase. These decisions are the topic of this chapter. The functional description of the circuits which are the focus of the greater part of chapter 5 are used as references and starting points for the design of the cells and their optimization. The challenges of each designed computational block of the signal chain are detailed with the design requirements of the cell and the simulations used to validate the choice of the various parameters.

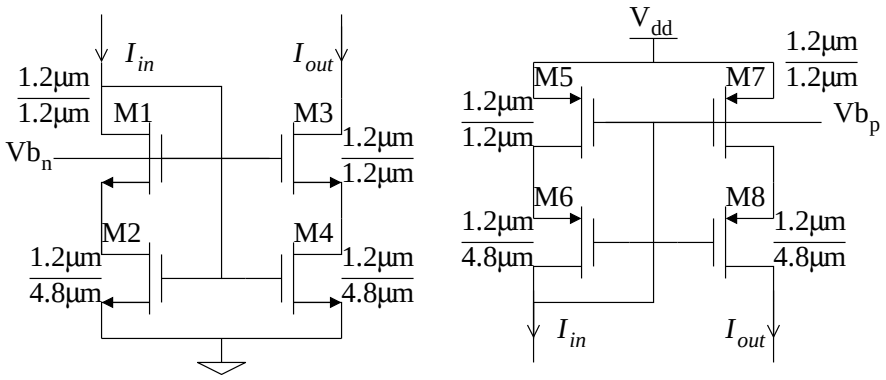


Figure 6.1: Cascode N- and P-type current mirrors

6.2 Cascode current mirror

As the main building block for most computation elements, the cascode current mirrors have to be reliable in their accuracy and spatial homogeneity over the chip. The design of the mirrors takes into account the need for linearity over the operating current ranges and aims for good matching for predictable arithmetic results.

The cascode current mirror as studied in section 5.2.2, also shown in figure 6.1, is used for many tasks in the convolution. They are described with the architecture of the chip in chapter 5. For an overview of their main uses, see section 5.2.2 on the pixel readout circuit and current memory, section 5.3 on the mixed-signal multiplier and section 5.4 on the pipeline structure of the accumulator. Although the design of each of these structures is studied separately in this chapter, they are important in this section as they all rely on the accuracy of the current mirrors over their operating range of currents.

With proper choice of the bias voltage, we can assume that all transistors in figure 6.1 are in saturation. For the N-type structure, we have on the input side:

$$I_{pix} = \frac{1}{2}\mu_n C_{ox}\left(\frac{W}{L}\right)_2 (V_{GS_2} - V_{tn_2})^2$$

$$\Rightarrow V_{in} = V_{GS_2} = V_{tn_2} + \sqrt{\frac{2I_{in}}{\mu_n C_{ox}\left(\frac{W}{L}\right)_2}}. \tag{6.1}$$

Similarly, on the output side,

$$V_{GS_4} = V_{tn_4} + \sqrt{\frac{2I_{out}}{\mu_n C_{ox} \left(\frac{W}{L}\right)_4}}. \tag{6.2}$$

Since the gates of M_2 and M_4 are connected, $V_{in} = V_{GS_2} = V_{GS_4}$. Therefore, equations 6.1 and 6.2 can be combined:

$$V_{tn_4} + \sqrt{\frac{2I_{out}}{\mu_n C_{ox} \left(\frac{W}{L}\right)_4}} = V_{tn_2} + \sqrt{\frac{2I_{in}}{\mu_n C_{ox} \left(\frac{W}{L}\right)_2}}.$$

A special case of this equation is the ideal one when the threshold voltages are equal. ($V_{tn_4} = V_{tn_2}$) It then yields the current mirror equation:

$$\frac{I_{out}}{I_{in}} = \frac{\left(\frac{W}{L}\right)_4}{\left(\frac{W}{L}\right)_2}.$$

The same derivation with P-type structures shows that for both types, any uncertainty on the geometrical parameters (W and L) and the threshold voltage variations will have a large effect on the effectiveness of the mirrors. Their matching across small and large areas will be studied in chapter 7, as will the computation errors they might introduce in the image convolution. The bias voltages also play a role in how well the current mirrors perform and must be set with care to insure the linearity of the mirrors over the desired range of currents.

All the current mirrors needed in the convolution chip circuits operate in the $[0 : 10\mu A]$ range of currents. They share the same bias voltages (V_{b_n} and V_{b_p}, depending on the polarity) and sizing. The exception is the occurrence of multiple parallel transistors used for current scaling.

The characteristic curves of the current mirrors and their associated difference plots (deviation from the ideal case when $I_{out} = I_{in}$) are shown in figures 6.2 and 6.3 respectively for the N-type mirror and figures 6.4 and 6.5 for the P-type mirror. Both types of mirrors only show small deviations from the ideal curve over the operating range. They lose their mirroring properties for input currents greater than $16\mu A$ for lack of headroom. This leaves a range significantly higher than necessary for the system performance during normal operation.

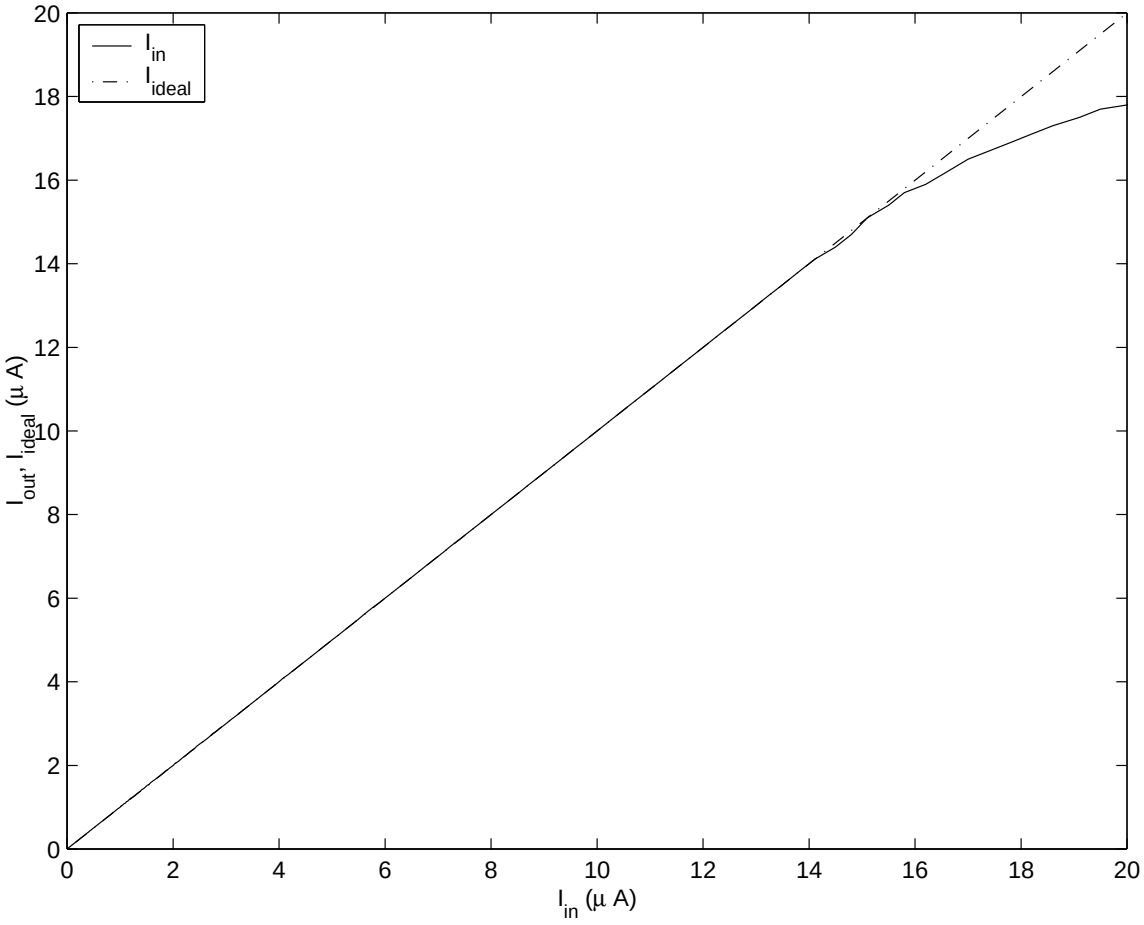


Figure 6.2: N-type current mirror simulation output

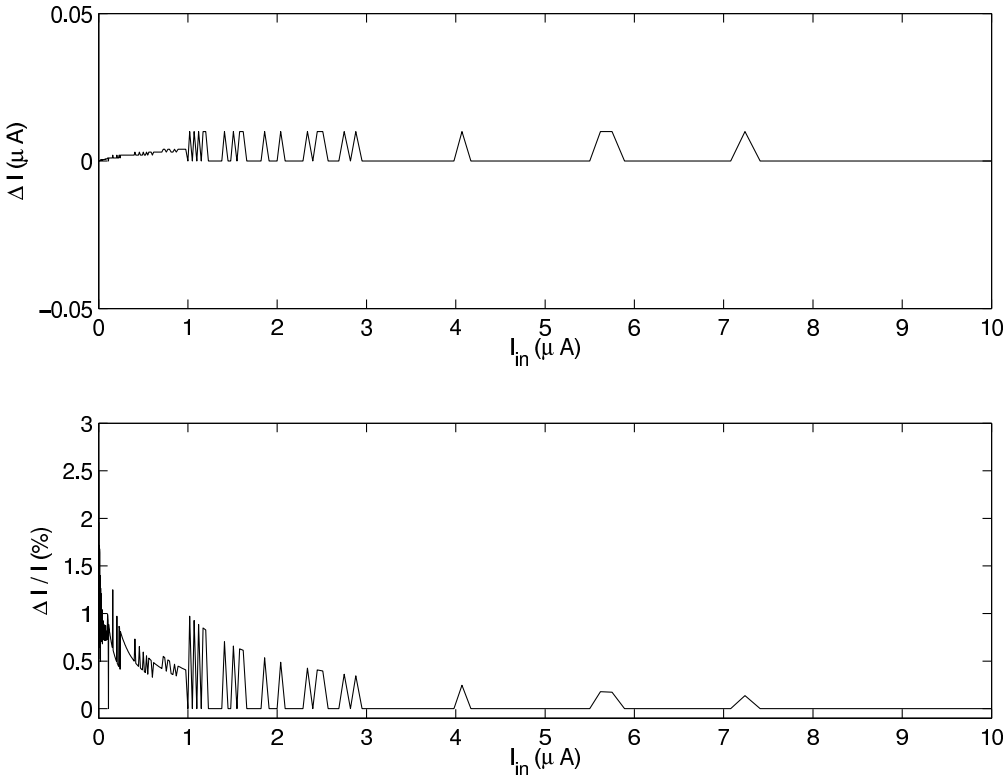


Figure 6.3: N-type current mirror simulation output difference plot (top) and fractional error (bottom)

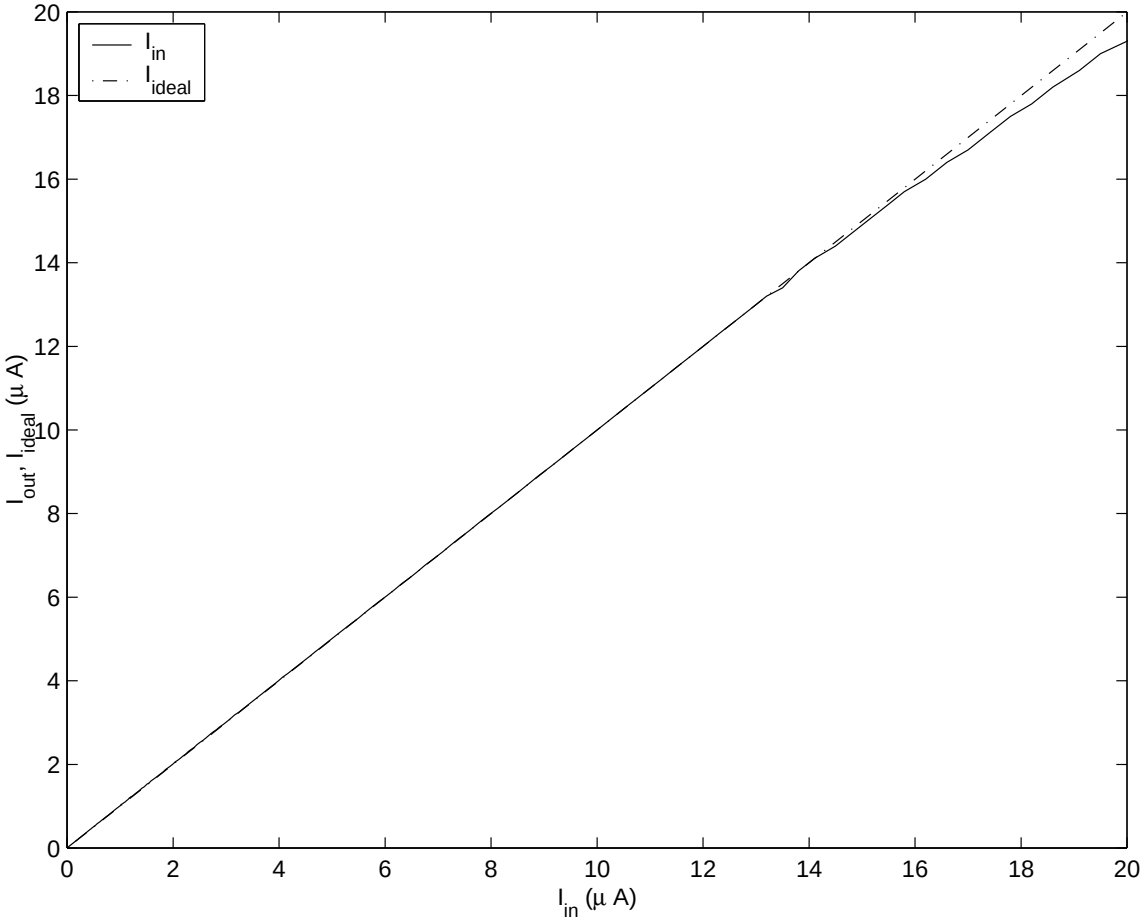


Figure 6.4: P-type current mirror simulation output

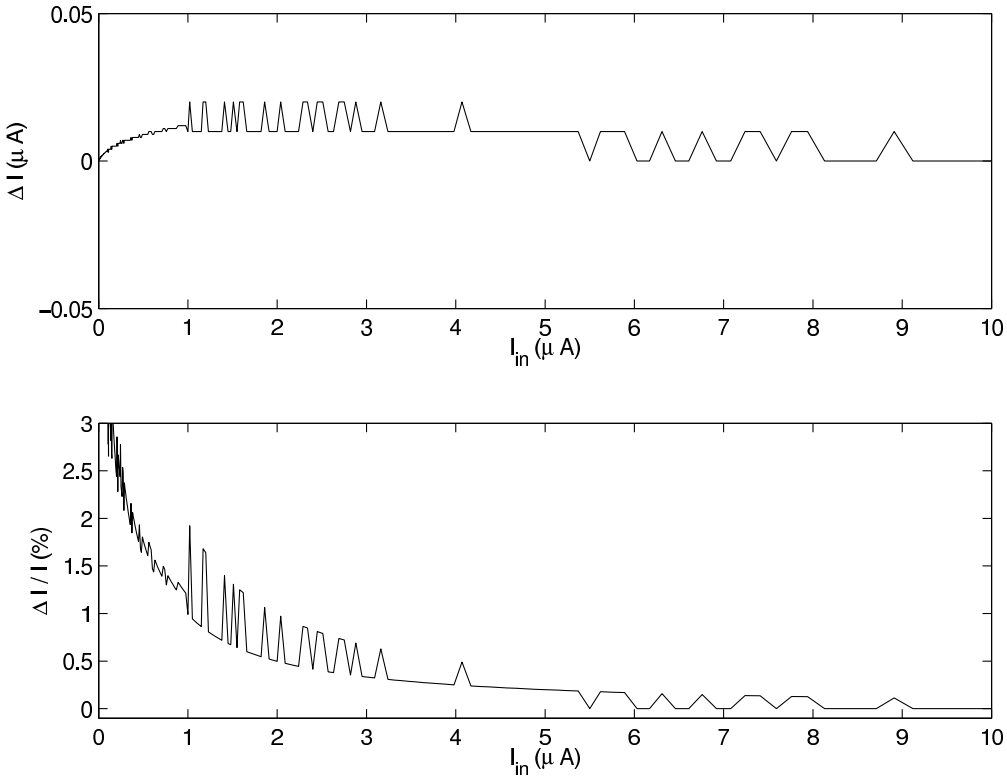


Figure 6.5: P-type current mirror simulation output difference plot (top) and fractional error (bottom)

6.3 Current memory

Current memories are used extensively in the convolution chip. Because it is such a versatile circuit, the description of its operation in section 5.2.3.1 makes no reference to the application in which the memory cells are used. To ensure all design considerations are appropriate, every function they perform in the chip have to be looked at separately in their respective context.

Following the natural data flow of the chip, the first occurrence of current memories is in the pixel readout, specifically the fixed-pattern noise reduction circuit shown in section 5.2.2 where the reset pixel level is subtracted from the saved pixel value to reduce the fixed column noise.

The other block taking advantage of these same memory cells is part of the convolution computation. They are used extensively for each incoming row to transfer the information of the partial products inside the accumulator pipelines described in section 5.4.

The design choices common to all applications of the current memories are discussed here in this section. The application-specific issues and the interaction with surrounding circuits are studied in the next sections on pixel read-out and accumulators.

A schematic of a current memory is shown in figure 6.7 as part of the pixel readout circuit. Since it is similar to all other current memories in the convolution chip, it is used here to illustrate the design description.

As explained in details in section 5.2.3.1, this current memory cell is similar to that described by Daubert et al. [61] and also Moeneclaey et al. [60] in that it uses two transistors, $M6$ and $M7$, operating as half a cascode current mirror. A switch, $M8$, allows changing the operating mode from load to source, while a MOS capacitor M_{cap} serves as the memory point. It is charged during the load operating mode, and its charge controls the current when in source mode.

To simulate the current memory circuit, a series of memorizations are performed with various current inputs. Figure 6.6 (bottom) shows the chronogram with the succession of read and load controls. The top figure shows the input of the cell (dashed line) and the output current (solid line) which is zero anytime the read signal is low and matches the input when the read signal is high. The input is held at the same level during the read phase only for comparison purposes. As expected for a memory cell, its switching immediately

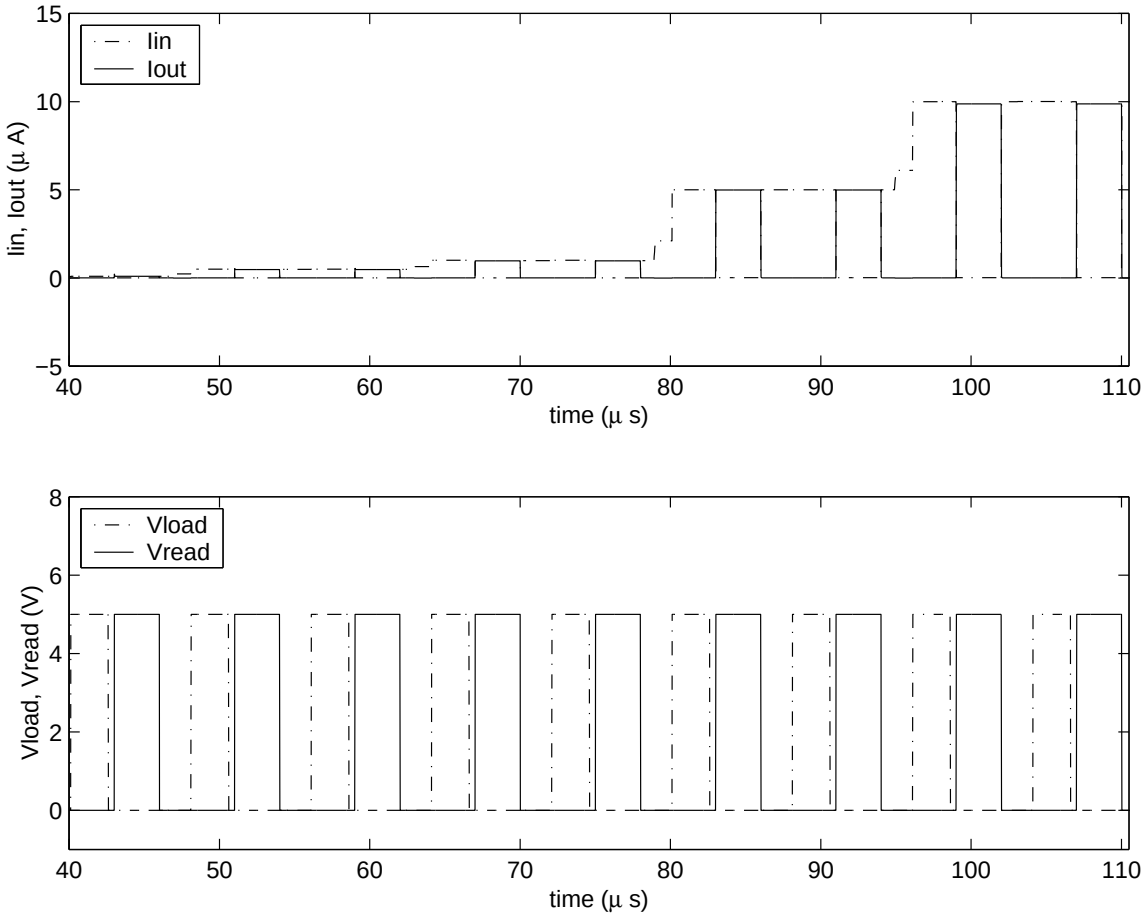


Figure 6.6: Current memory simulation. Input and output currents (top) and chronogram (bottom)

after the load phase does not affect the output.

The precision of the charge on M_{cap} is key to proper current restoration. Apart from the transistor noise, studied in section 6.7.2, which causes some uncertainty on the accuracy of the memorization, it is also impaired by the charge injection from the switching of the digitally controlled transistor $M8$. To minimize the effect of charge feed-through, a dummy capacitor $M8_d$ of half the width of $M8$ is added next to the switch. Its only purpose is to compensate for the charges injected when $M8$ changes states [88–90]. Practically, $M8$ and $M8_d$ are made of three transistors laid out next to each other to increase the quality of the matching and the equality of the number of charges injected. Two of them are wired to make the switch, and the third is used as the dummy.

6.4 Pixel readout

The initialization of the analog signal chain occurs when the pixel array is read out and conditioned to be transmitted to the computation unit. This two-step process described in section 5.2.2 starts with the conversion of the voltage mode pixel to a current, followed by a fixed-pattern noise reduction circuit. The two components with their connecting mirror form the complete readout cell as shown in figure 6.7.

The input interface of the readout cell is of course the pixel itself which provides the initial signals to be processed; therefore this section begins by studying the format of the pixel output. The specifications of the converter stage, which turns the pixel information into a current and voltage range that meets the requirements of the computing cells, are introduced next. Finally the entire readout circuit with a current memory is presented as the main element of the FPN reduction circuit. Section 6.3 goes over the details of the operation of the current memory.

6.4.1 Voltage-mode pixel

The voltage level at the output of the pixels depends on a number of parameters which affect the range of operation and consequently the design of the readout circuit. A close look at equation 5.1 (repeated here as equation 6.3) and figure 5.3 shows that the output voltage range is limited by the geometry of the pixel integration transistor (M_{pix}) and the current source at the bottom of the column (M_{LN}).

$$V_{pix} = V_{d_{RST}} - \frac{i_{diode} \cdot T}{C} - (V_{LN} - V_{tn_{LN}}) \sqrt{\frac{\left(\frac{W}{L}\right)_{LN}}{\left(\frac{W}{L}\right)_{pix}}} \tag{6.3}$$

The photodiode current i_{diode} and the integration time T are user controlled to maximize the light detection while avoiding saturation, that is to allow as large a signal swing as possible while avoiding pulling V_{pix} all the way to 0.

The available range is set by the geometry of the pixel and the current sink transistors M_{pix} and M_{LN} and the bias voltage V_{LN}. The specifications for the interface with the readout circuitry are set by the pixel. As seen in the simulation output of figure 6.8, a voltage in the range of $[1.5V : 0]$ spanning linearly the light levels between darkness and saturation is assumed when designing the downstream circuit.

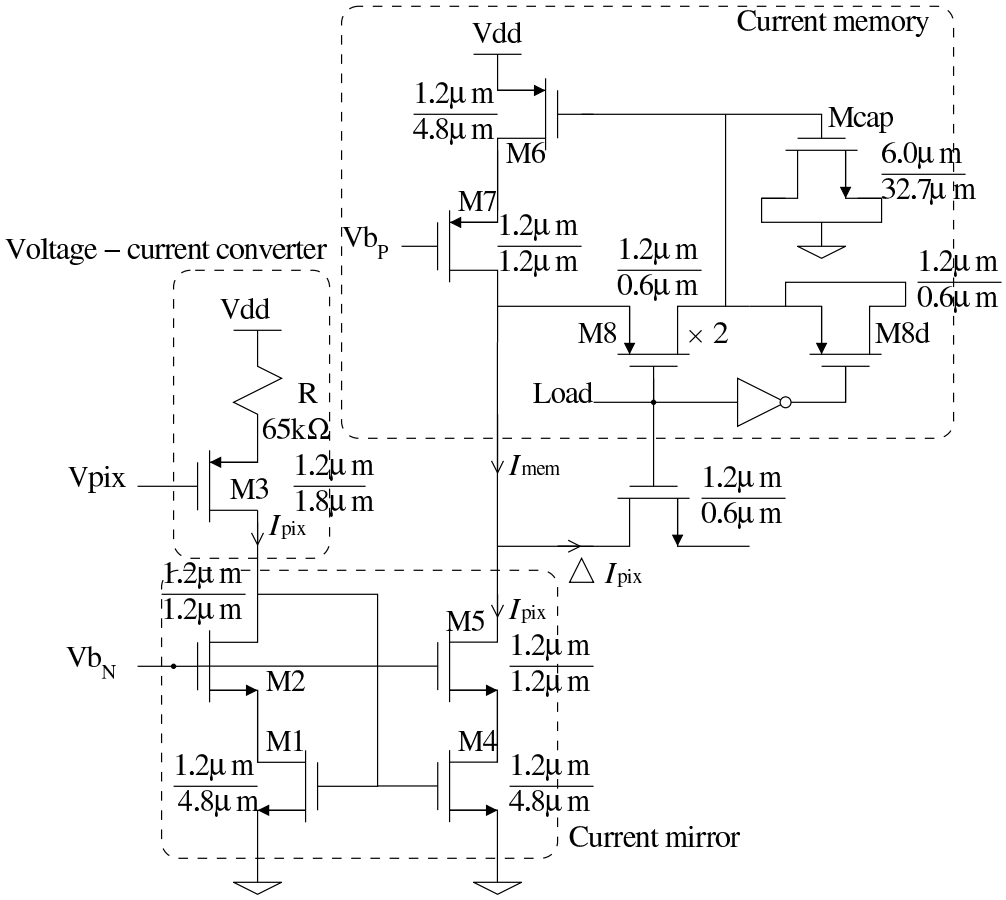


Figure 6.7: Voltage to current conversion and fixed pattern noise reduction

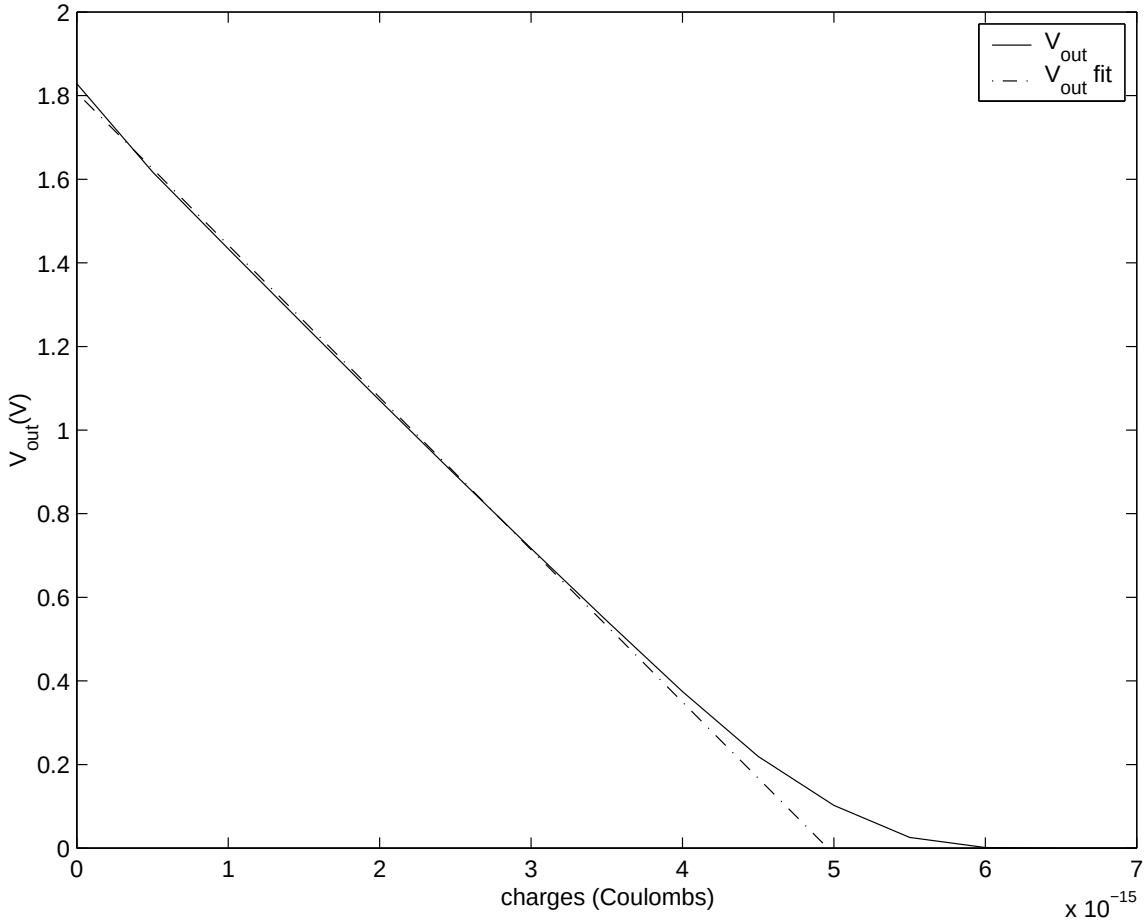


Figure 6.8: Pixel response to light. Simulation output with a linear fit on the linear region (dashed line)

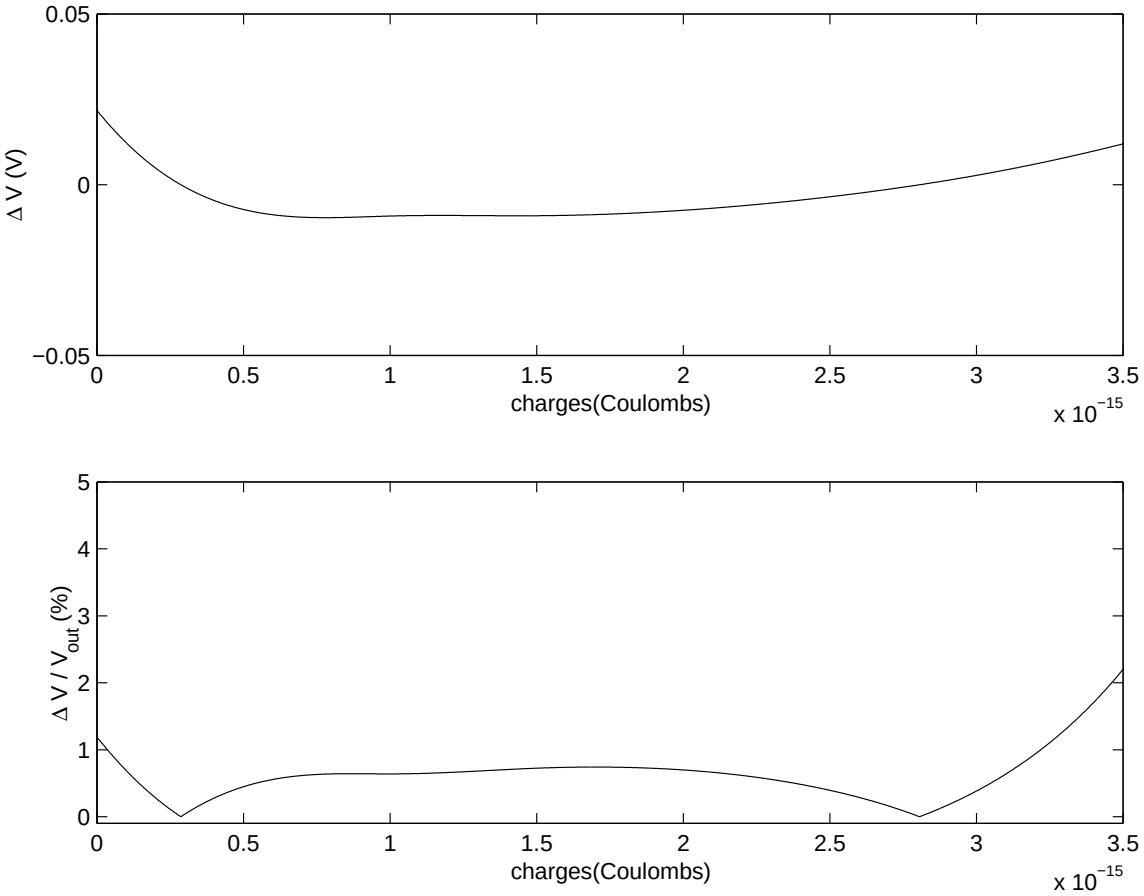


Figure 6.9: Pixel response to light: absolute (top) and relative (bottom) difference plots

6.4.2 V-I conversion

The voltage to current converter is the interface between the sensor and the computation stage. The requirements are to adapt the signal to transfer the $[0V : 1.5V]$ voltage range from the pixel to a usable range of current for the convolution blocks. Following the schematic and labels of figure 6.7, the transfer of the incoming pixel high impedance source (voltage mode) to a low input impedance load (current mode) effectively occurs in the modulating resistor R. R will therefore determine the magnitude of the current flowing through it and into its load: the current mirror pair $M1 - M2$. The input transistor $M3$ introduces a non-linear effect which should ideally be minimized in the range of operation. Section 5.2.2.1 on the circuit architecture explains the details of the operation of this cell.

The load of the converter is a current mirror with its output connected to a current memory for fixed pattern noise reduction. The design specificities of those cells are detailed in sections 6.2 and 6.3. For best linearity while conforming to their interface, the conversion stage should maintain the current between $0\mu A$ and $10\mu A$.

The simulation plot of figure 6.10 shows first that the range of current expected by the current mirror load is exceeded. This is addressed by the scaling inside the mirror. (See the complete readout cell description in section 6.4.3.)

The second important piece of information is the linearity of the conversion. A linear fit of the transfer function over the range of operation ($[0 : 1.5V]$ input voltage) predicts a slope of $-8.25\mu A/V$ with a relative error of less than 0.5%, as shown on the difference plot and relative error plot of figure 6.11.

6.4.3 Fixed pattern reduction

The purpose of the pixel readout circuit is to transform the raw information from the pixel into a usable signal for the rest of the chip. The term "usable" in this context means a signal that conforms to the nature of the circuits it is connected to, and as precise and predictable as possible. To achieve this, two steps are used. First, the voltage to current converter described above operates continuously to generate a current proportional to the pixel column line is it connected to. When a pixel is selected, it can thus be probed through a current mirror biased by the converter.

To increase the accuracy of the signal, a fixed pattern reduction stage is introduced in

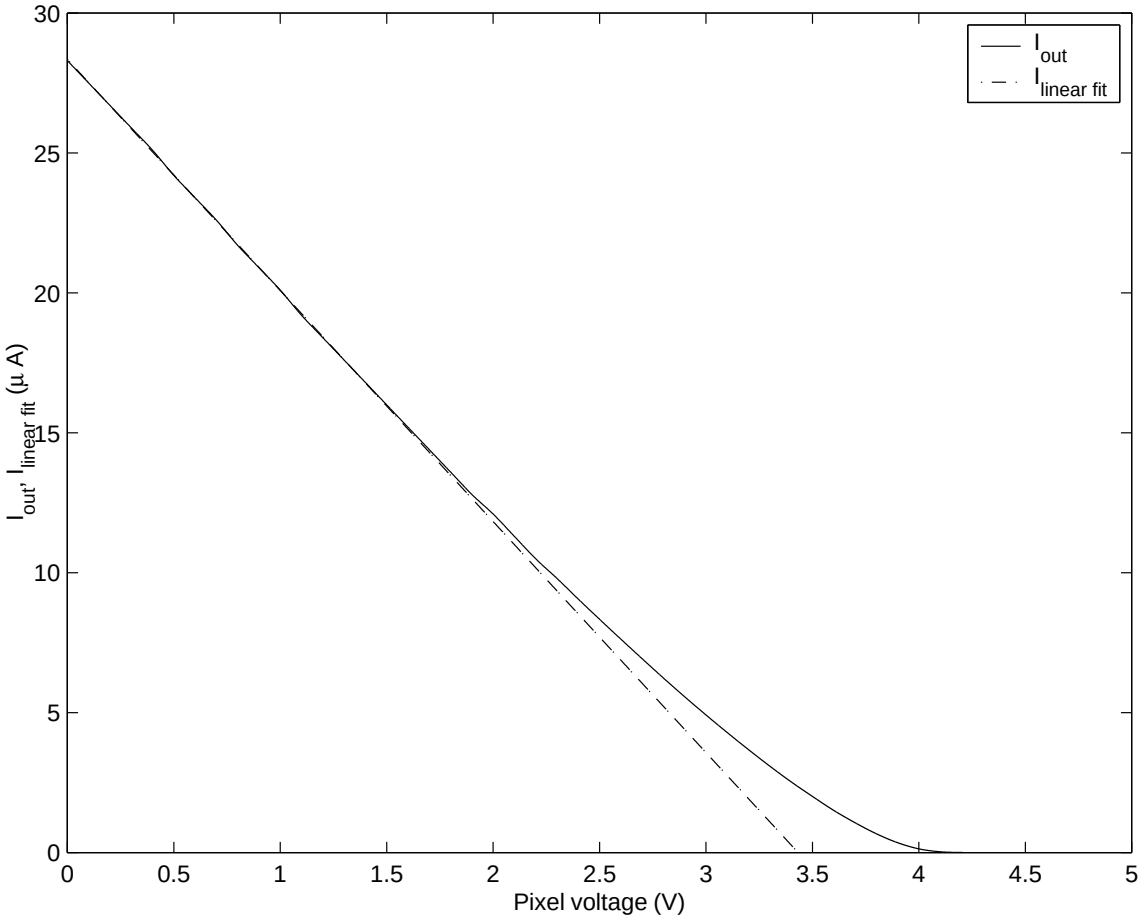


Figure 6.10: Voltage to current conversion simulation output with a linear fit (dashed line) on the region of interest $[0 : 1.5V]$. Slope of the fit: $-8.25\mu A/V$

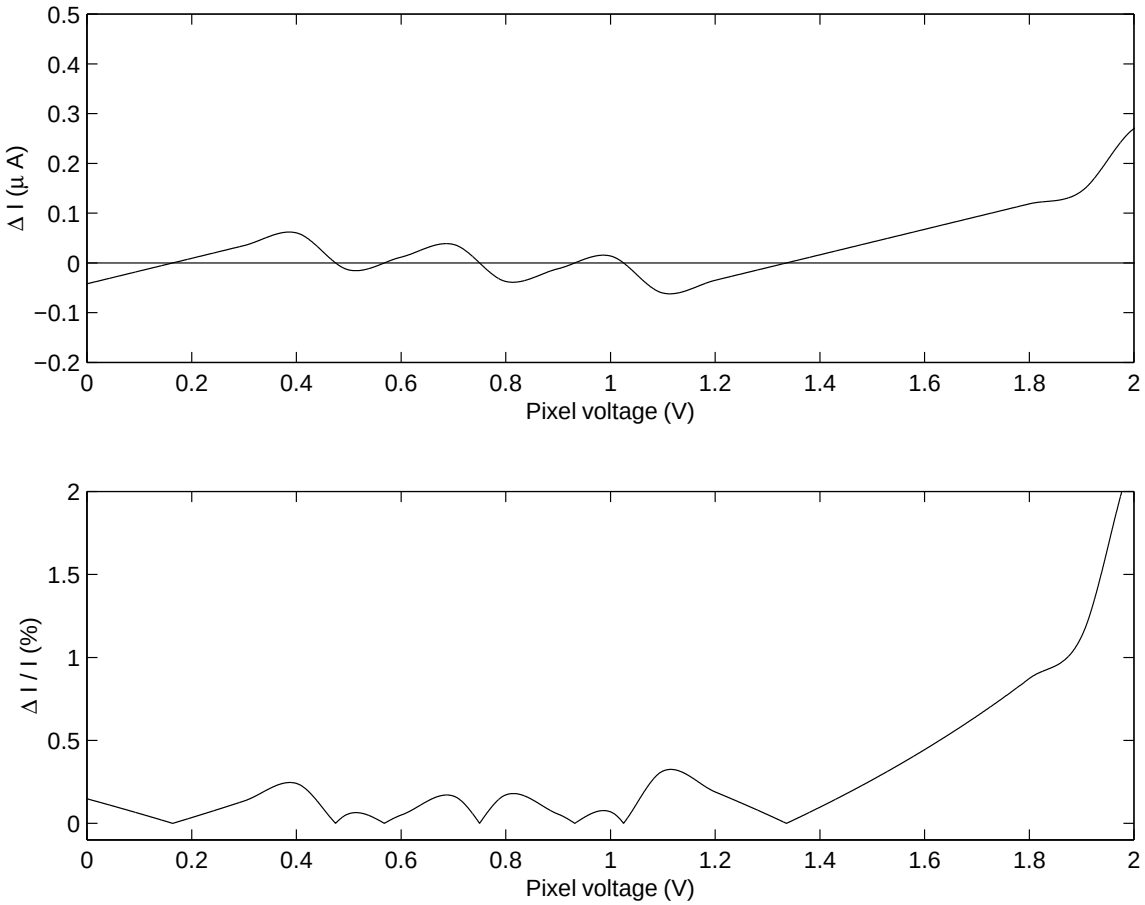


Figure 6.11: Voltage to current conversion simulation output difference plot (top) and fractional error (bottom). Range of operation is $[0 : 1.5V]$.

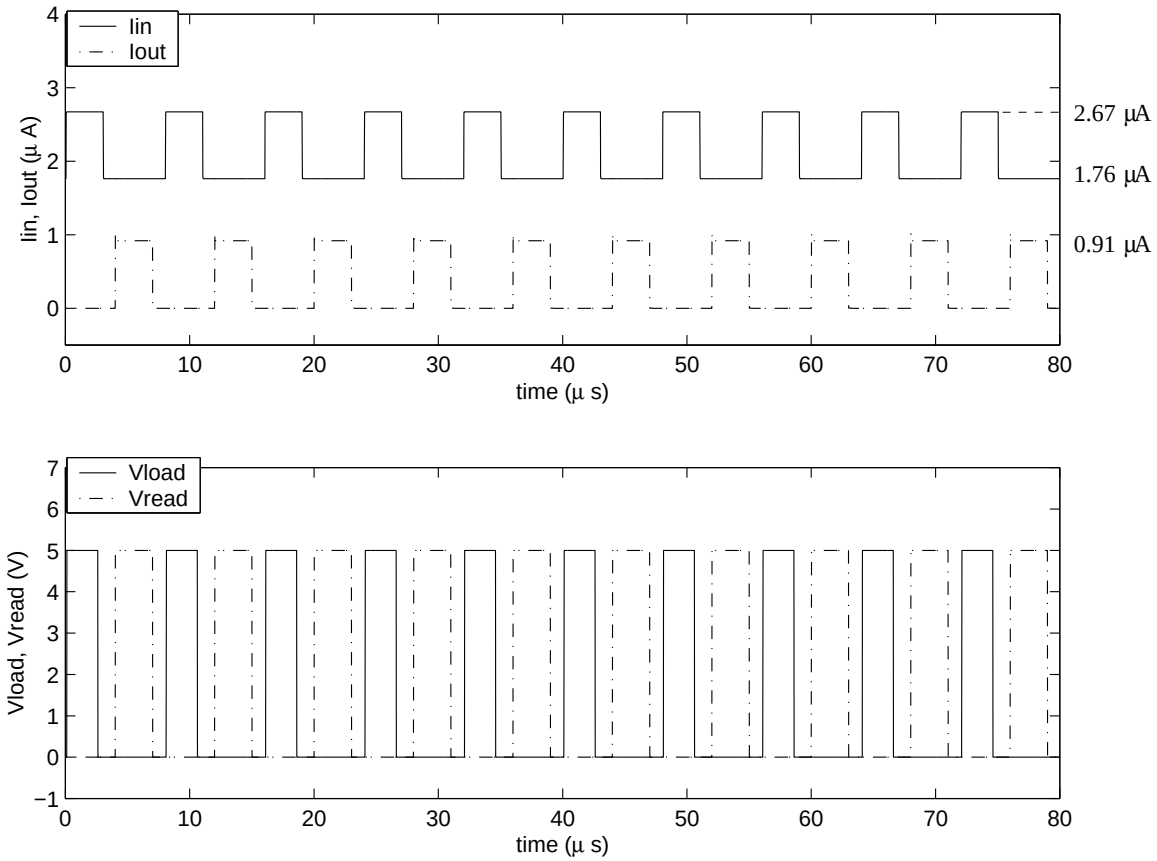


Figure 6.12: Pixel readout circuit simulation with fixed-pattern noise reduction. Top: Input current from voltage to current conversion (solid line. high value is exposed pixel and low value is reset level) and difference between exposed and reset level during readout cycle (dashed line). Bottom: current memory load (solid line) and read out (dashed line) control signals

the column readout signal chain. A current memory, as described in section 6.3, samples the exposed pixel value onto a capacitor and provides the saved value to the output interface while subtracting the pixel reset level from it, so the difference is independent of the pixel reset reference voltage, reducing the offset from the pixel. Further sampling is then homogeneous among the pixels and the range of signals are more accurately comparable.

The simulation setup for this two-step function consists in sampling alternatively a high and a low current level, as provided by the voltage to current converter. The higher current represents an exposed pixel response and is saved on the capacitor in the current memory. The lower current simulates the pixel reset level. When the current memory is in read mode, the difference between the two values is sampled at the output. A switch at the output

ensures that no current leaks out. An example with random values (chosen to be in the range of operation defined by the converter) is shown in figure 6.12. The output (dashed line) rises to a level equal to the difference between the high and the low input levels.

6.5 Multipliers

After going through the voltage to current converter and the fixed pattern noise reduction circuits of the readout circuit, the pixel information becomes one of the operands of the multiplier. The other operand being the digital kernel stored in an on-chip memory cell. The multiplier is built as a multiplying DAC with the digital operand controlling a binary-scaled ladder, as described in section 5.3. Because the kernel is fixed,[1] the multiplier effectively acts as a single-input cell. The advantage of having a fixed kernel is that no switching occurs, so no charge feed-through is generated which would affect the accuracy of the multiplication.

To implement the multiplication using equation 5.8, a series of cascode current mirrors were arranged as in figure 5.14 with identical but separate blocks for the lowest and most-significant bits. The regrouping is done by scaling the MSB by a gain of sixteen using a scaled current mirror, as in figure 5.16, rather than by using a time-controlled switch. Time-controlled switches require accurate clock division which is best achieved internally but is not convenient to provide externally. External control is important for a test chip where testability is critical. A time-scaled circuit was built on a separate test circuit and yielded encouraging results, but suffered from inaccurate clock ratios in the test setup.

The current mirrors used for the multiplier are identical to those described in section 6.2. Because there are so many of them, the current mirrors need to be well matched. Proper matching ensures that the scaled current accurately reflects the multiplication and maintains monotonicity when going from LSB to MSB. Layout considerations were crucial in this step and are the subject of the discussion of section 7.3.2. Those considerations include large $\frac{W}{L}$ for the mirrors, placement of elementary multiplying cells on a regular grid and close proximity of the controls for the two halves of the multipliers.

A simulation of the multiplier with varying input current and a number of different kernel values is shown in figure 6.13. It demonstrates how the output current levels when reaching $20\mu A$, which occurs when the voltage drop across both the N-type and P-type

[1] The kernel does not change during a frame readout, and is therefore considered fixed for that time.

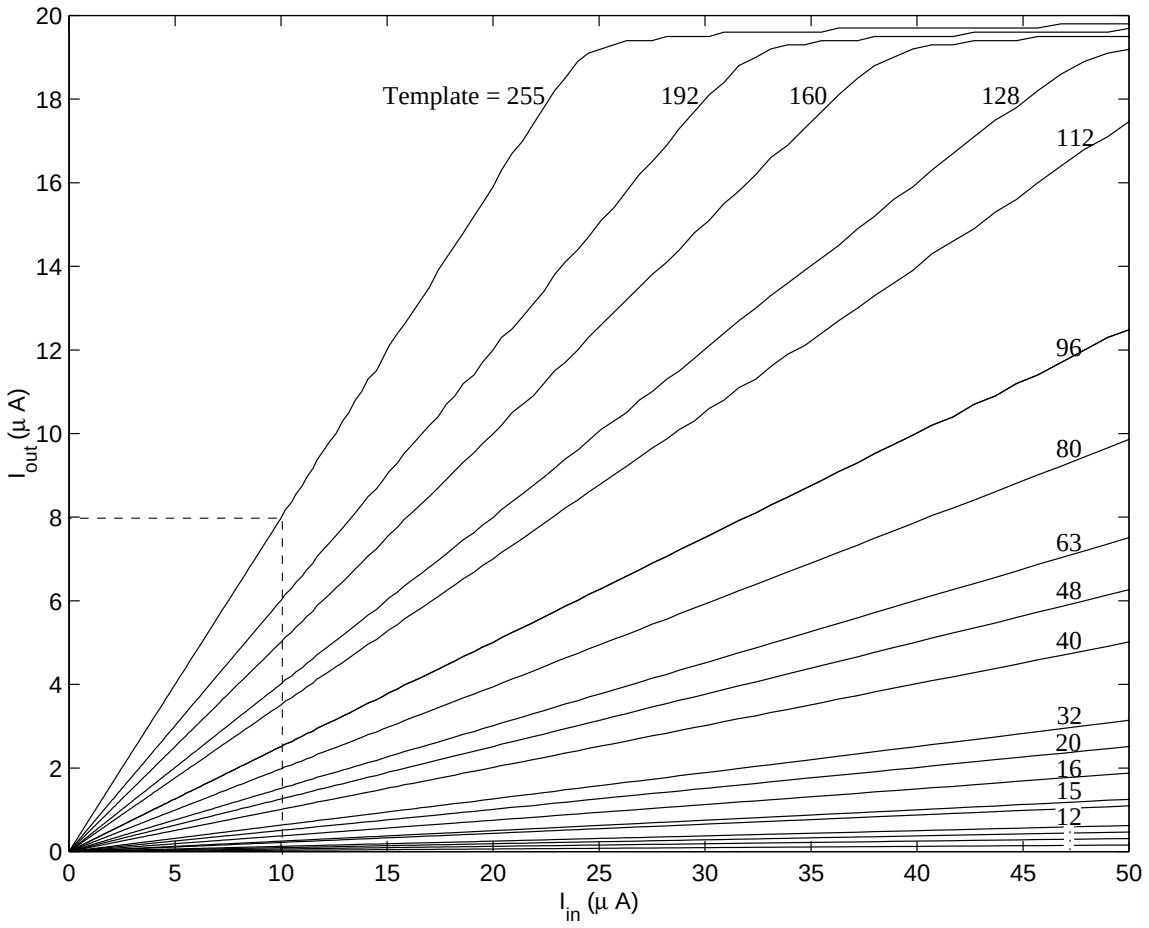


Figure 6.13: Multiplying DAC simulation output with varying input current (X-axis) for several kernel. Dashed area indicated the region of interest.

cascode current mirrors is too large and a larger supply voltage would be required to allow for sufficient headroom. This issue illustrates the necessity for the computation circuit to maintain a range of currents that is appropriate to all elements within operating conditions.

The current input of the multiplier is the output of the pixel readout circuit described in section 6.4. The interface design plans for a current range of $[0 : 10\mu A]$ to pass to the multiplier. The corresponding range in the multiplier simulation of figure 6.13 is inside the dotted rectangle. It is also magnified in figure 6.14 to show the linearity of the circuit output with respect to the current input. Switching the axis from current input to kernel value yields a similar plot. Simulation predicts a linear multiplication with respect to both inputs for the range of currents considered.

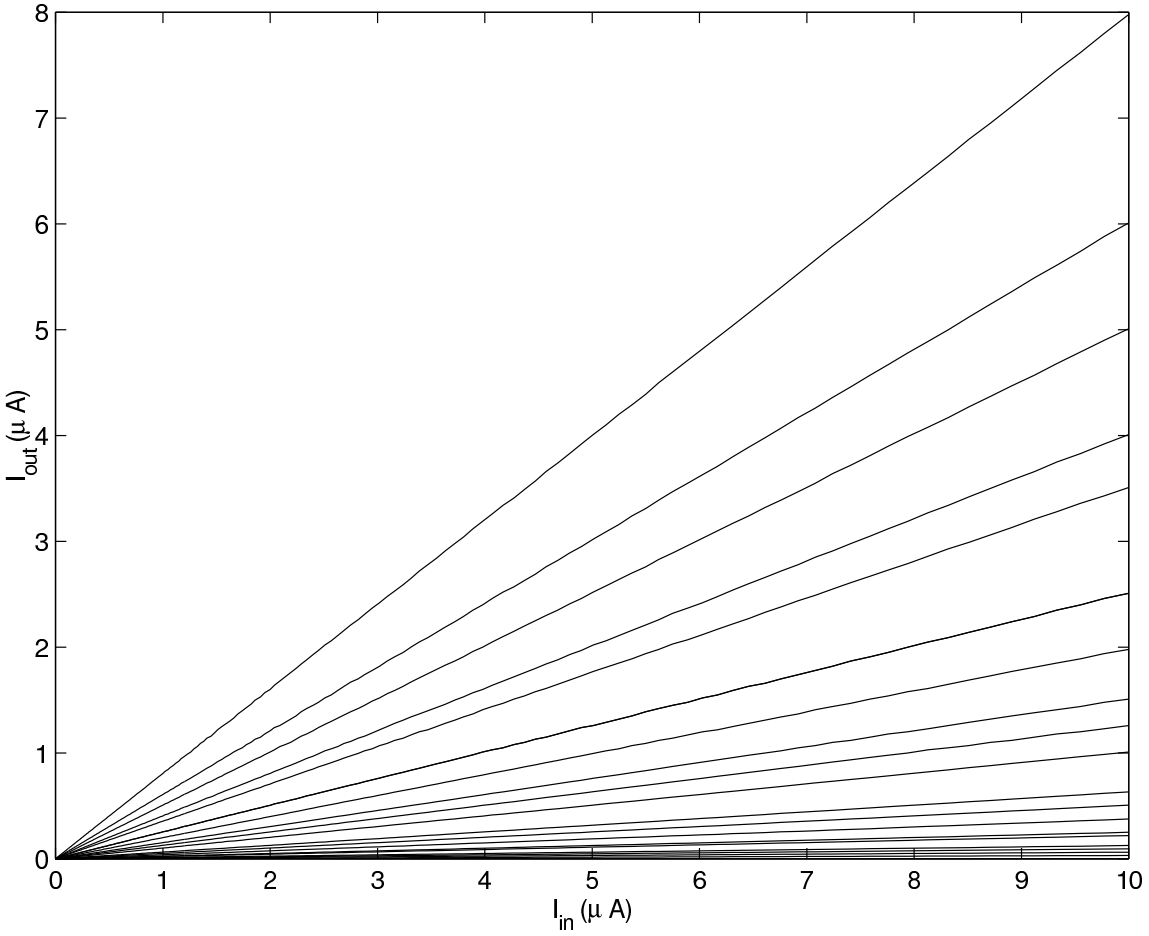


Figure 6.14: Multiplying DAC simulation output restricted to the range of signal used

6.6 Accumulators

The multiplication units generate the products and the inner sum of the convolution equation 5.9. Once these partial products are available, the accumulators are in charge of combining them with the partial products created from previous rows and saving them to be combined with products from future rows. The pipeline accumulator structure of section 5.4 is used for this task. The pipeline is made of a series of similar stages which only differ by the scaling factors of their inputs. The first pipeline stage is also different from the others since it only handles one input, a partial product from the multiplier, instead of two with the output of the previous stage. The simplified block diagram of figure 5.17 shows the operation of the pipeline.

The following paragraphs will show the simulation of both the first pipeline stage, a single accumulator cell, as well as a double accumulator cell, representative of all the other stages.

6.6.1 Single cell

Because it is the initialization stage, the first stage of the pipeline accumulator only accepts one input, from the multiplier. Furthermore, it does not perform any scaling of the signal. It only serves as a buffer and memory cell so the pipeline is initiated in a consistent manner. What is left is a current mirror at both the input and the output and a current memory cell. Those elements are identical to those described and simulated above, in sections 6.2 and 6.3.

Details of the single accumulator cells are described in section 5.4.2, with the simplified schematic in figure 5.18. Because it is very similar to a current memory, the simulation also looks very similar: the same setup is used in the single cell simulation of figure 6.15 with the load/read sequence shown in the bottom plot. The top plot shows how the output current matches the input on each read cycle. The operating range is also similar to that of the current memory so linearity of the response is also good in the range of $[0 : 10\mu A]$

6.6.2 Double cell

All other stages of the accumulator are built on the model of the circuit shown in section 5.4.3. Each of the two inputs is mirrored and scaled so the sum is the average of all the

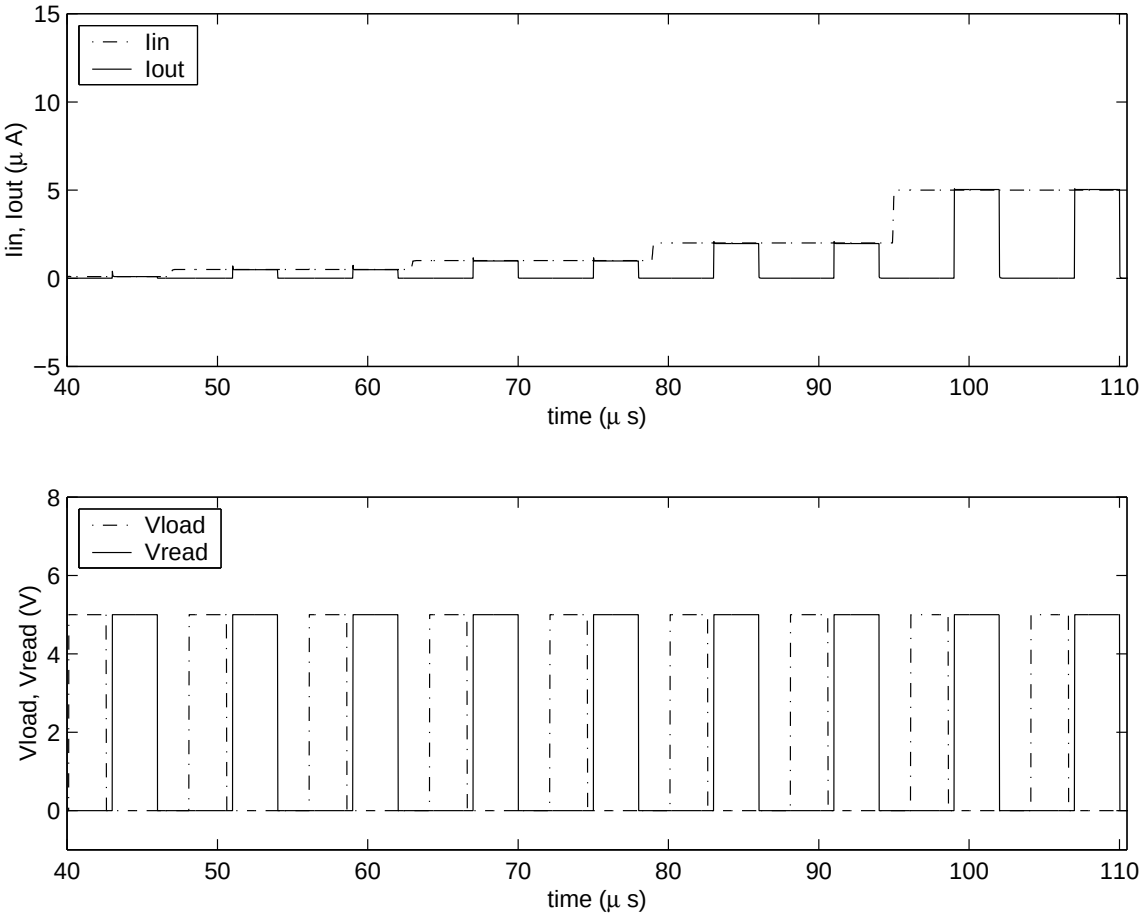


Figure 6.15: Single accumulator cell simulation. Input and output currents (top) and chronogram (bottom)

inputs to the pipeline so far. Because they are made of two current memories and handle two inputs, they are referred to as double cells. The scaling performed is asymmetric with respect to the inputs: the input from the previous stage carries more weight than the new incoming partial product so the rolling average of equation 5.12 is always correct for the current stage.

Since double cells receive inputs from the same source as single cells, the interfaces are similar and handle the same range of currents. This appears in the simulation shown in figure 6.16 which naturally resembles that of the single cell. In this example, the fourth stage of the pipeline was simulated and compared to the expected output from the equation. The output current is a function of the two inputs $I_{previous}$ and I_{in}. $I_{previous}$ comes from the previous pipeline stage and I_{in} from the multiplier. We expect the output of the fourth stage to be: $I_{out} = \frac{3}{4}I_{previous} + \frac{1}{4}I_{in}$.

As for the single cell simulation, the sequence of events is a succession of load and read cycles. The chronogram of figure 6.16 (bottom) shows the cycles. The top plot compares the simulated output to the expected one. They are equal during each read cycle.

6.6.3 Pipeline

When all stages are connected end to end, the complete pipeline accumulator can also be simulated. Every input of the pipeline is provided by the multiplier, so the interface with each stage is always the same. When a new window starts being processed, the accumulation works its way through the pipeline in as many cycles as pipeline stages. In the convolution chip, nine stages form the pipeline accumulator so the result is available on the ninth cycle. The propagation of the signal with the convergence towards the response to a set of inputs is shown in figure 6.17. The same chronogram is used as in the individual stage simulations. Assuming the pipeline current memories are initialized to zero (ground), the output rises at every cycle since one extra input is added to the overall sum. After the ninth stage, the same value is read out until the inputs change.

A major expectation of the pipeline accumulator is the linearity of the response. Equation 5.12 predicts that the averaged sum is linear with respect to every input. In particular, it should show as an identity function when all nine inputs are set to be equal. In this case, the output I_{out} follow the input I_{out}. With the average, the accumulation equation reduces to: $I_{out} = \frac{9 \times I_{in}}{9} = I_{in}$. Figure 6.18 shows the response over the range of currents of interest.

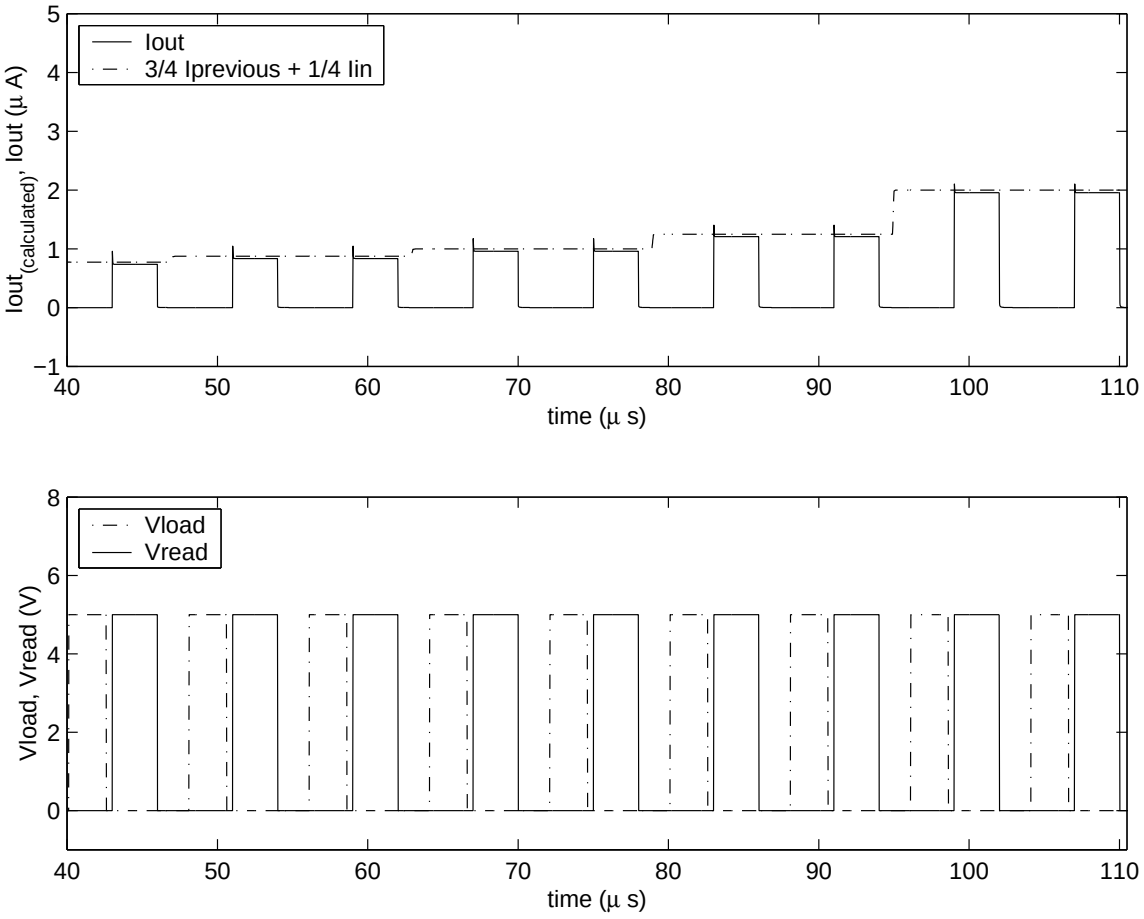


Figure 6.16: Double accumulator cell simulation. Expected (calculated from input) and simulated output currents (top) and chronogram (bottom)

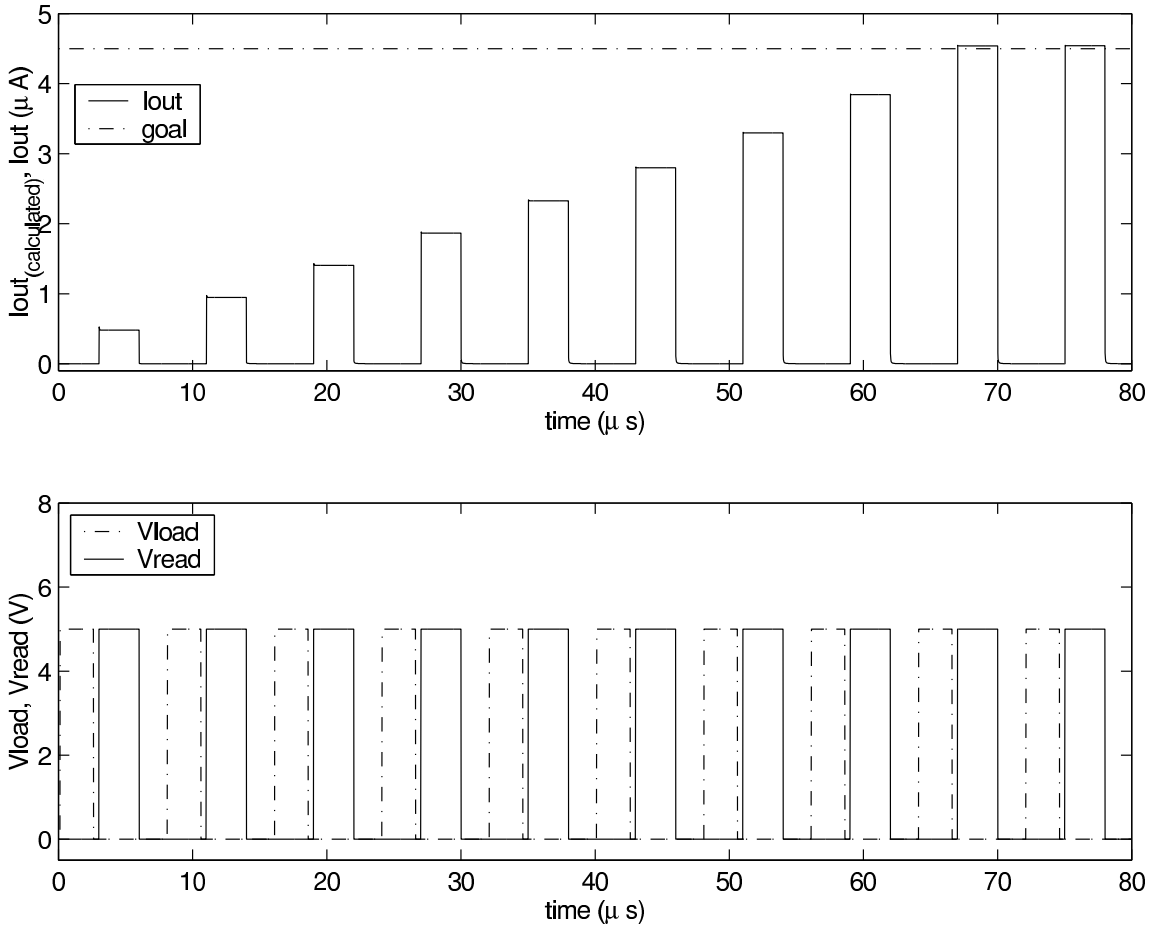


Figure 6.17: 9-stage accumulator with all identical inputs converges on the ninth step

The dashed line represents the linear fit of the simulated data. The absolute and relative difference plots are available in figure 6.19. They predict uncertainties of up to 2% to 4% can be expected from the complete pipeline.

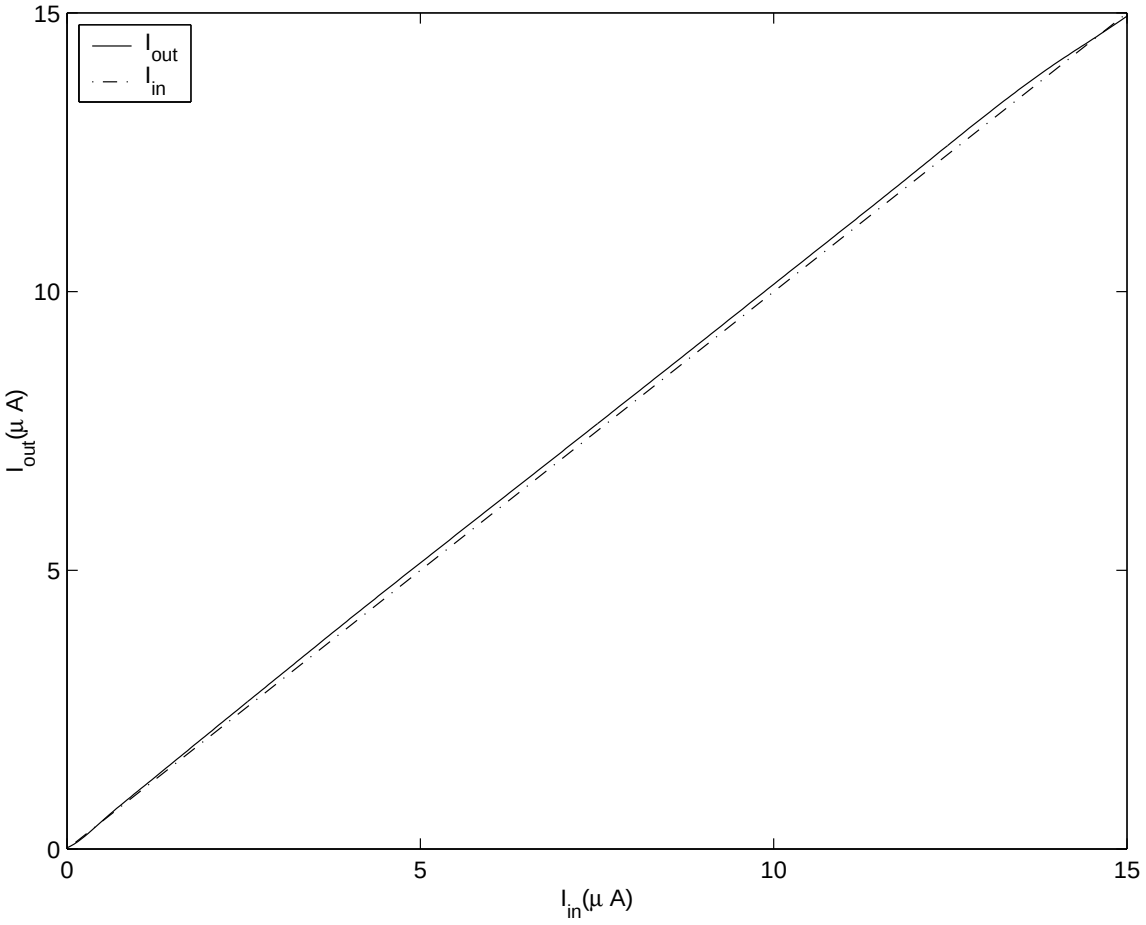


Figure 6.18: 9-stage accumulator with all identical inputs I_{in}

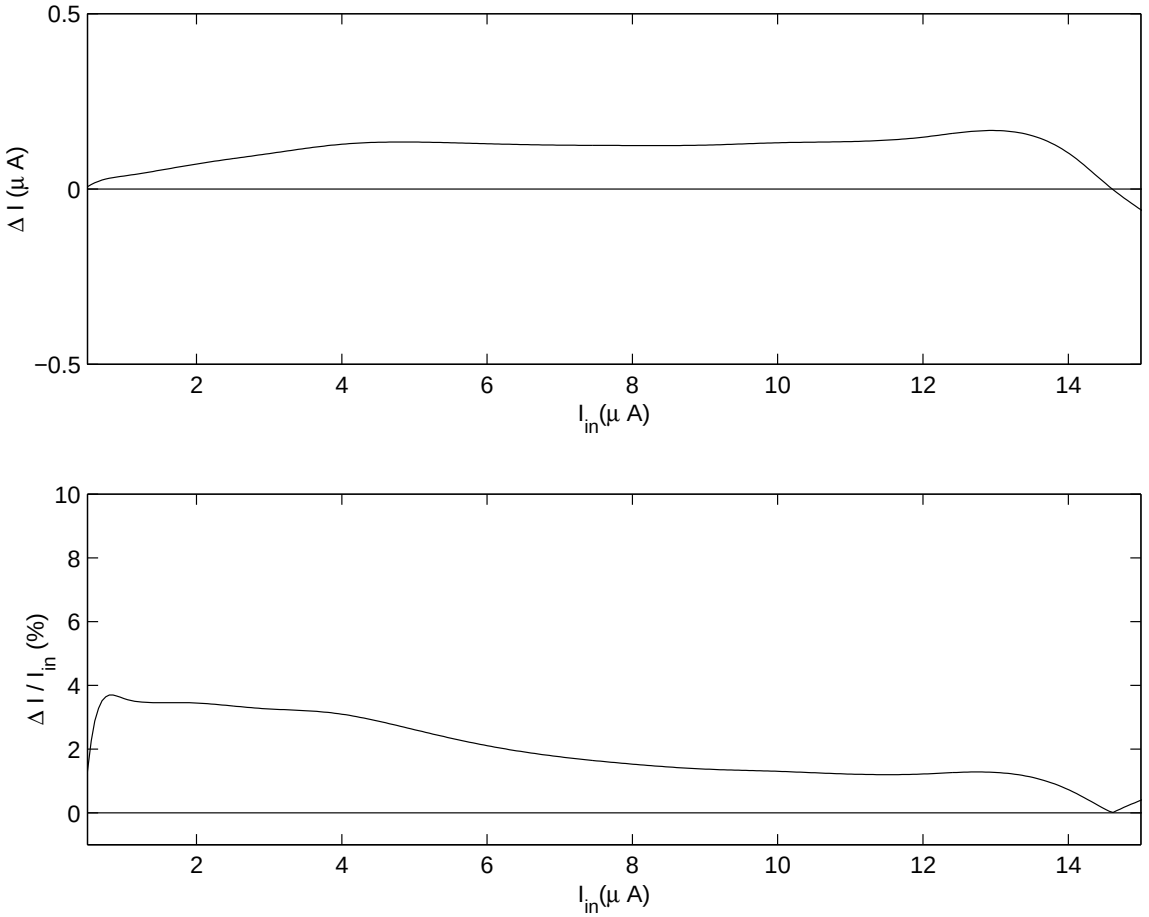


Figure 6.19: 9-stage accumulator with all identical inputs I_{in}: difference plot (top) and fractional error (bottom)

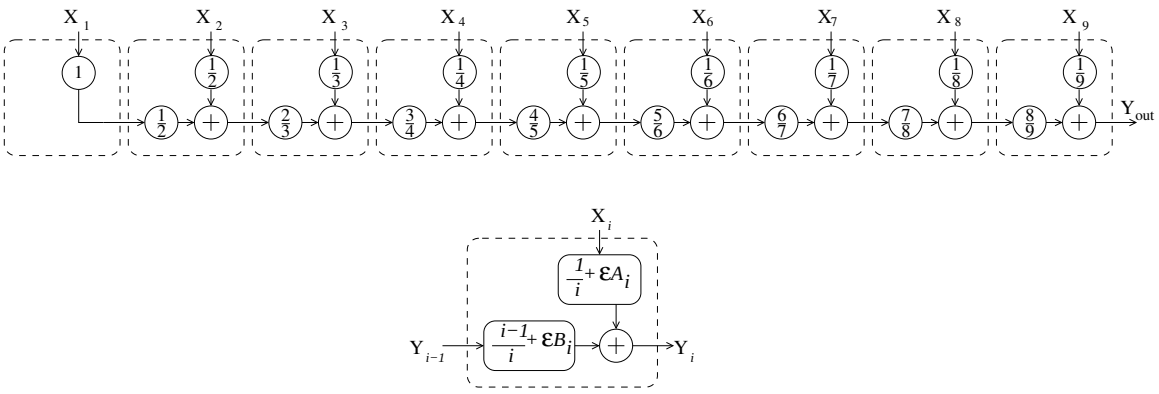


Figure 6.20: Accumulator block diagram (top) and sources of systematic errors in stage i (bottom)

6.7 Uncertainties

6.7.1 Accumulator propagation of errors

The pipeline making up the accumulator is a non-symmetrical structure as its nine inputs enter at different stages. If the pipeline were perfect, the uncertainties in the input variable would propagate evenly to the output and contribute equally to the output uncertainty. Because the amplifiers in each stage are not perfect and suffer from both mismatch and noise of the transistors, the actual amplification includes both a systematic and a random error component. The random component, due to noise will be examined in section 6.7.2 on circuit noise, while this section will look at how the systematic errors propagate in the pipeline and affect the final result. Intuitively, the first inputs to the accumulator go through more of these imperfect stages and therefore propagate more uncertainty than the last ones. On the other hand, the uncertainty from the last stage will not go through as many scaling stages and will therefore not be attenuated. These two effects are shown in figure 6.20 (top) where the scaling which introduce systematic errors appear. Figure 6.20 (bottom) details one stage with the scaling factors and the systematic errors associated to them.

Let A_i and B_i the two amplifiers of the stage i, $A_i = \frac{1}{i}$ scaled the pixel input X_i, and B_i the input from the previous stage of the pipeline Y_{i-1}. The systematic errors associated to those amplifiers are ε_{A_i} and ε_{B_i}. The real amplifier values (including all uncertainties) are noted $A_{X_{r_i}}$ and $A_{Y_{r_i}}$.

The first stage is unique in the sense that it only takes one input X_0 and has unity gain.

The output can then be expressed as:

$$Y_1 = A_{X_{r_0}} \cdot X_0 = A_{X_0} \cdot X_0 + \epsilon_{X_0} \cdot X_0,$$

$$Y_1 = X_0 \cdot (A_{X_0} + \varepsilon_{X_0}) .$$

All the subsequent stages are built on the same template shown in figure 6.20(bottom) with two inputs: X_i from the column and Y_i from the previous pipeline stage. The amplifiers are the only distinction between them.

For any stage i, the output Y_{i+1} in terms of the inputs X_i and Y_i is:

$$Y_i = Y_{i-1} (A_{Y_i} + \varepsilon_{Y_i}) + X_i (A_{X_i} + \varepsilon_{X_i})$$

$$\Rightarrow Y_i = \underbrace{A_{X_i} \cdot X_i + A_{Y_i} \cdot Y_{i-1}}_{ideal\ response} + \underbrace{\varepsilon_{X_i} \cdot X_i + \varepsilon_{Y_i} \cdot Y_{i-1}}_{error\ introduced\ in\ i^{th}\ stage} .$$

The amplifier values on both inputs are a function of the pipeline stage: $A_{X_i} = \frac{1}{i}$ and $A_{Y_i} = \frac{i-1}{i}$. With some algebra, we can directly express the output of any stage of the pipeline as a function of the inputs:

$$Y_i = \sum_{k=1}^{i} \left(\left(\frac{1}{k} + \varepsilon_{X_k} \right) \cdot X_k \prod_{l=k}^{i-1} \left(\frac{l}{l+1} + \varepsilon_{Y_{l+1}} \right) \right) .$$

In particular, the output of the complete pipeline is given by:

$$Y_9 = \sum_{k=1}^{9} \left(\left(\frac{1}{k} + \varepsilon_{X_k} \right) \cdot X_k \prod_{l=k}^{8} \left(\frac{l}{l+1} + \varepsilon_{Y_{l+1}} \right) \right) . \tag{6.4}$$

From the expression of equation 6.4, we can extract the influence of the errors depending on where they are introduced in the pipeline. If we only consider the first order, the contribution to the final output Y_9 of the amplifiers A_X is:

$$\varepsilon_{Y_9}|_{inputX} = \sum_{i=1}^{9} \frac{i}{9} (\varepsilon_{X_i} \cdot X_i) . \tag{6.5}$$

Equation 6.5 shows that the closer the stage to the output, the larger the contribution of the systematic error it introduces. While it is beneficial to attenuate the effects of the systematic errors from the first stages, it comes at the expense of symmetry on the

pipeline which is undesirable since we no longer expect these errors to compensate. A similar observation can be made from looking at the uncertainties from the A_Y amplifiers. Their contribution are also a function of their position and lessen as they get closer to the beginning.

6.7.2 Circuit noise

As with any circuit, the convolution chip is subject to uncertainties and errors due to circuit noise. Each transistor in the chip is subject of such perturbation. Those noise sources can be modeled and their effect on the output of the complete circuit can be estimated by propagating all the intermediate noise sources through the entire circuit. While it may seem like an overwhelming task, the analysis can be broken down to a few elementary blocks which are easily managed and the signal chain can be simplified by recognizing similarities and assuming a worst-case scenario at every stage. The diagram shown in figure 6.21 will be used as the basis for the calculations. The dotted boxes isolate the elementary building blocks that will be studied first:current mirrors, current memories and pipeline stages.

6.7.2.1 Current mirror

The most basic structure used as an elementary building block is the simple current mirror shown in figure 6.22(a). Ideally, when both transistors M_1 and M_2 are geometrically identical, the input and output currents are equal: $I_2 = I_1$. The noise introduced in this circuit can be observed as a variation in this equality. We introduce α where $I_2 = \alpha I_1$. Since only ratios of sizes are relevant here, we carry all discrepancies into the widths of the transistors, such that the lengths are equal: $L_1 = L_2 = L$ but W_1 and W_2 are not assumed to be. Consequently, no assumption is made for the gains g_{m_1} and g_{m_2}.

$$\left.\begin{aligned} g_{m_1} &= \frac{\partial I_1}{\partial V_{GS_1}} = \sqrt{2 I_1 C_{ox} \frac{W_1}{L}} \\ g_{m_2} &= \frac{\partial I_2}{\partial V_{GS_2}} = \sqrt{2 I_2 C_{ox} \frac{W_2}{L}} \end{aligned}\right\} \Rightarrow \frac{g_{m_1}}{g_{m_2}} = \sqrt{\frac{I_2 W_2}{I_1 W_1}}.$$

Also, by definition,

$$\alpha = \frac{I_2}{I_1} = \frac{W_2}{W_1} \Rightarrow \alpha = \frac{g_{m_2}}{g_{m_1}}.$$

The noise on the output current I_2 is then:

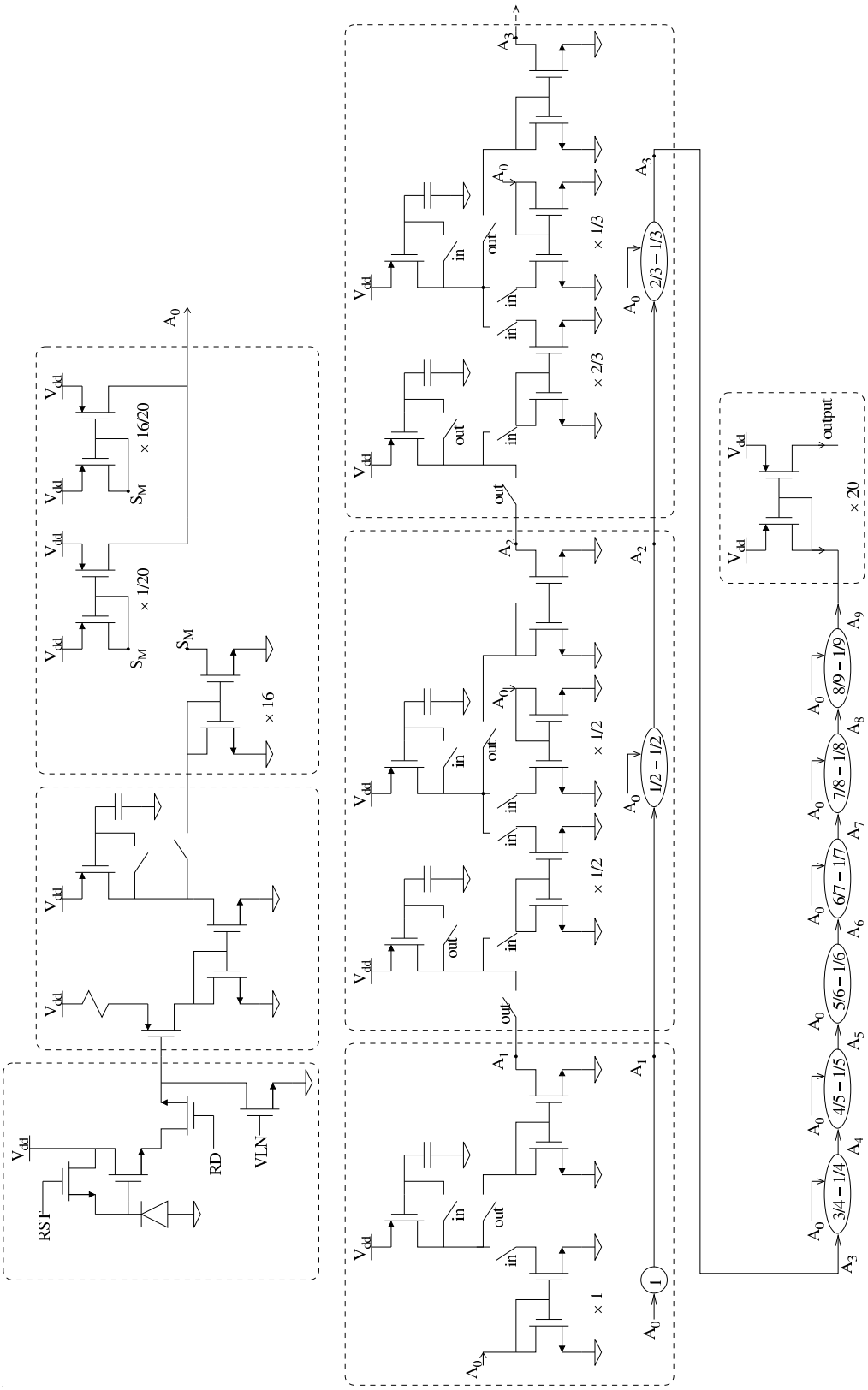
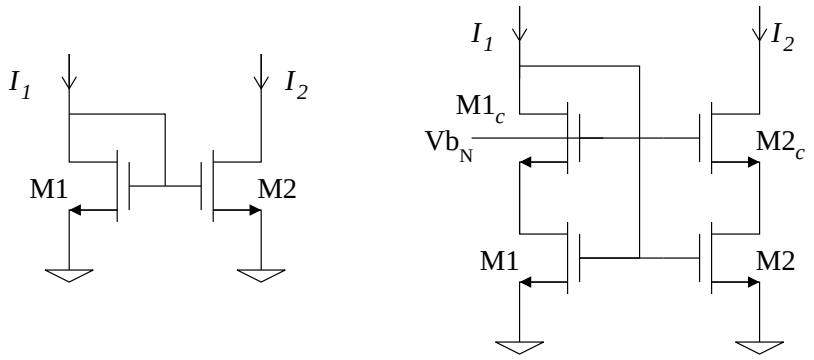


Figure 6.21: Signal chain for the noise analysis



(a) Simple current mirror (b) Cascode current mirror

Figure 6.22: Simple (a) and cascode (b) current mirrors.

$$Sv_{I_2} = \frac{8}{3}kTg_{m_1}\alpha^2 + \frac{8}{3}kTg_{m_2}$$

$$Sv_{I_2} = \frac{8}{3}kTg_{m_2}\left(\frac{g_{m_1}}{g_{m_2}}\alpha^2 + 1\right)$$

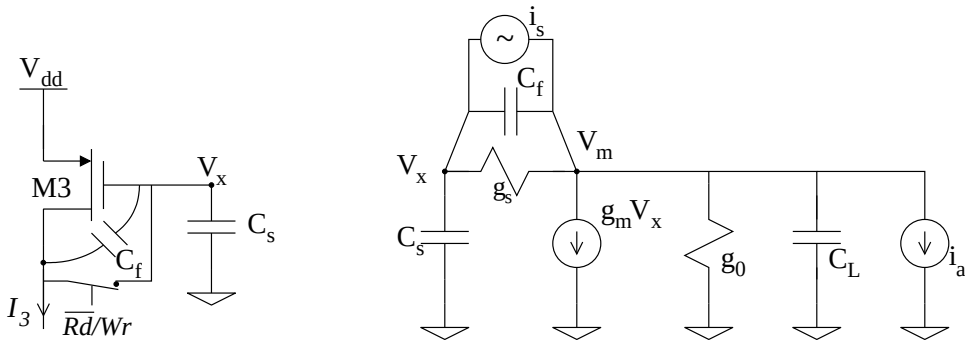
$$\Rightarrow Sv_{I_2} = \frac{8}{3}kTg_{m_2}\left(\alpha + 1\right). \tag{6.6}$$

Typical parameters from the fabricated chip process allow to estimate the noise level to expect in the final circuit. In the specific case of the simple current mirror, numerical application of equation 6.6 give $Sv_{I_2} = 0.73pA/\sqrt{Hz}$. The small number reflects the compactness of the structure, the small capacitance, and the absence of sampling in this cell.

The cascode current mirror of figure 6.22(b), which is used in the chip design rather than the simple current mirror, yields to the same noise equation as the influence of the cascode transistor changes the noise in the output branch to $Sv_{I_2} = \frac{8}{3}kT\left(g_{m_2} + \frac{g_{m_{2_c}}}{A}\right)$ where A is the gain. Since $A \gg g_{m_2}$, the expression reduces to the one above. Simple stage current mirrors will therefore be used from now on in the noise analysis.

6.7.2.2 Current memory

The other main building block in the chip is the current memory cell. When in *Read* mode, it behaves similarly to the output of the current mirror seen above. However, when



(a) Current memory (write mode)

(b) Small-signal model

Figure 6.23: Simple current memory (a) and its small-signal model (b)

a current is written to it, it differs from that model: the sampling on the capacitor through the switch affects the saved information and noise is introduced on the sampled voltage V_m. The cell in $Write$ mode is shown in its simplified version in figure 6.23(a). The small signal model of figure 6.23(b) is used to determine the noise. The main perturbations of the sampled voltage come the drain to gate capacitance C_f, the switch resistance represented by its admittance g_s and current fluctuations modeled as the two current sources i_f and i_a which are represented in the circuit diagram. The output of the cell is a current which depends directly on the sampled voltage. It is the noise of this voltage that we will analyze here.

Before the switch is closed, the charge at V_x is:

$$Q_x = C_s \cdot \hat{V}_x + C_f \cdot (\hat{V}_x - \hat{V}_m).$$

When it is toggled, the charges distribute and $V_x = \frac{Q_x}{C_s} \Rightarrow Qx = V_x \cdot C_s$. Because of charge conservation, those two expressions of Q_x must be equal, and therefore:

$$V_x = \hat{V}_x \frac{(C_s + C_f)}{C_s} - \hat{V}_m \frac{C_f}{C_s}. \tag{6.7}$$

The noise at the node V_x can therefore be expressed in terms of $\hat{V}_x$ and $\hat{V}_m$:

$$\langle V_x^2 \rangle = \left\langle \left[\hat{V}_x \frac{(C_s + C_f)}{C_s} - \hat{V}_m \frac{C_f}{C_s} \right]^2 \right\rangle.$$

The noise sources i_s and i_a being independent, we can study them separately and apply the superposition principle to merge the results. We determine the transfer functions with respect to both sources from the schematic of figure 6.23(b):

$$\frac{\hat{V}_m}{i_s} = \frac{1 + \frac{g_m}{jC_s\omega}}{(g_s + jC_f\omega)(g_0 + jC_l\omega)\left(\frac{1+\frac{g_m}{jC_s\omega}}{g_0+jC_l\omega} + \frac{1}{jC_s\omega} + \frac{1}{g_s+jC_f\omega}\right)}, \tag{6.8}$$

$$\frac{\hat{V}_m}{i_a} = \frac{1 + \frac{C_f}{C_s} + \frac{g_s}{jC_s\omega}}{(g_s + jC_f\omega)(g_o + jC_l\omega)\left(\frac{1+\frac{g_m}{jC_s\omega}}{g_0)+jC_l\omega} + \frac{1}{jC_s\omega} + \frac{1}{g_s+jC_f\omega}\right)}, \tag{6.9}$$

$$\frac{\hat{V}_x}{i_s} = \frac{\frac{1}{g_s+jC_f\omega} + \frac{1}{gs}}{\frac{1}{g_s+jC_f\omega} + \frac{1}{g_0+jC_lw} + \frac{1}{g_s}}, \tag{6.10}$$

$$\frac{\hat{V}_x}{i_a} = \frac{1}{(g_0 + jC_l\omega)\left(1 + \frac{g_m}{g_0+jC_l\omega} + jC_s\omega\left(\frac{1}{g_0+jC_l\omega} + \frac{1}{g_s+jC_f\omega}\right)\right)}. \tag{6.11}$$

Applying the superposition principle to equation 6.7, we obtain from equations 6.8 to 6.11 an expression for the transfer functions $H_s = \frac{V_x}{i_s}$ and $H_a = \frac{V_x}{i_a}$:

$$\begin{aligned} H_s = &\frac{(C_f + C_s)\left(\frac{1}{g_s+jC_f\omega} + \frac{1}{g_s}\right)}{C_s\left(\frac{1}{g_s+jC_f\omega} + \frac{1}{g_o+jC_l\omega} + \frac{1}{g_s}\right)} \\ &- \frac{C_f\left(1 + \frac{g_m}{jC_s\omega}\right)}{C_s(g_s + jC_f\omega)(g_o + jC_l\omega)\left(\frac{1+\frac{g_m}{jC_s\omega}}{g_o+jC_l\omega} + \frac{1}{jC_s\omega} + \frac{1}{g_s+jC_f\omega}\right)}, \end{aligned} \tag{6.12}$$

$$\begin{aligned} H_a = &\frac{C_f + C_s}{C_s(g_o + jC_l\omega)\left(\frac{g_m}{g_o+jC_l\omega} + jC_s\omega\left(\frac{1}{g_o+jC_l\omega} + \frac{1}{g_s+jC_f\omega}\right) + 1\right)} \\ &- \frac{C_f\left(\frac{C_f}{C_s} + \frac{g_s}{jC_s\omega} + 1\right)}{C_s(g_s + jC_f\omega)(g_o + jC_l\omega)\left(\frac{1+\frac{g_m}{jC_s\omega}}{g_o+jC_l\omega} + \frac{1}{jC_s\omega} + \frac{1}{g_s+jC_f\omega}\right)}. \end{aligned} \tag{6.13}$$

The noise sources i_a and i_s being uncorrelated, the noise power of V_x is the sum of the powers contributed by the noise sources separately:

$$< V_x^2 >= \left[H_a^2 \cdot S_{i_a} + H_s^2 \cdot S_{i_s}\right] \cdot BW, \tag{6.14}$$

where BW is the frequency spectrum of the signal, or bandwidth. In the case of this circuit. S_{i_a} and S_{i_s} are the spectral density of the two noise sources.

$$BW = \frac{\omega_c}{4} = \frac{g_m}{4C_s} \; ; \; S_{i_a} = 4kT\frac{2}{3}g_m \; ; \; S_{i_s} = 4kT\frac{2}{3}g_s$$

Substituting into equation 6.14, we get:

$$< V_x^2 >= \left[H_a^2 \cdot 4kT\frac{2}{3}g_m + H_s^2 \cdot 4kT\frac{2}{3}g_s\right] \cdot \frac{g_m}{4C_s}$$

$$\Rightarrow < V_x^2 >= 4kT\frac{2}{3}\left(g_m + g_s\right)\left[H_a^2 + H_s^2\right] \cdot \frac{g_m}{4C_s}. \tag{6.15}$$

Similarly to the numerical application that equation 6.6 yielded, typical process parameters are used to estimate the noise level in the memory cell. Equation 6.15 predicts a noise level of $V_x = 101.4\mu V$ on the memorized voltage. The higher noise (compared to the noise obtained for the simple current memory) can be attributed to the higher capacitance in the cell as well as the sampling of the voltage, which is an necessary feature of the memory cell.

The current memory studied above is the building block for most of the computation units in the convolution chip. It was therefore important to study the noise source in that cell. The other critical element is the pipeline in which each stage produces circuit noise. The noise signal chain of figure 6.21 illustrates the similarities of each stage. The stages are based on the two cells analyzed above: the cascode current mirrors and the current memories. The noise produced is therefore the combination of the circuit noise sources from three current mirrors and two current memory cells.

6.8 Conclusion

The successful design of a complex circuit such as a computational imager relies on a number of techniques that must be followed in order to produce a working chip. The circuits analysis of chapter 5 already separated the chip into a number of simple blocks that can each be studied and tuned for good performance. The challenge is then to make

choices appropriate to ensure proper operation in the operating condition of the final circuit. Interface requirements have to be met so each block interacts with its neighbors without loss of signal swing or linearity. The simulations presented in this chapter play an important role in this process as they raise the level of confidence that the architecture was properly designed.

Also relevant to the design phase is the circuit uncertainties and noise sources. Circuit noise appears in all elements, transistors and capacitors. A study on how if affects the convolution chip is presented through detailed analysis of the building blocks used throughout the chip. Uncertainties are a consequence of the mismatch of real circuit elements with respect to their expected values and with respect to each other. While all efforts are made to minimize mismatch through conservative transistor sizings and proper layout techniques, it is important to study the effect on the overall computation. The computation stage most affected by mismatch is the accumulator cell, more particularly the input and output scaling amplifiers in each stage. The effect on the signal in each stage was studied as was how errors propagate through the entire pipeline. It was shown that the influence of uncertainties depends on their position in the pipeline. It was also shown that an offset from the ideal accumulator response can be expected in the fabricated circuit.

Chapter 7

Implementation

7.1 Introduction

In the process of designing a circuit for a convolution imager, several integrated circuits were designed, primarily for the purpose of testing the various methods presented in chapter 5. Those include the performance evaluation of current-mode pixels, using charge-mode computation and time-controlled multiplying DACs.

The layout presented in this chapter is that of the latest revision. Although it still includes some test structures, it is the closest to a final design with such necessary features as a fully scalable layout.

7.2 Chip description

Once the resources of the chip being designed are understood and evaluated, the layout starts by identifying the footprint it will occupy in the radical and how it will be used efficiently. Both the current design and its use in possible future developments are taken into account in the floorplan process. The convolution chip can be functionally divided into four separable blocks.

1. **Image sensor.** A pixel array of 128×128 pixels is included in the design to provide a real image as an input of the convolution with the kernel.

2. **Kernel memory.** The second input to the convolution is a digital memory that holds the value of the kernel. The pixel array and the kernel memory are the two operands of the convolution.

3. **Mixed-signal multipliers.** Pixel-wise multiplication is at the heart of the convolution. Result addition inside each row is performed on the fly thanks to a parallel readout of the imager in this direction.

4. **Accumulators.** Partial products from each row are combined in the accumulators to reconstruct the complete convolution.

The four-block structure is preserved in the layout and yields to a signal flow oriented floorplan. The chip diagram of figure 7.1 shows the imager above the multiplier and the kernel memory on the side. They are the two inputs, and the output is placed below. The main advantage of this placement is the natural wiring of the analog information flowing vertically while the many digital lines (the kernel memory has 648 output signals) run on horizontal straight lines. Being a prototype, all control signals are externally provided, resulting in a large number of pads for the interface. With all the test fixtures and wiring added, the fabricated chip is pad limited. Bringing some of the control logic on chip, e.g., column and row binary counters, would allow for a more compact circuit at the expense of some of the testing flexibility. Sufficient space was available on the radical so a completely addressable circuit was preferred. The circuit, fabricated using a $0.5\mu m$ feature size process, measures $9.2mm \times 5.2mm$ and is bounded in a standard 84-pin PGA package.

The hashed blocks in figure 7.1 are test structures identical to the blocks they are placed next to. They allow independent testing of the various functions of the chip, used for the characterization in chapter 8. However, they are not part of the circuit and would be removed in future designs. The floor plan of the useful elements can be easily stretched to scale the design to more attractive imaging dimensions. The imager sizes in both directions, while the multipliers and accumulators only need to be arrayed to accommodate the full width. The kernel memory being unique, does not need to be scaled. Figure 7.2 shows the chip floor plan for a 1024×1024 pixel array. The computing units that were taking up most of the area when paired up with a small imager now occupy only a fraction of the chip thanks to their unidirectional scaling properties due to the row-parallel structure chosen. The overhead from adding the convolution to an imager-only circuit is less than 25%. Assuming a process similar to that used for fabricating the smaller convolution chip, we anticipate a total size of $16.9mm \times 14.0mm$. Technologies providing much smaller feature sizes are commonly available and would reduce the footprint almost proportionally.

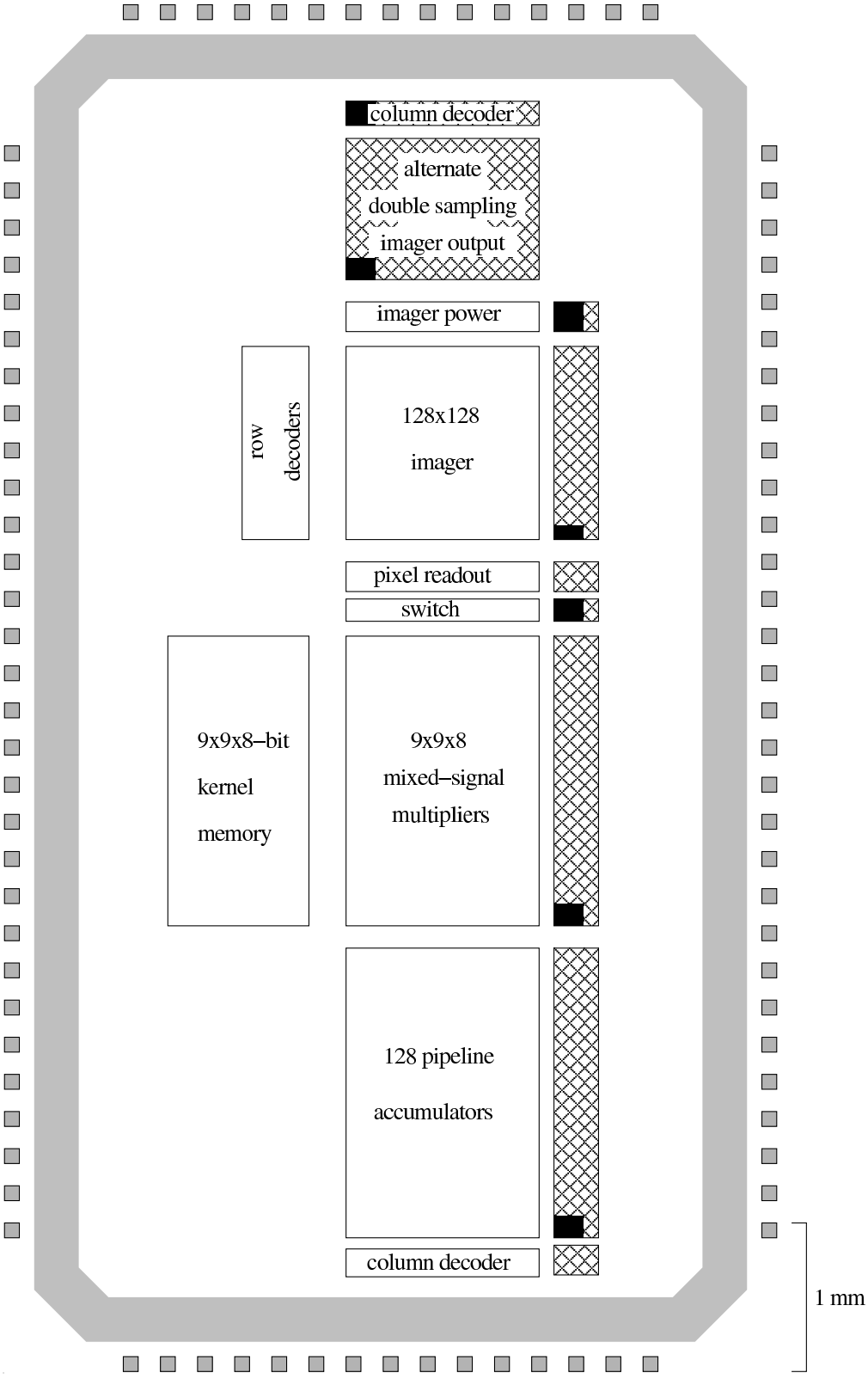


Figure 7.1: Convolution chip layout floorplan. Hashed blocks are test structures.

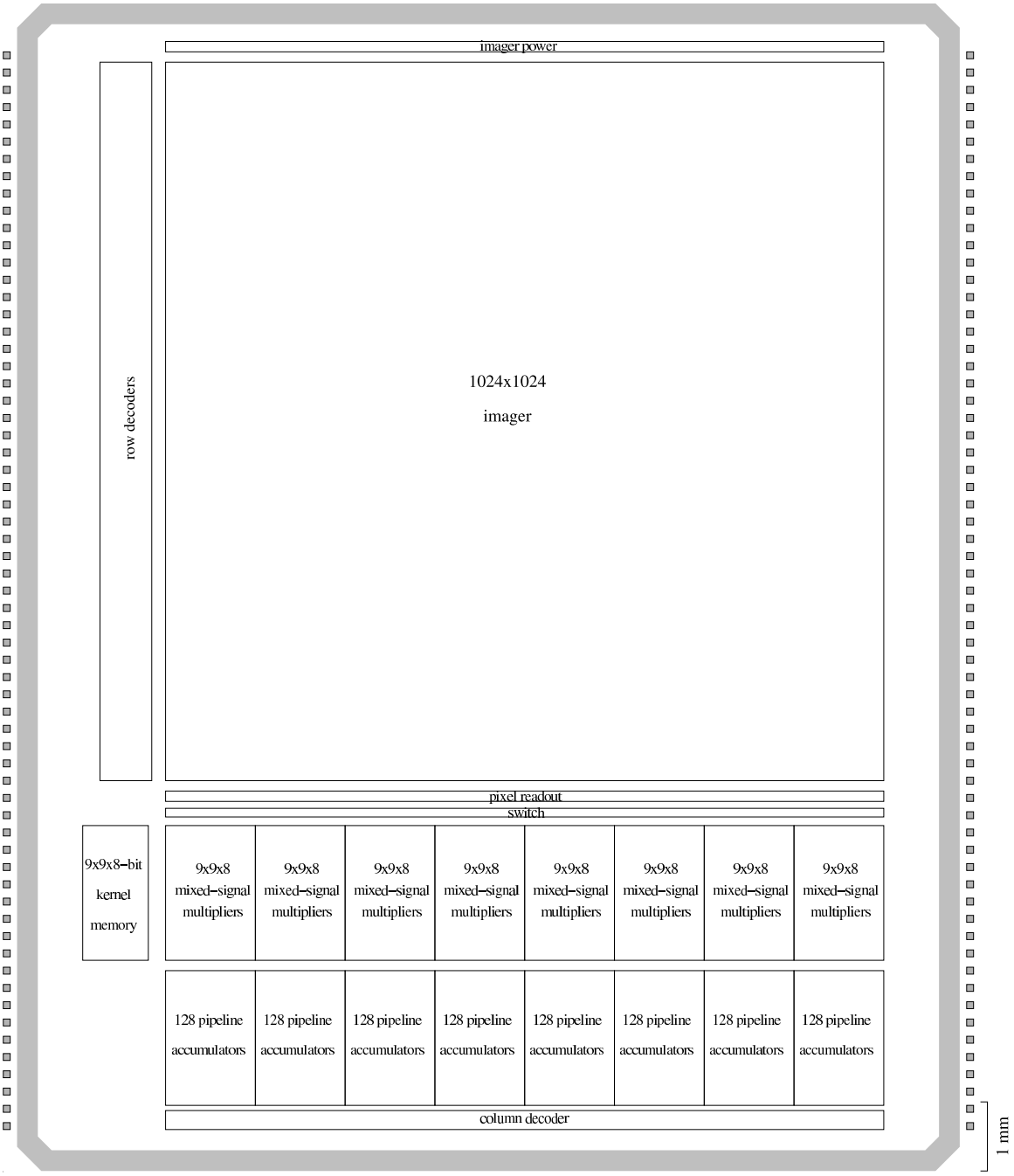


Figure 7.2: $1k \times 1k$ convolution chip scaling layout floorplan. The image sensor scales in both directions but the convolution only in the column direction.

7.3 Layout specifics

The Verilog description of the digital implementation of chapter 4 for the convolution chip was synthesized and generated a layout using a library of standard cells. For the analog design, however, such library was not available, so a fully customized layout was manually drawn. Special care had to be taken to address the specific requirements of this type of layout. Of particular concern are the maximization of matching of similar features through proper transistor sizing, close neighboring of linked structures, multiplicity of small transistors for scaling, and use of dummy cells to preserve the symmetry of the features. Other issues of special interest include the compactness of the cells to keep the full chip size small and help with the transistor matching. Arrayability is necessary on the microscale, e.g., the pixels need to stack together to create the imager, and on the macroscale to allow the chip to grow into a large format imager as in figure 7.2.

7.3.1 Imager

The image sensor is by definition an array. The elementary cells that compose it must therefore be not only compact but also perfectly stackable. Each individual pixel is small but is duplicated so many times that even small increase in size turns into a significant increase in total chip area. The pixel size in the convolution chip is quite large, at $10\mu m \times 10\mu m$, primarily due to the technology used for fabrication which allows for a minimum feature size of $0.5\mu m$ and a large fill factor of 57%. With currently available technologies for imager fabrication of $0.18\mu m$ feature size [2,91] and smaller fill factors, pixel sizes of $2.5\mu m \times 2.5\mu m$ are commonly used. This scaling is necessary when large format arrays are produced. Other techniques, such as sharing transistors between neighboring pixels [92] also help keep the size small.

Figure 7.3 shows the detail of a single pixel layout, that is, the dotted area of figure 5.3. The photodiode occupies a large area, and the empty space around it is required to preserve minimal distance between different diffusion types. The same pixel appears in a small array in figure 7.4 where the interaction between neighbors is more obvious. The vertical metal lines are the power supply on metal 1 and the pixel output above it on metal 2. Each of these lines are common to the entire column. The selection of which row connects to them is done through the two horizontal lines in each row, the selection and reset signals, both

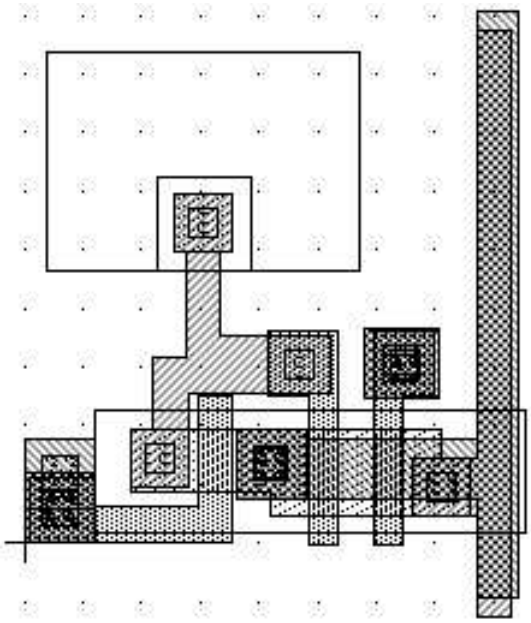


Figure 7.3: Pixel layout (grid spacing = $1\mu m$)

on the highest metal layer, metal 3. so they can cross safely the vertical lines.

Also part of the imager is the readout circuit which allows to transfer the information from the sensor to either a processing unit or the chip output. Its goal in the convolution chip is to transfer the voltage output of the pixels to the current-mode multipliers. A separate readout circuit is implemented for each column according to the schematic of figure 5.4, so a full row can be read simultaneously. It is therefore necessary that it be laid out so that the width corresponds to the width of a column, i.e., of a pixel: $10\mu m$. This explains the elongated form of figure 7.5.

The resistor is made of a high-resistivity poly2 line drawn in a snake pattern to remain compact. The scaling current mirror is divided in small transistors, identical for both the input and output stages but in different number. They are all arranged close to each other to maximize their matching. The capacitor is a MOS transistor, as determined in chapter 5, which is much more compact than a poly-poly2 structure of the same capacitance value. The pitch of $10\mu m$ is preserved with space on each side to allow for minimum clearance between transistor diffusions in adjacent cells.

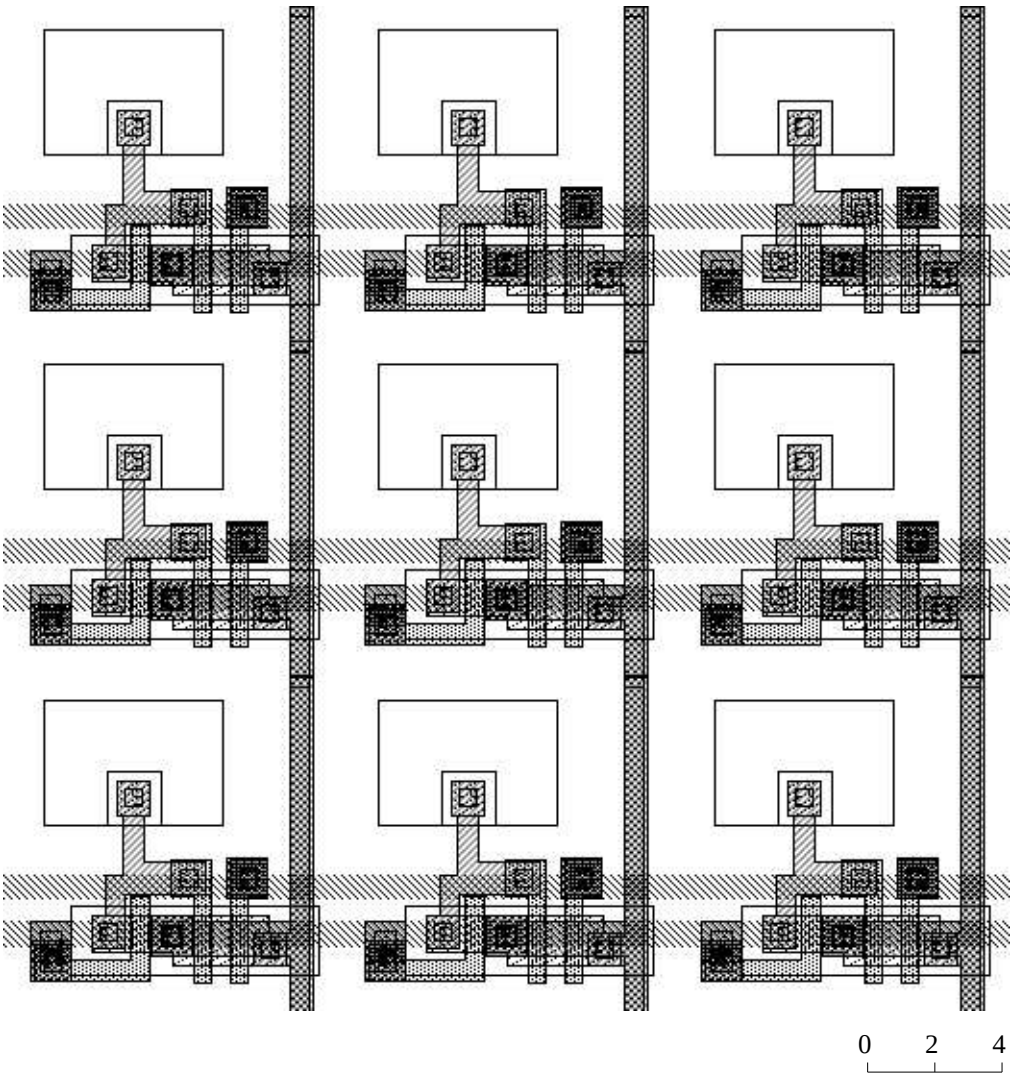


Figure 7.4: Pixel neighborhood (units in μm)

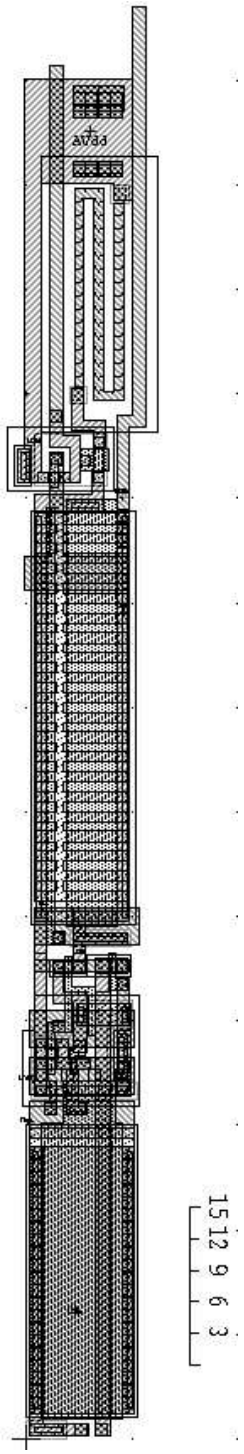


Figure 7.5: Pixel read out (units in μm)

7.3.2 Multiplier

Each incoming pixel column is processed along with its neighbors in a nine-column block. The pixel-wise multiplication combines these nine columns with all nine rows of the kernel, requiring $9 \times 9 \times 8 = 648$ single-bit multiplying units. Continuing with a fully parallel layout would excessively stretch the chip into an unreasonable strip. As a compromise, a multi-level arraying setup was chosen. The multipliers are placed in a 9×9 grid of eight-bit multipliers. As a result, the pixel pitch is no longer respected. The new pitch occupies $160\mu m$ corresponding to the width 16 columns. Each multiplier array is then used to compute the products for all 16 columns under which it is placed.

The schematic presented in section 5.3 accounts for unity-scaled cells duplicated once, twice, four times and eight times to create the multiplication. Each of these mirrors is duplicated so both the lower and upper significant bits are handled. Corresponding bits are combined to form a symmetric structure, yielding the mirrored basic cell containing two current mirrors of figure 7.6. The bottom pair of transistors mirror the input current in a cascode configuration while the top pair are switches controlled by the kernel bits, one from the LSB and its MSB counterpart, i.e., first and fifth, second and sixth, etc. It is this basic cell that will be arrayed 15 times to form the $\times 1$, $\times 2$, $\times 4$, $\times 8$ pairs shown in figure 7.7.

The switch control signals from the kernel are brought on metal two horizontal lines that stretch over all the 8-bit multipliers so they can reach both those working on the same block but on other pixels of the same row and those in different multiplier blocks multiplying other columns in parallel. Basic cells controlled by the same bits are grouped together to minimize wiring and only use one level of metal in the layout. Because of the space needed to draw the wires from the kernel to each line of multipliers, a minimum space has to be respected between groups of basic cells. This explains the gaps shown in figure 7.7. Additional dummy cells with switches tied to ground can be used to fill the gaps (not shown in the figure) and realize a perfectly regular structure which helps with the matching and accuracy of the multiplication.

The elementary multiplier element of figure 7.6 occupies an area of $12\mu m \times 12\mu m$. The full eight-bit single-pixel structure of figure 7.7 with vertical space for dummies and power, control and result lines added stretches to $12\mu m \times 216\mu m$. The 9×9-pixel multiplier needed for each column block is simply a tight array of these single-pixel, eight-bit ones. The size

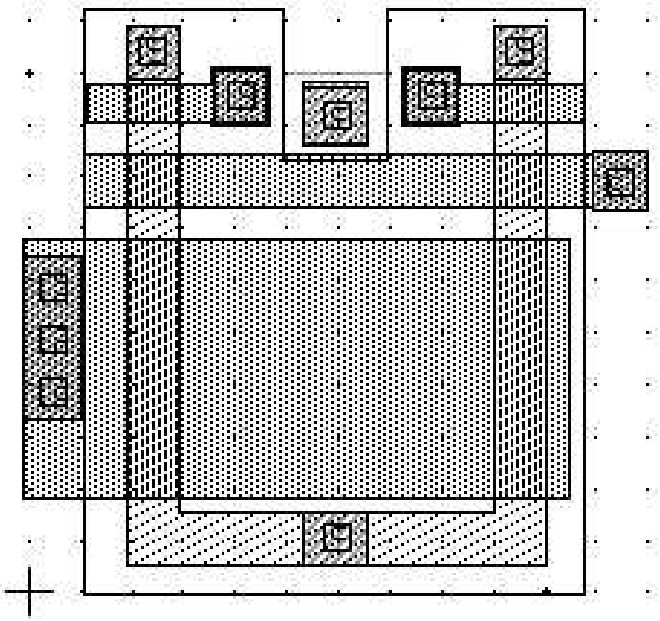


Figure 7.6: Multiplier unit (grid spacing = $1\mu m$)

of the complete block multiplier is then $108\mu m \times 1944\mu m$. Because the fabricated chip received a 128×128-pixel image array, and each block multiplier can occupy the width of sixteen pixels, e.g., $160\mu m$, eight such blocks are placed next to each other. Space is left between each block that will be used for wiring the accumulator, that is, $160 - 108 = 52\mu m$.

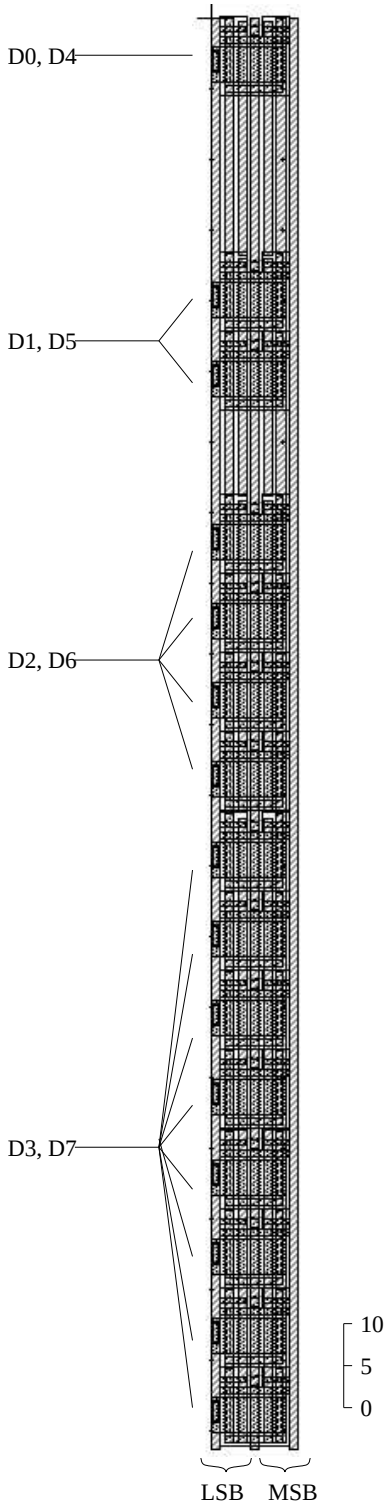


Figure 7.7: 8-bit multiplying DAC (units in μm)

7.3.3 Accumulator

The pitch used in laying out the multipliers is to be used downstream in the accumulators. The difference is that the memory cells that make up the heart of this step cannot be reused when scanning through the 16 columns they are supposed to work on. It is therefore necessary to place 16 complete accumulator pipelines under each other.

All of the nine stages of every pipeline are almost identical. They only differ by the scaling of the two input current mirrors, as required by equation 3.3. The common parts, such as the two current memories and the current mirrors frames, were therefore placed in a template used by all stages. To accommodate the scaling, the repeated transistors are either wired to be part of the circuit or disconnected to become dummies with the sole purpose of creating a regular structure when arrayed into the nine-step pipeline. The complete template shown in figure 7.8 is fully wired with scaling factors of $\frac{1}{9}$ and $\frac{8}{9}$ for the ninth and last stage.

The elongated shape of each pipeline stage, $14\mu m \times 140\mu m$, reminds us of the shape of the single-pixel, eight-bit multiplier. When in a nine-stage array, the pipeline ends up almost perfectly square, nine times wider than the single stage layout. The full nine-stage pipeline measures $126\mu m \times 140\mu m$. Because the accumulators are again to fit under the width of sixteen pixel columns, this leaves room on the side to pull the results on vertical wires so all sixteen pipeline outputs can be sent to column selection switches to be read out. The spare space is $160\mu m - 126\mu m = 34\mu m$, which is enough to fit sixteen metal lines in parallel.

The complete layout of an accumulator pipeline shown in figure 7.9 only displays the computationally useful parts, leaving space for dummy structures. The leftmost stage only has one input so the upper half, corresponding to the input from the previous stage, is left blank. The complexity of the following stages grows linearly until the last which is fully stuffed. The layout closely follows the diagram of figure 3.4.

When all sixteen accumulators are stacked vertically and fully wired to the rest of the chip, they resemble a strip of width $160\mu m$, i.e., sixteen pixel columns, and height $16 \times 140\mu m = 2240\mu m$.

The height of both the multipliers and the accumulators does not grow with the imager; it remains constant regardless of the number of columns because it always handles sixteen

of them. It is duplicated on the horizontal axis to accommodate a larger imager: eight are used for a 128-column imager and 64 for a 1024-column one. The chip floor plans of figure 7.1 and figure 7.2 show the difference. The circuits are fully scalable and their relative area with respect to the imager decreases as the imager size increases. Assuming a square imager, the pixel array grows as n^2 while the computation circuits grow as n. This feature is very advantageous for large imagers when $n^2 \gg n$.

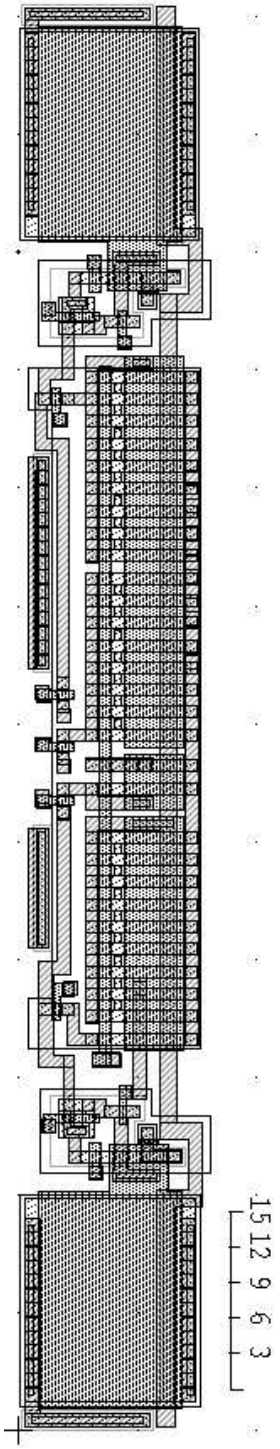


Figure 7.8: Accumulator - single stage (units in μm)

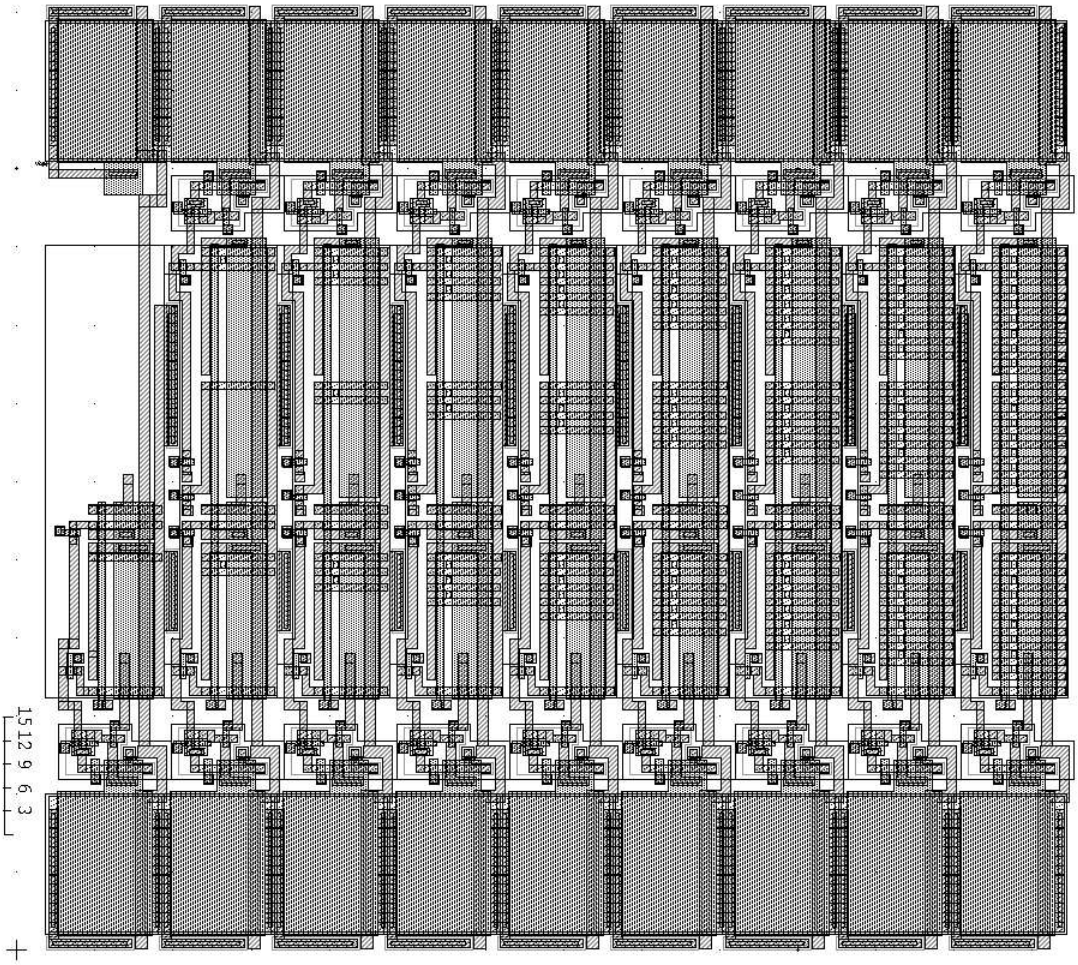


Figure 7.9: Accumulator - complete (units in μm)

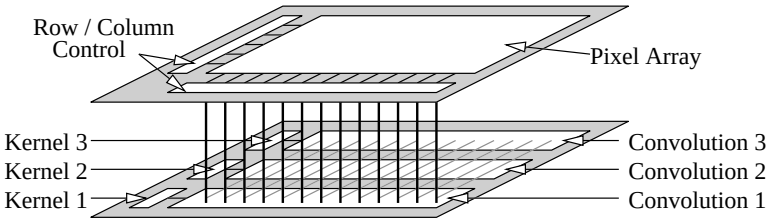


Figure 7.10: Multilayer chip architecture

7.4 Growth prospects

One of the main drive when laying out the convolution was to ensure easy scalability as the size of the pixel array grows. The vertical structure arrayed to interface with the 128×128-pixel imager fabricated can directly be scaled up to any size, yielding such result as illustrated in section 7.2 and the floorplan for a $1k \times 1k$ imager in figure 7.2.

When investigating the resources needed to implement an optical flow computation in chapter 2, references were made to three-dimensional stacking of dies using vertical interconnections. In the case of a purely convoluting chip, such an arrangement opens the door to many exciting possibilities. The proposed schematic of figure 7.10 is an example of such a possibility. The imager of arbitrary size, assumed to be large so it covers a significant field of view with good resolution, occupies the top layer of silicon. The imager controls are on the periphery as in a standard APS architecture. The column readout circuitry, however, connects vertically to a lower level where multiple convolution computations take place in parallel. Because of the scalability of the convolution circuit, each convolution is as wide as the imager above. However, their height is independent of the imager size and therefore several can be stacked up to the imager height. In the technology of the chip that was fabricated, the convolution unit is $3mm$ high so three of them can be placed under a $1k \times 1k$ imager. Each of the convolution circuits being completely independent of each other, they process different kernels in parallel. This is particularly useful for such applications as tracking multiple targets or features identification.

7.5 Micrographs of the chip

Micrographs of the chips are photographs taken through a microscope to illustrate the fabricated circuit. The micrographs of the convolution chip are shown in figure 7.11. The

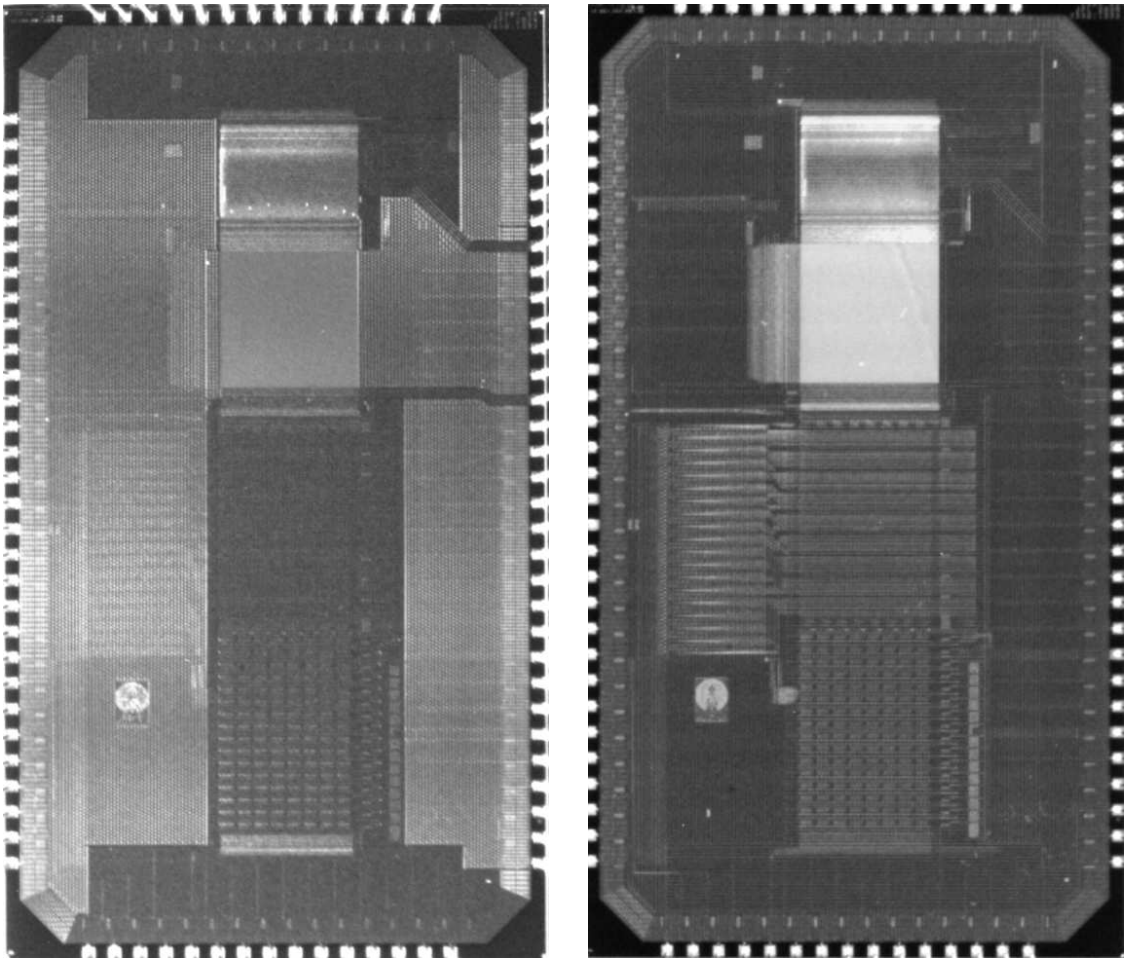


Figure 7.11: Chip micrograph under different illuminations

dominant features can be easily made out, especially the imager which appears as a large square in the upper half. The rest of the chip is covered with a sheet of the highest-level metal to shield the underlying circuits from the high illumination levels. Elements below the shield are harder to see but can still be identified under the right light conditions. The features under the metal-3 layer create relief and distort the light shield. Shadows can be seen when shining light at an angle, thus revealing the features. Micrographs were taken under different illuminations to facilitate identification of the features which only appear through their shadows. The floor plan layout of figure 7.1 helps to recognize the various areas of interest and can be compared to the overlay placed on the micrograph of figure 7.12.

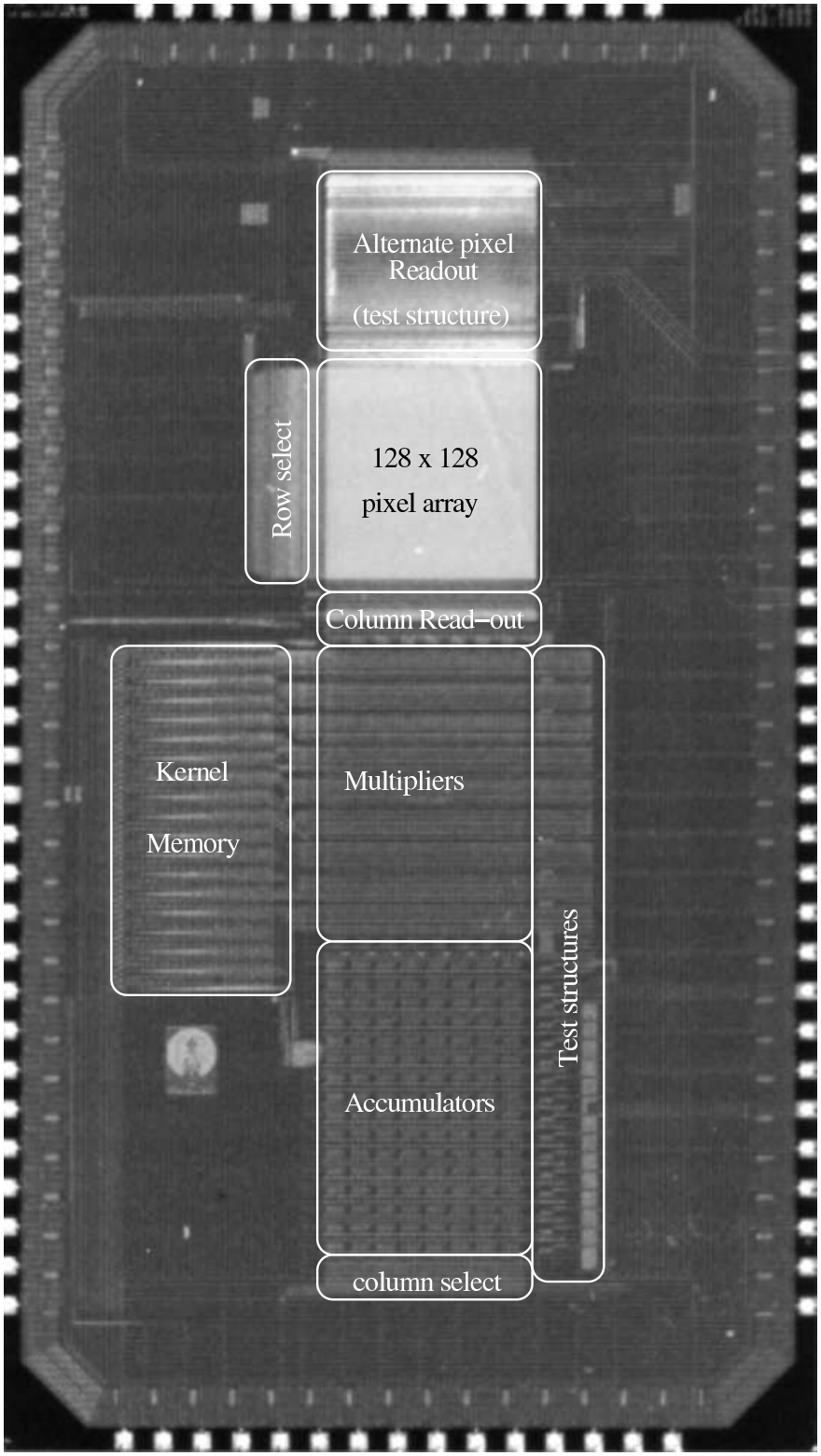


Figure 7.12: Chip micrograph with floor plan overlay

Chapter 8

Characterization and Verification

8.1 Introduction

A challenge when designing a circuit is to ensure it can be thoroughly characterized. For that, each functional block has to be accessible from the interface and has to be made available for independent testing. For that purpose, the convolution chip that was fabricated includes a number of access points for test probes as well as on-site duplicate hardware specifically used for characterization.

The circuits described in chapter 5 were designed to follow the specifications as in chapter 6. The layout presented in chapter 7 was fabricated in an AMI $0.5\mu m$ process and tested on a custom interface board. Waveform generation and digitized data acquisition were performed with a commercial 32-bit digital acquisition board and a 12-bit analog board respectively. Transimpedence amplifiers were used on the interface board to probe output currents (from the imager, the multiplier, and the accumulators) as voltages.

8.2 Imager

The purpose of imager characterization is to be able to assess the quality of both the pixel and the image readout signal chain. For that, a number of tests are performed that look at each step of transforming the incoming light into a usable electrical signal.

The tests performed to characterize the image sensor include looking at the linearity of the response to light, the quantum efficiency of the photodiodes, the conversion gain of the readout circuit [93] and the spectral response. The assessment of the imager noise is done by quantizing the fixed pattern noise, the temporal noise, the dark current and the

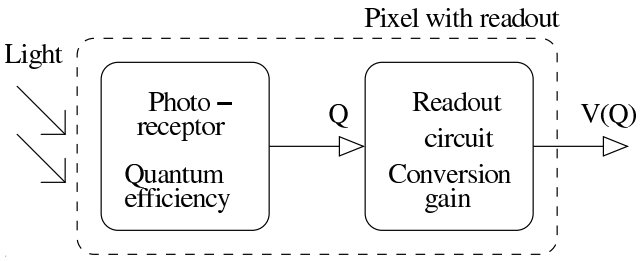


Figure 8.1: The two stages of the imaging cell: a photoreceptor of quantum efficiency QE and a readout circuit of conversion gain CG.

pixel gain variations and non-uniformities. [59, 94–97] The setup, data collection process and analysis of each of these tests are described in detail in the following sections.

When characterizing the imager, a subwindow is usually used to clean the acquired data from pixels suffering from dirt deposited on the pixel array and corrupting the images. The current provided by the pixels is transformed into a voltage through a transimpedence amplifier situated on the test board. It can then be acquired by a commercial acquisition system for data analysis. The amplifier bias is modified to best suit the test performed, which explains the small discrepancies of the absolute values read in the graphs as in figures 8.5 and 8.6. This is only an offset and does not affect the measurements nor the usable range of signals.

8.2.1 Linearity, quantum efficiency, and conversion gain

The signal resulting from illuminating the imager varies according to the transfer function of the system. Ideally, a linear relationship exists between the number of photons reaching the pixel matrix and the output level of the chip (current in the case of the convolution chip). The linearity test consists in analyzing the range of illumination over which the output is indeed a linear function of the light input. This dynamic range is bounded by the noise level when too dark a scene is observed and by the saturation of the pixels when too much light is provided. The illumination is the amount of light on the imager in an image frame, that is, the number of photons reaching the pixel array during the integration time of one frame. It can be controlled either by increasing or decreasing the intensity of the light source or by changing the integration time. A light measuring device is necessary to quantify it.

The test setup consists in shining a controlled light source uniformly over the entire array

and measuring the output of each pixel over multiple frames. A tunable monochromator filters a white light source allowing only a narrow frequency band of light to go through. The filtered light is then diffused into an integrating sphere that uniformly distributes the light onto both the imager and a calibrated photodiode placed diametrically across. A dark frame difference is done to reduce the fixed pattern noise as the imager integration time is scanned with a fixed light wavelength. The pixels are averaged in space and time to take into account the spatial and temporal random variations. Another test will scan the incoming light wavelength to characterize the imager spectral response as in section 8.2.6.

The imaging cell transforms the incoming light into an electric signal in two steps which are each characterized by a transfer function. The photoreceptor generates electrons proportionally to the number of photons reaching the imager according to its quantum efficiency QE. It is a function of the pixel layout (photodiode fill factor) and characteristics of the semi-conductor.

The readout circuit turns those charges Q into a voltage or a current with a conversion gain that depends on the circuit amplification. The transfer function of this stage is the conversion gain CG of the imager. Although the readout of the convolution chip is in the form of a current, it is probed as a voltage in the imager test setup so it is referred here as $V = V(Q)$, as in figure 8.1.

When testing the chip, a readout circuit is attached to the pixel. The quantum efficiency and conversion gain are therefore not readily available as separate entities and some transformations must be done to obtain their values. The only measurable quantities are the output voltage $V(Q)$ and the light reaching the imager. A light-integrating sphere is used to distribute evenly a light source to both the imager and a calibrated photodiode. The calibrated photodiode provides a current that translates precisely into the number of photons hitting it per second and then into the total number of photons integrated on each pixel per frame when combined with the known integration time of the imager.

Linearity

The raw data collected from the imager output when increasing the number of photons reaching the pixel array shows the linearity of the sensor with respect to light. The integration time is controlled to vary the image intensity while the photodiode accurately measures the collected photons. The imager response is linear until saturation is reached for a large

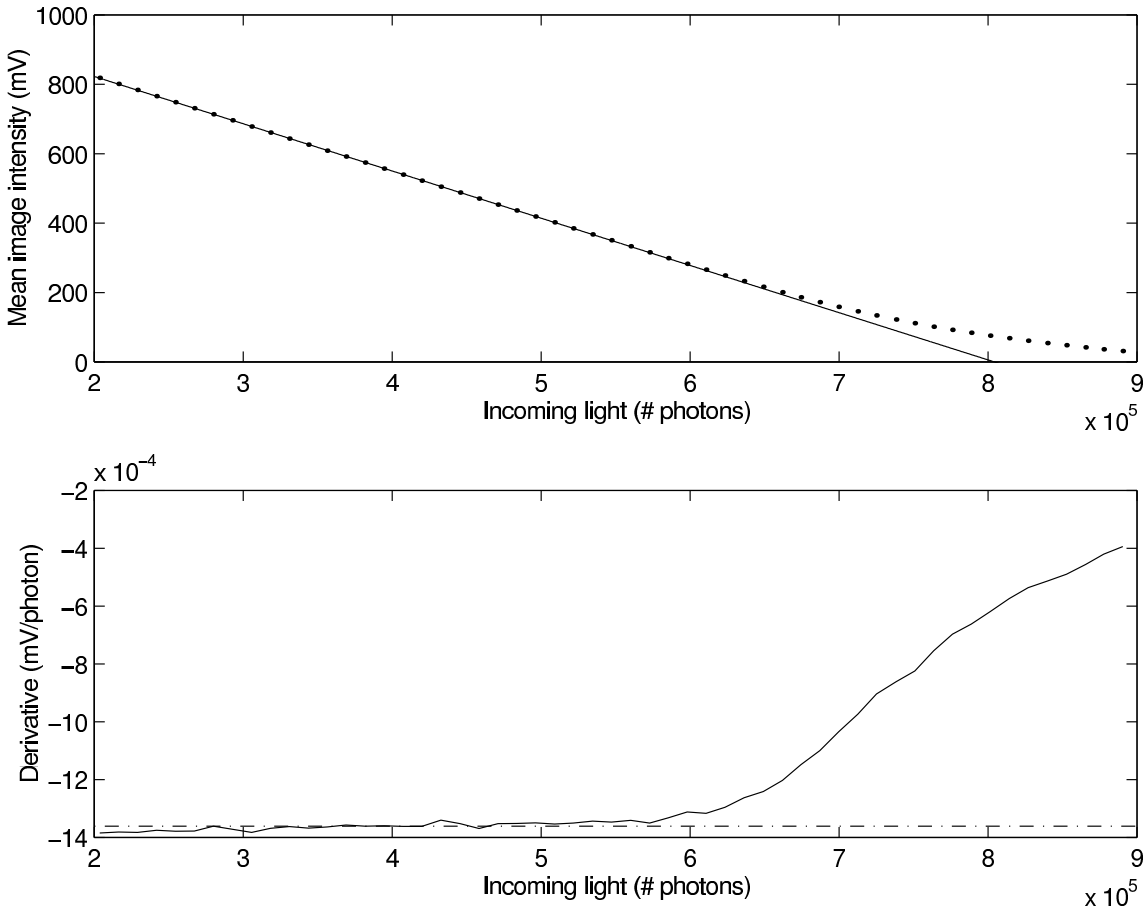


Figure 8.2: Imager response to increasing light exposure with a linear fit (top) and its derivative showing the linear region of operation. (bottom) The dashed line represents the slope of $-13.6\mu V/photon$ in the linear region.

number of photons. The region of operation appears in the plot of figure 8.2 with a fit over the linear region, as well as the derivative showing the slope in that range.

The total conversion is given by the slope and is $-13.6\mu V/photon$.

Quantum efficiency and conversion gain

The data used to show the imager linearity can be analyzed to yield the quantum efficiency of the pixels and the conversion gain of the readout circuit. The collected data include the image mean, its variance and the amount of light sent to the imager.

The number of charges Q given by the photoreceptor is a random variable obeying to a Poisson distribution. Its variance σ_Q^2 equals its expectation: $\sigma_Q^2 = Q$.

The output voltage is a function of the charges, so:

$$\sigma_V^2 = \left(\frac{dV}{dQ}\right)^2 \sigma_Q^2 \Rightarrow \sigma_V^2 = \left(\frac{dV}{dQ}\right)^2 Q + \sigma_{Q0}^2.$$

Note that this equation displays the conversion gain: $CG = \frac{dV}{dQ}$.

Let the number of charges

$$Q = K \cdot L, \tag{8.1}$$

where:

L is a measure of the light (number of photons, integration time under constant illumination, etc.) reaching the imager: $\frac{dV}{dQ} \propto \frac{dV}{dt} \Rightarrow Q \propto t$;

K is the quantum efficiency whose units depend on the definition of L. When L is expressed in number of photons, K is in e^-/photons.

Because of non-ideal elements, we can expect residual charges Q_0 on the photodiode when starting the light integration. (Due to dark current, as measured in section 8.2.5) Therefore,

$$\sigma_V^2 = \left(\frac{dV}{dQ}\right)^2 \cdot (Q + Q_0) + \sigma_0^2. \tag{8.2}$$

Combining (8.1) and (8.2),

$$\sigma_V^2 = \sigma_0^2 + \frac{1}{K^2}\left(\frac{dV}{dL}\right)^2 \cdot (Q_0 + K \cdot L) \tag{8.3}$$

$$\Rightarrow \underbrace{\sigma_V^2}_{y} = \underbrace{\sigma_0^2}_{a_0} + \underbrace{\frac{Q_0}{K^2}}_{a_1}\underbrace{\left(\frac{dV}{dL}\right)^2}_{x_1} + \underbrace{\frac{1}{K}}_{a_2}\underbrace{L\left(\frac{dV}{dL}\right)^2}_{x_2}. \tag{8.4}$$

Equation 8.4 is in the form $y = a_0 + a_1x_1 + a_2x_2$, where y, x_1 and x_2 are measured quantities. The parameters a_0, a_1 and a_2 can be estimated with the least squares fit method. The quantum efficiency and conversion gain can now be separated:

$$K = \frac{1}{a_2}, \tag{8.5}$$

$$CG = \frac{1}{K}\frac{dV}{dL}. \tag{8.6}$$

The underlying assumption in this result is that the quantum efficiency K is a constant over the entire range of the imager, which corresponds to the ideal case. To treat K as a variable, we use a different method to derive it from equation 8.2.

$$(8.2) \Rightarrow \frac{dV}{dQ} = \sqrt{\frac{\sigma^2 - \sigma_0^2}{Q + Q_0}}$$

$$\Rightarrow \frac{dV}{\sqrt{\sigma^2 - \sigma_0^2}} = \frac{dQ}{\sqrt{Q + Q_0}}$$

$$\Rightarrow \int \frac{dV}{\sqrt{\sigma^2 - \sigma_0^2}} = \int \frac{dQ}{\sqrt{Q + Q_0}} = 2\sqrt{Q + Q_0}.$$

Let $r = \sqrt{Q + Q_0}$ and $s = \sqrt{\sigma^2 - \sigma_0^2}$. We now have a simple relationship between these two quantities:

$$r = \frac{1}{2}\int \frac{dV}{s}.$$

The number of charges Q generated is directly found from the definition of r: $Q = r^2 - Q_0$ while the number of photons reaching the imager comes from the calibrated photodiode, noted L in the previous method as in equations 8.3 and 8.4. The integration constants Q_0 and σ_0 are chosen to zero the slope of the conversion gain over the linear region of the imager. The resulting quantum efficiency is plotted in figure 8.3 where a dashed line indicates the value when the imager illumination corresponds to the mean of the image reaching half well (≈600mV).

Once the quantum efficiency of the imager has been determined, it can be used to apply equation 8.6 to extract the conversion gain, as plotted versus the image intensity in figure 8.4. The graph shows a constant conversion gain of $4.7\mu V$ over the linear region of the imager. It drops when the image level approaches saturation and increases (not shown on the graph) for a dark image when the noise of the readout dominates over the signal from the pixel.

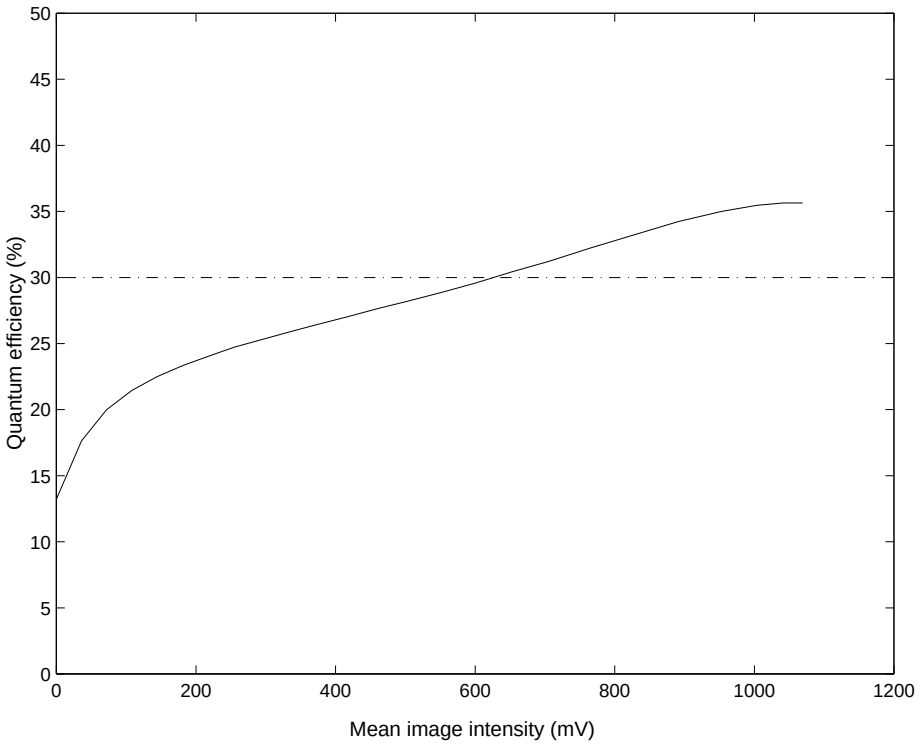


Figure 8.3: Quantum efficiency (%) as a function of the image mean (mV). The dashed line indicates the QE at half well.

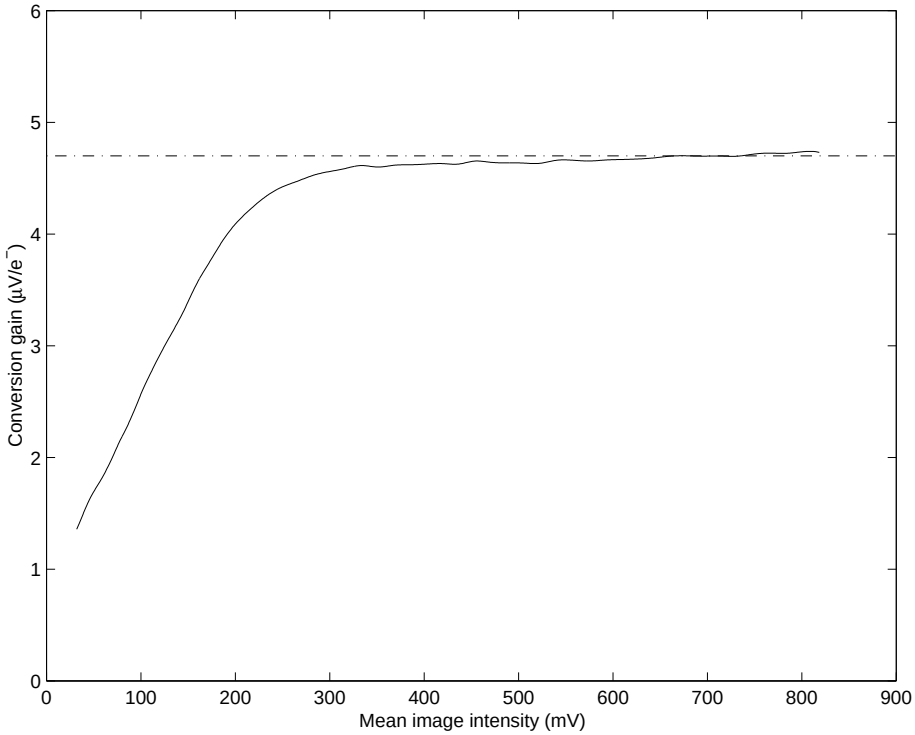


Figure 8.4: Conversion gain ($\mu V/e^-$) as a function of the image mean (mV). The dashed line indicates $4.7\mu V/e^-$ conversion gain over the useful imager range.

Linearity (slope)	$13.8\mu V/photon$
Quantum efficiency	30%
Conversion gain	$4.7\mu V/e^-$

8.2.2 Temporal noise

The variations of each pixel in time (from one frame to the next) form the system temporal noise. This time dependent noise can not be easily removed as the fixed pattern noise and therefore define the precision of the imager. To calculate it, a large number of frames are acquired in the dark over a short integration time to minimize the influence of the dark current. The temporal variance of each pixel form a variance map from which we find the root of the temporal variance average. The result, expressed in $mVrms$ is the noise of the system.

The convolution imager yields a noise of:

$$Temporal\ Noise = 1.32mVrms. \tag{8.7}$$

8.2.3 Dynamic range

The full well of the imager is the usable signal swing between saturated response and darkness. We measured it to be:

$$Full\ Well = 1200mV. \tag{8.8}$$

The sensibility of the imager is given by the dynamic range of the imager which combines the information from the full well (equation 8.8) and the noise of the system (equation 8.7) and is expressed in decibels (dB).

$$Dynamic\ Range = 20 \times log_{10}\left(\frac{Full\ Well}{SystemNoise}\right)$$

$$\Rightarrow Dynamic\ Range = 59.2dB.$$

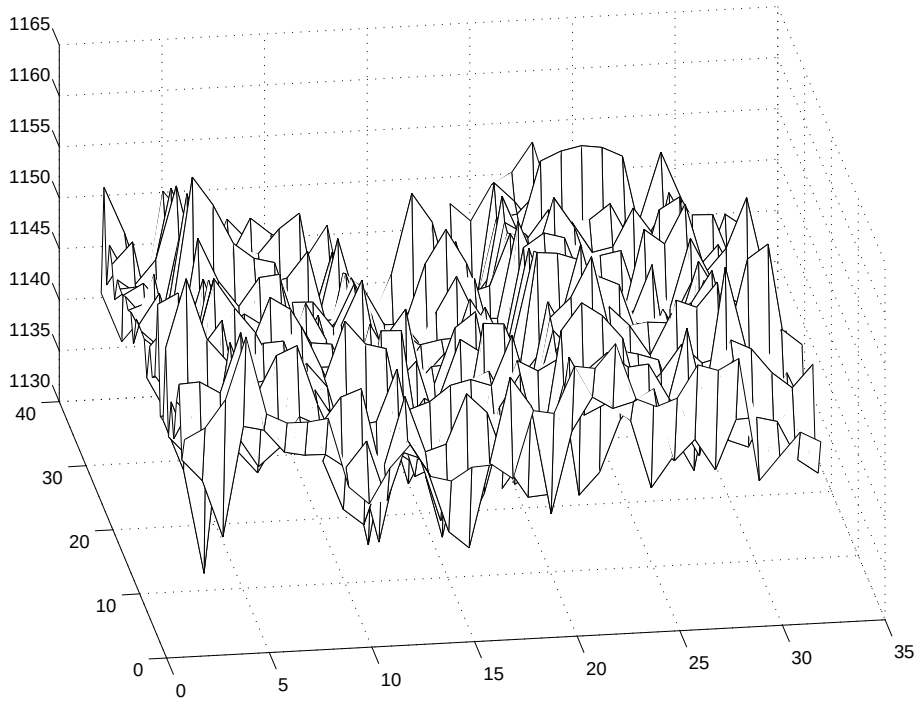


Figure 8.5: Imager fixed pattern noise (mV)

8.2.4 Spatial noise

When the imager is in the dark, the image acquired is not a flat matrix with all pixels equal. A pattern is observed that is typically column based and due to the readout circuitry. A short integration time allows to separate this pattern from the dark current which is characterize in a different test. (See section 8.2.5.) This noise is neither light nor time dependent and creates an offset in the output image. The double sampling scheme implemented and described in section 5.2.3 reduces it by compensating the mismatches occurring before that point in the signal chain. The resulting pattern can be compensated for by external processing if needed.

A large number of frames need to be acquired so the temporal noise can be averaged out and the fixed pattern isolated. The average frame, shown in figure 8.5, represents the offset introduced in the image. It is this average frame that can be used for possible compensation during external processing. We use $6.5 \times \textit{Standard Deviation}$ of the average frame rather than the peak to peak measure as the data span to ignore possible outliers. This number is then normalized by the full well as determined in equation 8.8.

$$Standard\ Deviation(average\ frame) = 4.85mV$$

$$Full\ Well = 1200mV$$

$$fpn = \frac{6.5 \times Standard\ Deviation(average\ frame)}{Full\ Well}$$

$$\Rightarrow fpn = 2.63\%$$

8.2.5 Dark current

The electrons generated thermally when no light reached the imager create a small current flow that can be measured by comparing frames measured over varying integration times. Each frame is an average of many frames (typically 200 frames) acquired under the same conditions so the noise is removed. This being a thermal phenomenon, the dark current increases with the operating temperature. All the experiments were conducted at a room temperature of 298K.

The dark current is found in a two step process. First, the dark rate is found by looking at the increase of the mean signal with increasing integration time. It does not account for circuit amplification but shows how long of an integration time is needed to saturate the imager in the dark. Second, the dark current is determined by combining the dark rate with the conversion gain found in section 8.2.1.

Assuming the dark current is constant, the relationship between the dark image mean and the integration time is linear. This is confirmed by the plot in figure 8.6. The slope of the linear fit is the dark rate, found to be:

$$Dark\ Rate = 0.202V.s^{-1}.$$

The dark current and conversion gain are combined to determine the dark current:

$$Dark\ Current = \frac{Dark\ Rate * q}{CG * Pixel\ area},$$

where $q = 1.6.10^{-19}C$ is the charge of an electron and the pixel area is $121\mu m^2$. The dark current is expressed in $nA.cm^{-2}$:

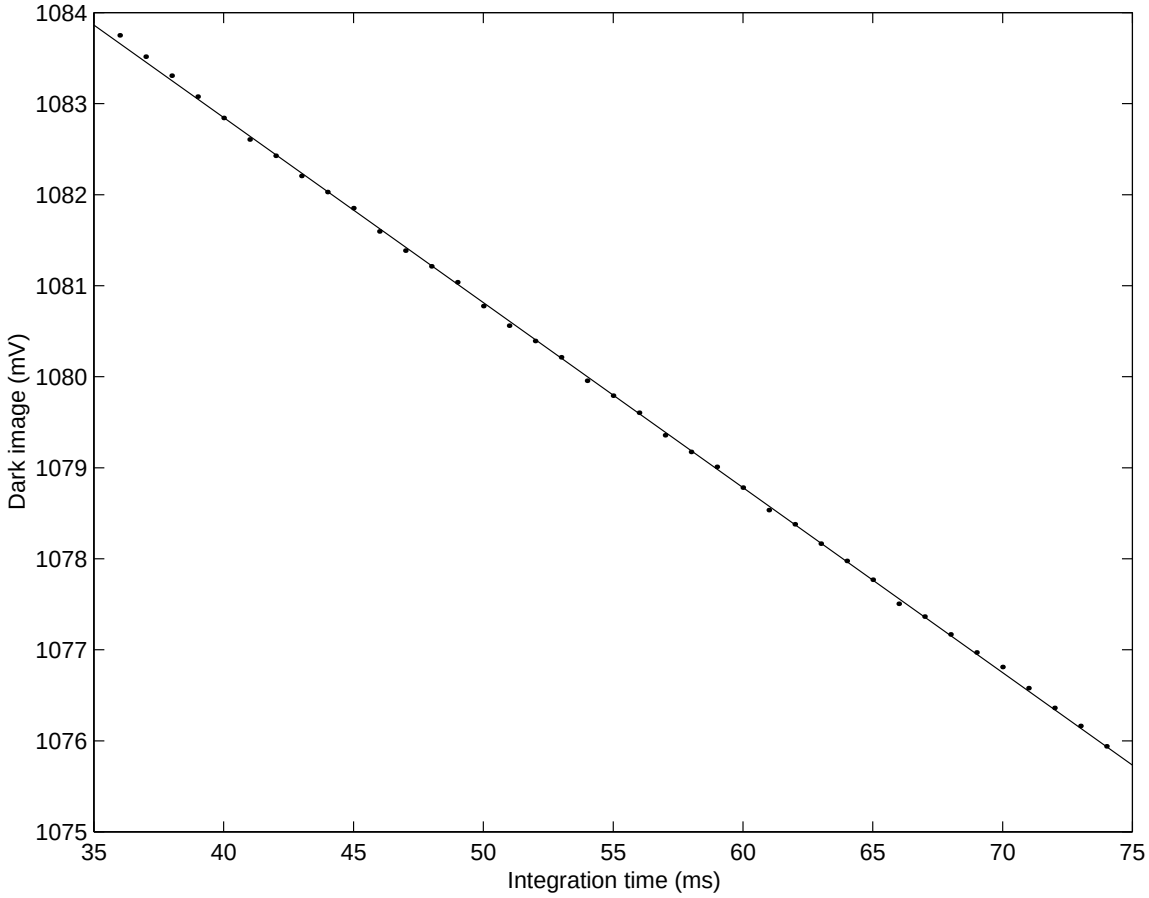


Figure 8.6: Dark rate: measured (marks) and linear fit (line)

$$Dark\ Current = 429nA.cm^{-2}.$$

8.2.6 Spectral response

The sensitivity of silicon to light is a function of the light wavelength. In a setup similar to the one used to determine the quantum efficiency and the conversion gain in section 8.2.1, a monochromatic light illuminates the imager while the response is measured. Unlike the previous experiments where the illumination was made variable by changing the integration time, the spectral response uses a constant integration time but the wavelength is scanned across the spectrum. A calibrated photodiode is again used to measure the number of photons reaching the pixel array. The imager sensitivity, in volts per photon, can then be plotted as a function of the wavelength as shown in figure 8.7.

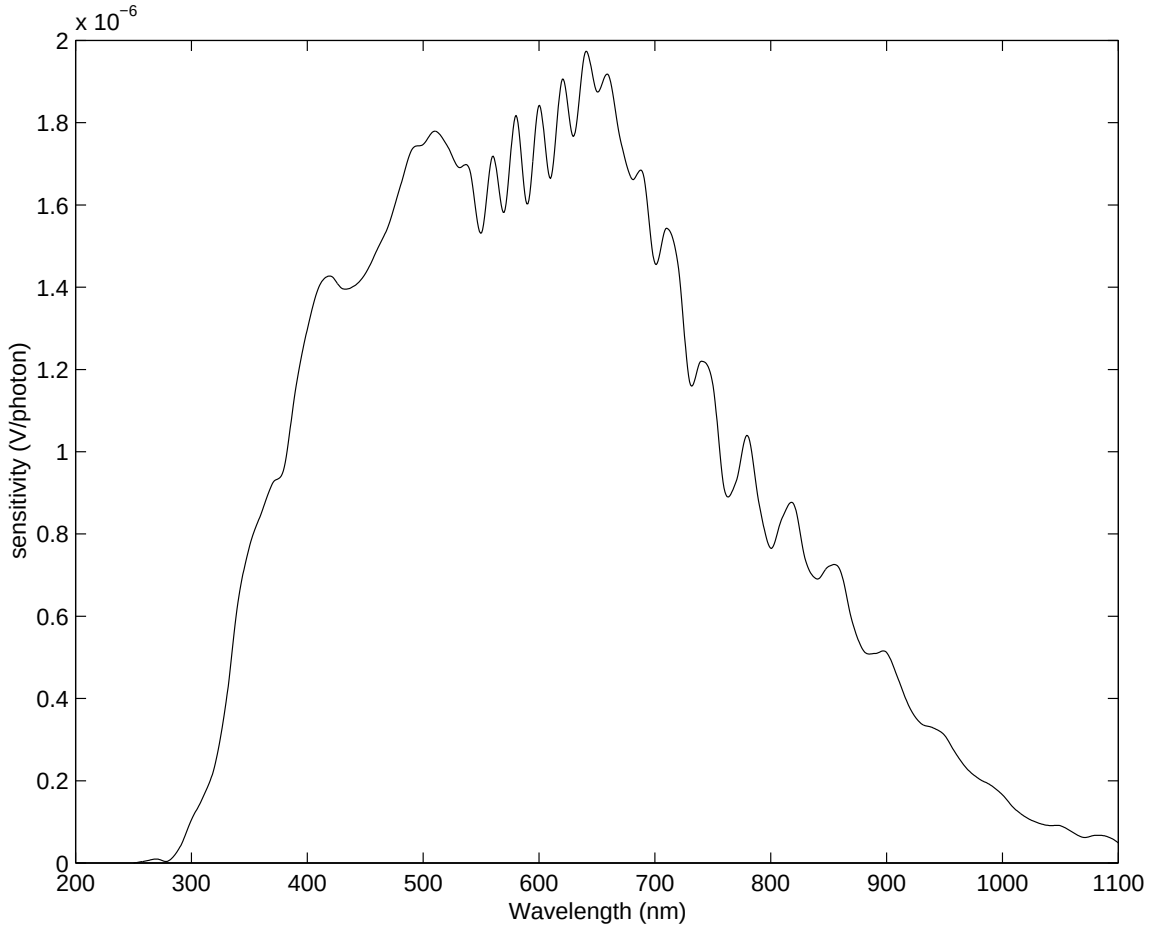


Figure 8.7: Spectral response of the imager. Peak at $\lambda = 640nm$.

(a) 64×64-pixel current mode imager

(b) 128×128-pixel voltage mode imager

Figure 8.8: Images from the direct pixel output of the two types of imagers implemented

8.2.7 Images

Both current-mode and voltage-mode pixel implementations, presented in section 5.2, were fabricated in two versions of the convolution chips. Although the tests in this chapter focus on the latter type which was eventually chosen for the final release, it is interesting to show the imaging properties of both side to side.

Figure 8.8 shows two images of George Washington side to side taken at about the same magnification with two versions of the imager. The image in (a) was taken with the current mode pixel type and displays visible column noise. The noise is mostly due to mismatch in the column current sources that cannot guarantee that exactly the same current flows in each column. Because of the non-linearity of the modulation, compensation is only approximate and very large column mismatches can not be entirely removed. The self-biased readout current mirrors in each column also affect the imager performance. The image in (b) was taken with the voltage mode pixel and displays much lower noise level. The column artifacts after fixed pattern compensation are barely noticeable. This is the pixel used in the convolution chip presented here.

8.3 Computation performance

The two arithmetic units that are used to compute the convolution, the multiplier and the accumulator, can be tested independently of the imager and of each other. Their individual characteristics can thus be asserted, which is a necessary step toward the convolution chip characterization as the reliability of the convolution result depends on the accuracy of both entities. The test measurements for each are detailed in the following sections.

8.3.1 Multiplier

The mixed-signal multiplier can be characterized through a multiplying cell added to the multiplier array and implemented for that purpose only. The digital input is shared with all the other cells as it connects to one of the 8-bit words of the digital memory used to store the kernel information. The analog input is externally controlled to provide a set, measured current, while the output is probed on a transimpedence amplifier placed on the interface board. Testing the multiplier involves verifying its accuracy and its linearity with respect to each of its two inputs: the analog current and digital kernel.

A sweep of the current for various values of the kernel is performed, similar to the simulation shown in section 6.5. A wide scan of input current for several kernel values shows, in figure 8.9, a linear region followed by a saturation region, as was expected from figure 6.13. The region of interest shown in figure 8.10 closely matching the simulated output of figure 6.14.

The reliability of the calculations depends on how well the circuit under test matches the ideal expected results. To quantify this, the multiplier outputs for set kernel values were fitted with a straight line for input currents in the useful range of $[0, 10\mu A]$. The fit parameters would ideally be zero for the constant term and the slope equal to the kernel value, with a fixed scaling constant, independent of the kernel. Only small variations were found in the origin of the lines, as shown in figure 8.11 (top).

The slope of the ideal multiplier is $slope = T/cst$ where cst is the product of the attenuation at the input and at the output of the multiplier cell. The differences from the ideal case are normalized to show their influence on the final result, giving more weight to larger kernel values that cause greater output variations. The plot in figure 8.11 (bottom) shows the observed variations for several kernel values.

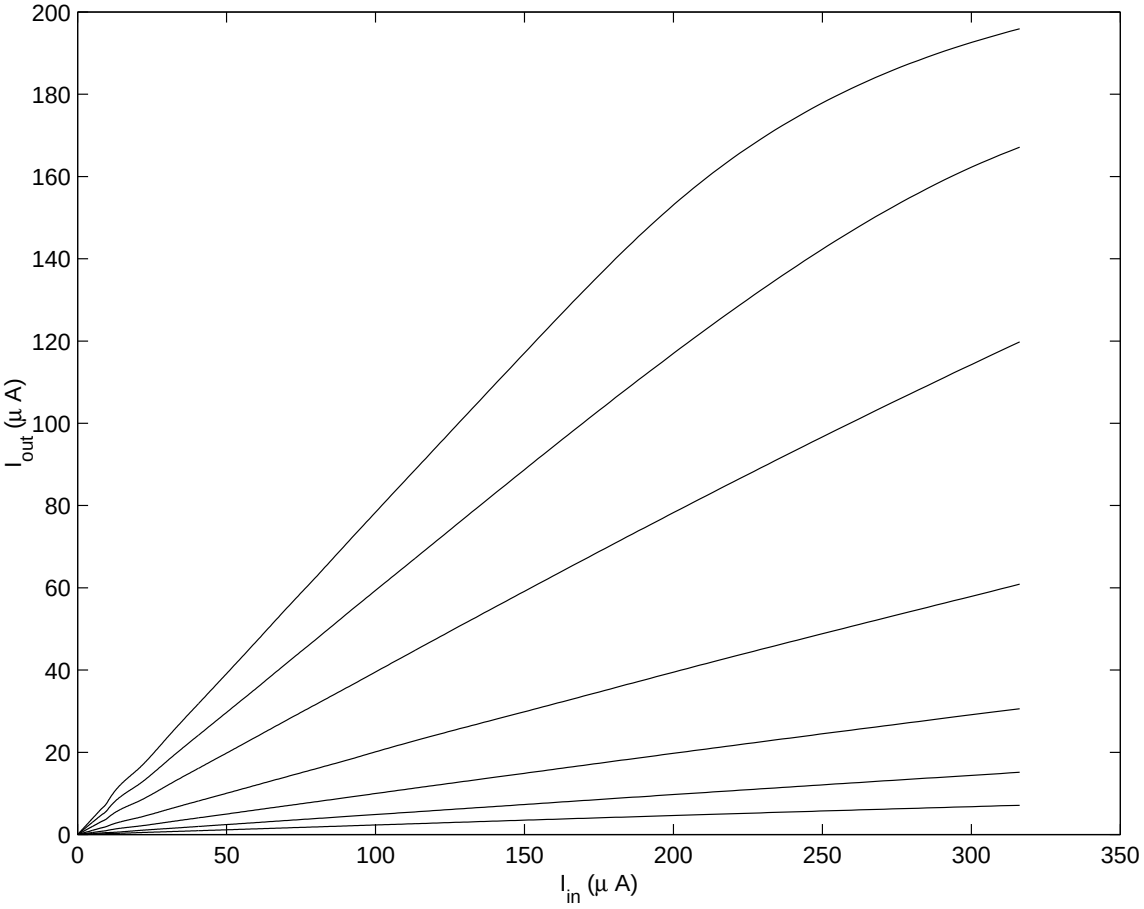


Figure 8.9: Multiplier output with kernel values of: $K = 8, 16, 32, 64, 128, 192, 255$.

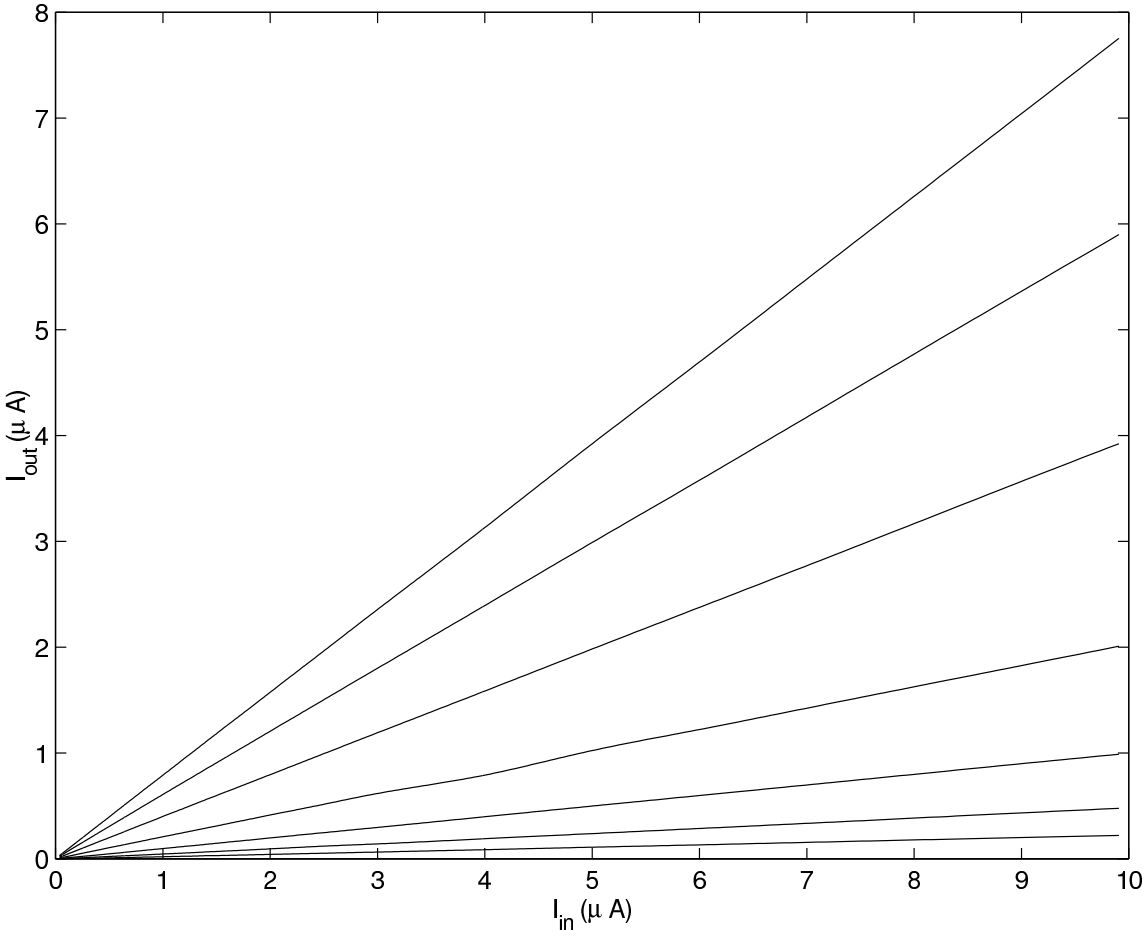


Figure 8.10: Multiplier output restricted to the range of signal used.

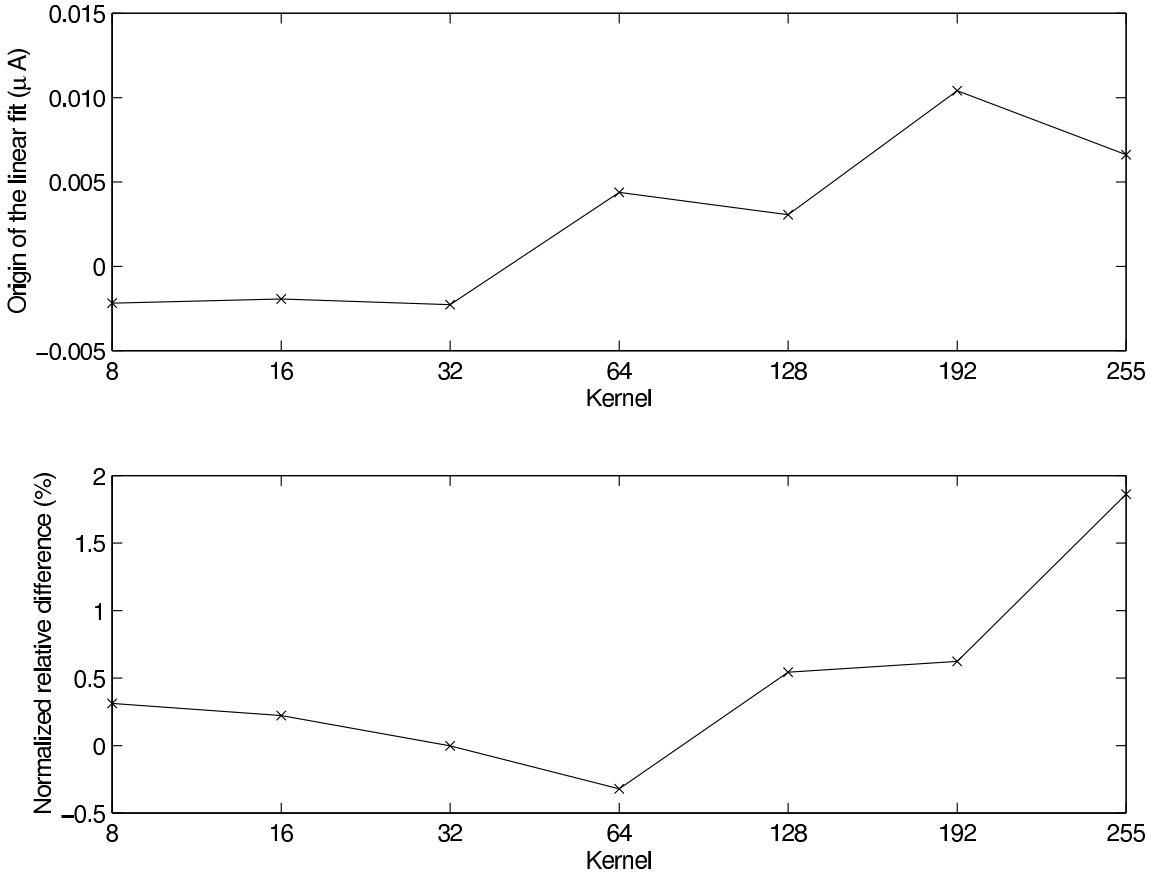


Figure 8.11: Parameters of the linear fits for kernel values of $K = 8, 16, 32, 64, 128, 192, 255$. Constant term on top (ideally zero) and weight of the slope variations on the multiplication output (bottom)

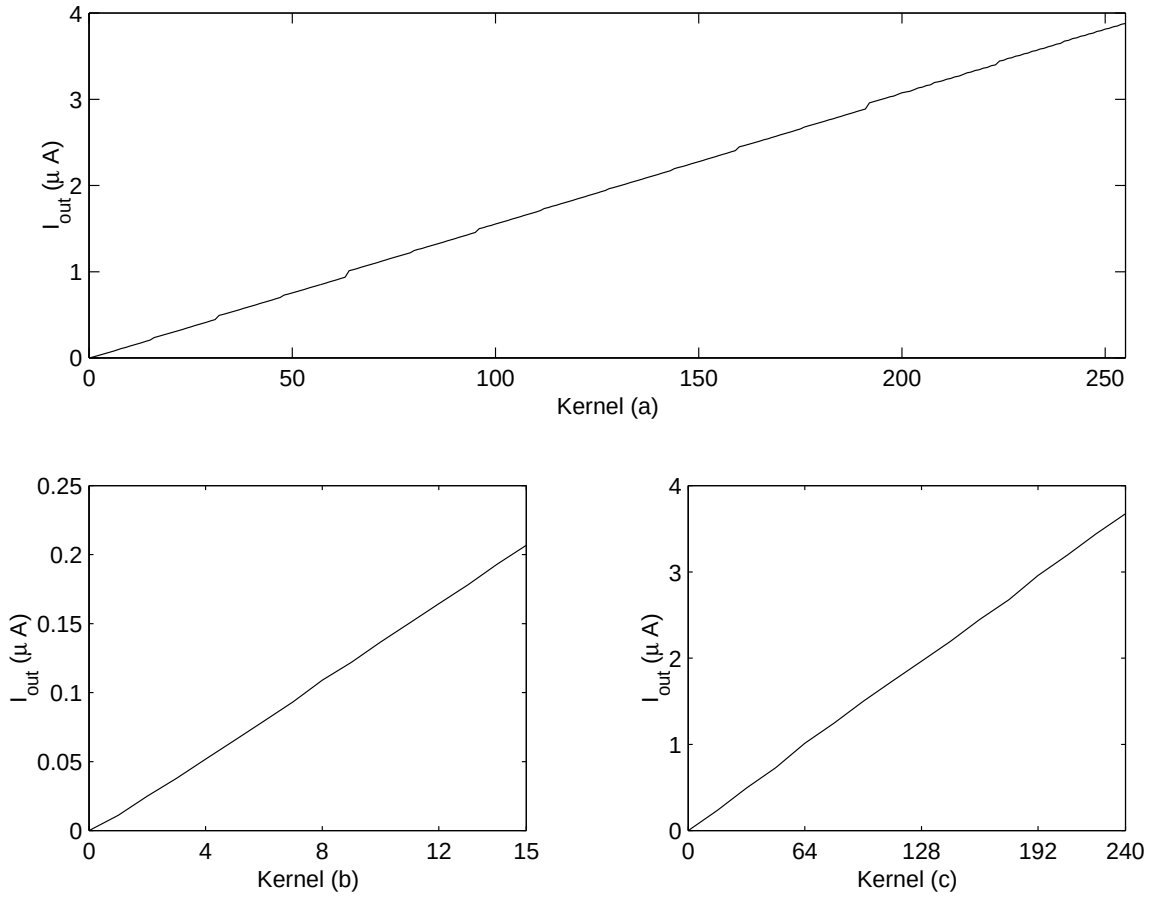


Figure 8.12: Sweep of the kernel for a fixed input current $I_{in} = 5\mu A$. (a) Kernel swept from 0 to 255. (b) Kernel swept from 0 to 15: least-significant bits only. (c) Kernel swept from 0 to 240 by steps of 16: most-significant bits only.

In a second test, we can look at the multiplier linearity with respect to its digital input. The analog current input is set to a fixed value (midrange: $I_{in} = 5\mu A$) and the kernel value is varied from 0 to 255. The inaccuracies observed in the linear fits described above and in figure 8.11 appear as discontinuities in the graph of figure 8.12(a). Consistent with the findings of figure 8.11 (top), the 7^{th} bit of the kernel creates a noticeable jump in the data, that is when the kernel is 64 and 192. The inaccuracies are however small enough that they do not compromise the monotonicity of the function. The multiplier is therefore reliable enough to perform the convolution function it is intended for.

8.3.2 Accumulator

The accumulator is the second arithmetic cell that complements the multiplier for computing the convolution. As described in section 5.4, each accumulator is made of nine cells arranged in a pipeline. The first cell is a simple current memory, that is a single-cell structure while every other stage integrates two current memories in double-cell structures. Test results for both types are presented here, as well as for the entire pipeline, in a similar fashion as the simulations of section 6.6.

8.3.2.1 Single-cell module

The first stage of the accumulator pipeline is primarily a current memory that initializes the nine step sequence. The input current is mirrored into a memory cell which is read out later through another current mirror. The interface of one of the cells is directly accessible from the chip pads, allowing the direct testing, in a setup similar to the simulation of section 6.6.1. Unlike the representation of figure 6.15 showing continuous waves responding to the chronogram, the actual test samples the output on the readout phase, giving the relationship between the input and output currents.

The description of the cell in section 5.4.2 shows that the output follows the input as no scaling was introduced in this stage. The relationship $I_{out} = I_{in}$ is expected by design, also confirmed by the test results, similar to the simulation. A linear fit and difference plot show the characteristics and linearity of the cell.

8.3.2.2 Double-cell module

With the exception of the very first stage tested above, each cell of the accumulator follows the same structure. They comprise two current memories and a scaling coefficient on each of the two inputs which depends on the position of the cell in the pipeline. For testing purposes, one of these stages was made accessible with full control and monitoring of the inputs and output. The cell follows the architecture of section 5.4.3.

The test performed to characterize the double-cell module is similar to that of the single-cell one. The input normally received from the previous stage is held at a typical value of $1\mu A$, so only the scaling was taken into account when comparing the output to the expected value. The results shown in figure 8.15 and 8.16 are that of the fourth stage of the pipeline,

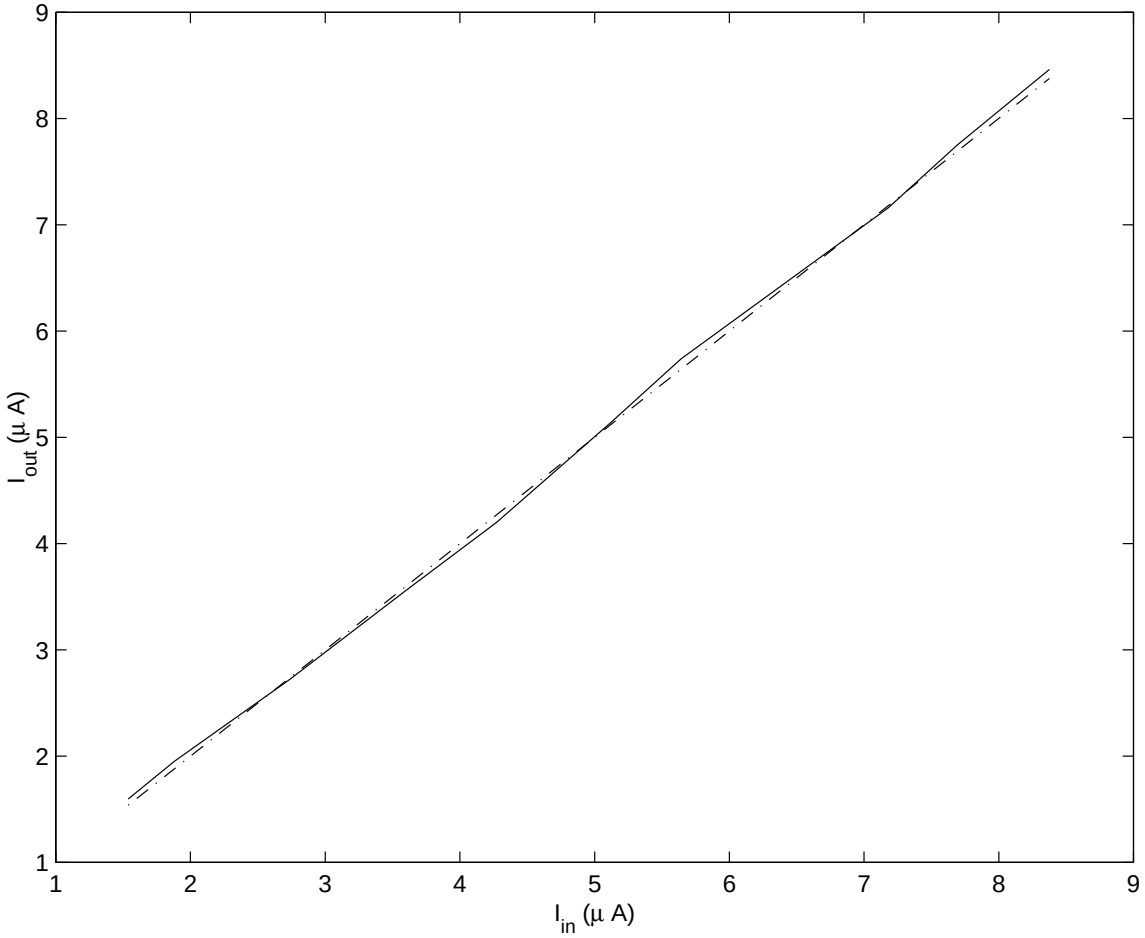


Figure 8.13: Single-cell module. First stage of the accumulator pipeline: solid line. Ideal response: dashed line.

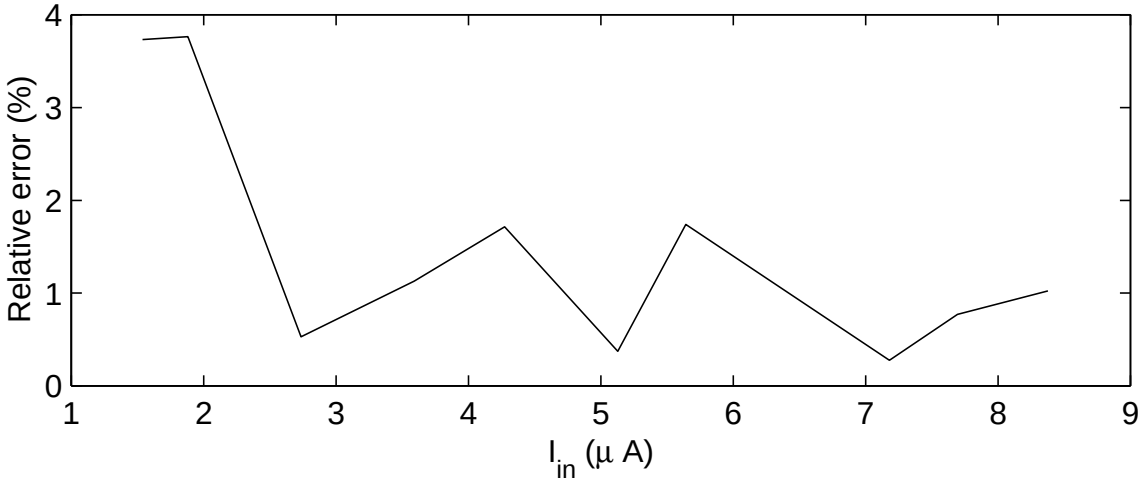


Figure 8.14: Single-cell module. First stage of the accumulator pipeline: relative error.

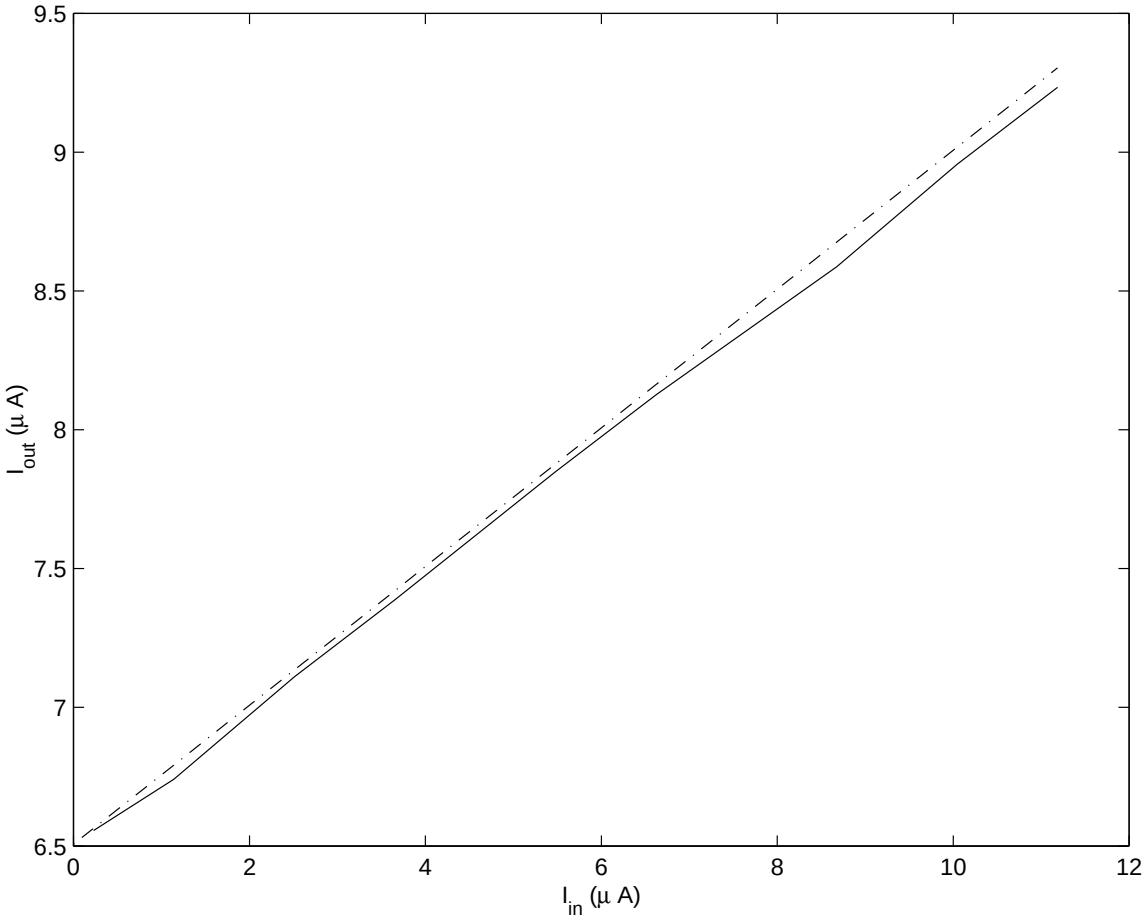


Figure 8.15: Double-cell module. Fourth stage of the accumulator pipeline: solid line. Ideal response: dashed line.

placed under test as an independent cell. The expected output from inputs $I_{previous}$ and I_{in} is then:

$$I_{out} = \frac{3}{4} I_{previous} + \frac{1}{4} I_{in}$$

8.3.2.3 Pipeline

The accumulator pipeline combines, in series, nine stages similar to the one tested above. The nine input currents that make up the accumulator interface are created for this test from a single current that passes through the multiplier unit. The digital kernel uploaded to the chip allows to create nine different inputs, and even to change them rapidly, simulating an incoming flow of pixels. To illustrate the global behavior of the accumulator, nine identical

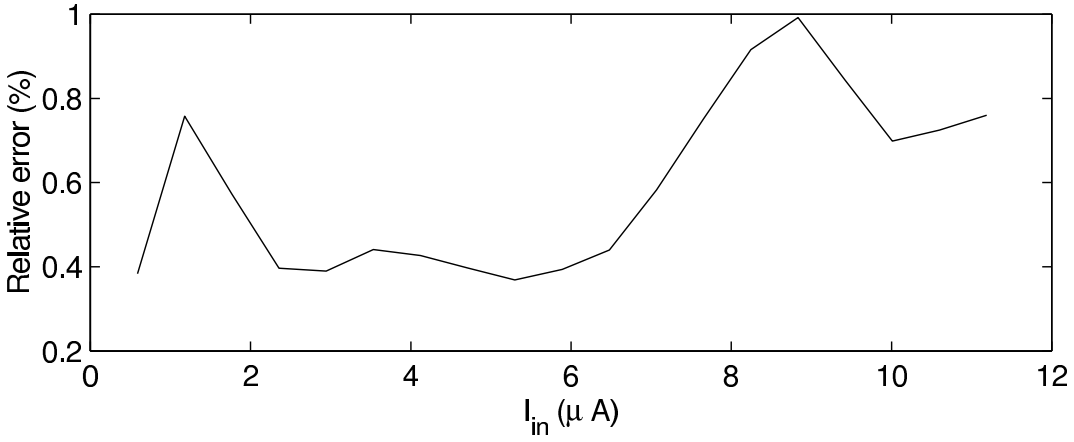


Figure 8.16: Double-cell module. Fourth stage of the accumulator pipeline: relative error.

inputs are sent to the accumulator and nine cycles are run so all stages are used. The kernel used cancels out the on-chip current scaling so the output is expected to follow the input. A sweep of the input current is done as illustrated in figure 8.17.

The description of error propagation in section 6.7.1 shows how random statistical errors compensate but systematic errors in each block accumulate, creating an offset in the accumulator output. Because it is an offset, its relative influence is larger when the output signal is small. Figure 8.18 shows how this effect on the circuit under test.

8.4 Power dissipation

A usual requirement for usability of a chip in autonomous or semi-autonomous systems is a cap on the power dissipation so it can be operated on batteries. The power dissipation of the convolution chip was therefore measured in both modes of operation: imaging only and imaging with the convolution filter running.

Operating mode	Power dissipation
Imaging only	$10mW$
Convolution imager	$90mW$

Table 8.1: Imager and convolution power dissipation

To perform this measurement, a dedicated power supply is provided to the chip independently of the interface board. The power line is monitored, and the average power is

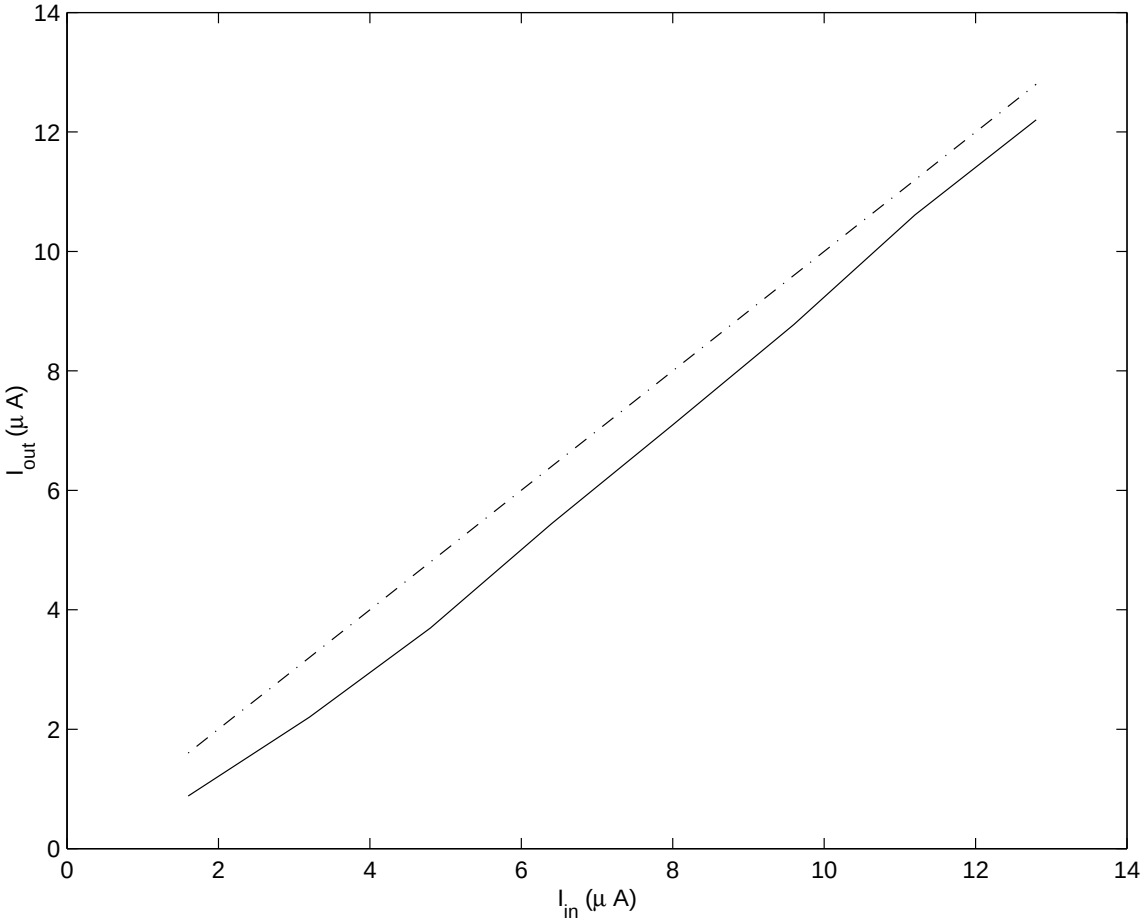


Figure 8.17: Complete accumulator pipeline: solid line. Ideal response: dashed line.

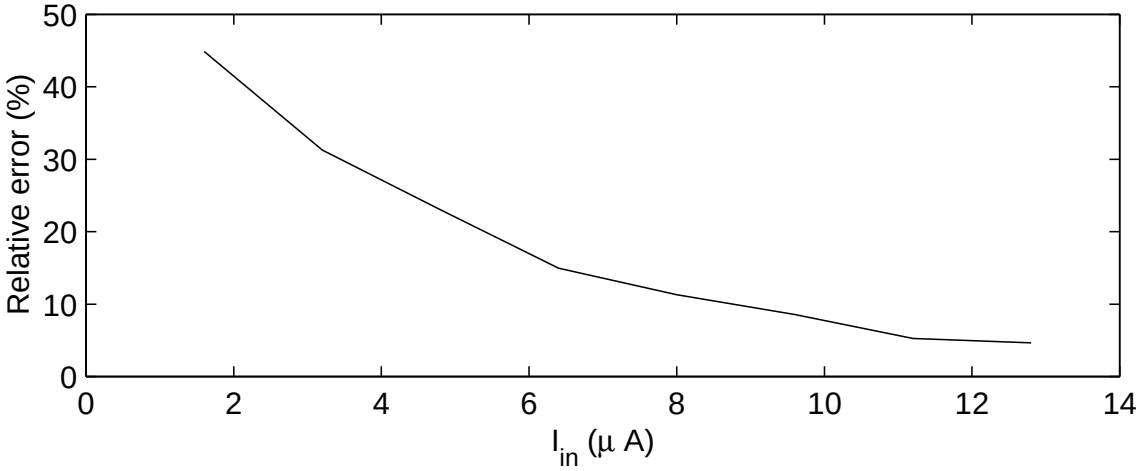


Figure 8.18: Complete accumulator pipeline: relative error.

measured over multiple cycles with the imager outputting video at a rate of 30 frames per seconds. The results for both modes are summarized in table 8.1.

8.5 Conclusion

Each of the main computation elements were fabricated as independent test circuits to allow easy characterization of each of them separately of the convolution chip. Implementing these test structures is a convenient way to evaluate the performance of each individual block and predict the performance of the assembled complete convolution chip without having a fully operational chip. It also allows to identify the possible weaknesses in the circuit and the elements that could be improved upon. Finally, troubleshooting is simplified as internal probing along the signal chain permit tracing error sources inside the chip.

Thanks to the exhaustive testability of the circuit, the convolution chip could be tested thoroughly, with the results comprising this chapter. The imager performances, summarized in table 8.2, show that despite reasonable sensitivity, the image quality suffers from high levels of noise. The experimental circuits created when designing pixel readout blocks are certainly to blame as it suffers from two major sources of error. First, it suffers the uncertainty of the current memory from figure 8.13 used for reducing the effect of fixed pattern noise in the pixel column and voltage to current conversion. Second, the subtraction to reduce the FPN is done before any other processing. The rest of the signal chain is not differential and therefore any mismatch will affect the column to column signal accuracy. A workaround is to perform the entire convolution on the pixel reset and signal levels independently and combine them just before digitization or chip output in the case of an analog readout.

Full well	$1200mV$
Dynamic range	$59.2dB$
Quantum efficiency	30%
Conversion gain	$4.7\mu V/e^-$
Dark current	$429nA.cm^{-2}$
Fixed pattern noise	2.63%

Table 8.2: Imager performance summary

The computational stages were also analyzed and give various degree of confidence in

their accuracy. The multipliers proved to be linear in the range of operation they are designed to operate in. The noticeable mismatch between lower and upper halves being small enough to not compromise the bijectivity of the multiplication.

The accumulator, however, displays a more problematic trend. Each stage only introduces a small error as shown in the graphs of figures 8.14 for the first stage and figure 8.16 which is representative of the other stages. However, the errors are not random and they systematically add in the same direction, creating a drift in the output. This behavior was predicted in the accumulator error propagation analysis of section 6.7.1. The outcome is a significant drift in the simulation of the full pipeline in figure 8.17. The resulting relative error is large but because it results from an offset from the ideal response, it decreases with signal strength. It can also be easily accounted for with a simple calibration scheme.

Chapter 9

Conclusion

9.1 Summary

The resolution of images from solid-state sensors keeps increasing as new technologies and fabrication processes allow for ever more compact integration of circuits on silicon. With the size of the images, the amount of data produced grows as well. The problem of image processing continues to be a sensitive part of any imaging system, as the processing must be able to keep up with the flow of information and the ever increasing sophistication of the algorithms. The work presented in this thesis outlines the complexity of choosing a processing scheme most appropriate for a given imaging task. To illustrate this, a computationally intensive, yet common image processing task was chosen and analyzed: the computation of the optical flow of a stream of video. Real-time operation of this task requires efficient data handling capabilities as well as computing power able to process fast arithmetic operations such as multiplications and accumulation.

Imaging systems based on active pixel sensors (APS) technology were used for the work presented here. Their versatility and the possibility to integrate on-chip processing without affecting imaging performance make them an attractive choice for this task. The appropriateness and usefulness of implementing optical flow computation on the focal plane was shown through analysis of a hardware oriented derivation. At the heart of the optical flow computation is a convolution operator which convolves the image with a preloaded kernel. Because that convolution operator is the computation bottleneck for the optical flow, it is the focus of the work on hardware implementation. To efficiently compute the convolution, an algorithm developed specifically for hardware implementation was presented that takes fully advantage of the column-parallel structure which is characteristic of APS

image sensors. The algorithm was demonstrated on a fully digital Verilog description. A prototype was built which combines an existing image sensor with a FPGA programmed with an interface, a kernel memory and a convolution operator. The digital description also yielded a layout for a full custom circuit that was used as a reference for comparison with the circuits developed after that.

A compact, fully analog implementation of the convolution algorithm was investigated to demonstrate the appropriateness of analog design for low-level processing. Unlike with digital designs, analog implementations critically depend on their accuracy to transmit information. The design of each block must play their role in the signal chain and smoothly interface with other stages. The final result of the convolution depends on it, so the design phase was described in details to demonstrate how critical choices can be dealt with in order to create the circuit. The expected errors sources, both from random geometrical and matching variations and from circuit noise were studied to anticipate how a fabricated chip might vary from its expected and simulated behavior. Because the circuit noise and the transistor mismatches are a direct consequence of the layout, the philosophy and techniques used to layout the convolution chip were also explained, and each relevant circuit block scrutinized to show how proper geometry considerations had to be observed. This concluded the presentation of the chip and the development steps from concept to product.

The results of the tests performed on the fabricated convolution imager chip, and the comparisons with the digital test bench, highlight the usefulness of analog image processing on the focal plane. The limitations of fully analog implementations are also pointed out and should always be taken into consideration when designing such a system. The digital implementation used as a test bench revealed compelling reasons to look into analog image processing solutions. Investigation of both options showed that analog arithmetic units offer area reductions over their digital counterparts of 160 times. Trade-offs sacrificing timing and complexity to area only reduce this ratio down to 50 times. Analog processing, however, lack the simplicity of a reliable description language that easily translates an algorithm into a synthesizable circuit. They also lack the predictability of digital operators, as they are sensitive to noise and mismatch. Drifts from ideal characteristics were observed in the circuit tested which could be reduced by revisiting the design of some of the most critical elements but not entirely eliminated.

An example is the good performance of the multiplier which proved to be satisfactorily

linear and predictable. The pipeline accumulator, however, showed large deviations due to accumulating and amplifying errors due to mismatch. Previous circuits tested a charge mode accumulating circuit that was symmetrical and therefore less sensitive to errors propagating but resulted in an unreasonably large layout. The critical choice to make for an optimal circuit implementation is to decide when to digitize the signal in the signal chain. An analog to digital converted is necessary at some point to use the image off-chip, but conversion can occur at any time in the processing. In the circuit presented here, it appears advantageous to use analog multipliers but accumulators seem to better perform in the digital domain. The analog to digital converters could therefore be placed at the output of the multipliers in order to take advantage of what both worlds have to offer, combining the compactness and power reduction of analog circuits with accurate, easily implemented digital operators.

9.2 Future work

A few openings were introduced in this report that have the potential to take this work another step further. The first step would naturally be to use the data collected and analyzed to feedback into the design process and re-visit the blocks that yielded large errors when characterized. Of particular importance, the fixed pattern noise reduction circuit showed that the mismatch of the subtracting current memory affected the column noise, as did the single-ended convolution signal chain. The accumulators, which proved to be very sensitive to errors, would benefit from a design review based on the test results.

Beyond the current circuit, the demonstrated feasibility of low-level, integrated analog image processing opens the door to implementing other useful algorithms that would also benefit from this approach, such as applications based on convolution. A full optical flow computation is the obvious choice, as would be a tracking system which uses convolution to perform normalized correlation between the image and a template. Tracking introduces little extra circuitry since the normalization only adds an accumulator to measure the weight of the image patch and a divider. The accumulator is similar to the only used in the convolution so only a dividing circuit would have to be designed.

Finally, equally as exciting as developing new algorithms and new highly integrated systems on a chip using analog or mixed-signal processing is the many extensions that ever evolving technology can offer. State of the art sensors use very large pixel arrays of up

to 16 million pixels which put scalable processing units, such as the one described here, a practical solution. Also, as vertical interconnections between dies are becoming a more mature technology, they represent a natural evolution, going from on-focal plane computing to "close to focal plane" systems. Vertical stacking offers the advantage of producing a chip of only the size of the pixel array while increasing the computing capabilities by using an equivalent area for multiple image processors on a different layer. This might prove to solve the issues associated to fully parallel computational imagers that had to crowd the pixel site with processing units. With vertical interconnection, the processing units can be placed under the pixel on another layer, preserving the imaging performance of a high-end image sensor and preserving the benefits of fully parallel interconnections.

Appendix A

Appendix

A.1 Matlab simulations

A.1.1 Convolution algorithm using the pipeline accumulator

```
function IMGout = convING(T, IMGin);
% function IMGout = convIMG(T, IMGin);
%
% 9x9 convolution of image IMG with template T
% IMG     : greyscale input image
% T       : 9x9, greyscale template
% convIMG : output image of same size than input image.

% Create the temporary processing image from the input image

% Extract the size (Height, Width) of the input image
[Hin, Win] = size(IMGin);

% Define the size (Height, Width) of the image to process;
%   add a 4-pixel padding around the input image;
%   fill the padding with a border of '0's for processing
Hproc = Hin + 8;
Wproc = Win + 8;
IMGproc = zeros(Hproc, Wproc);
IMGproc([5:Hin+4],[5:Win+4]) = IMGin;

% Initialize the output image variable
```

```
IMGnew = zeros(Hproc, Wproc);

% Start the convolution algorithm

% Work on one column at a time as the hardware does
for i = 5 : (Wproc - 4),
   % Initialize the pipeline accumulator memories
   CM1 = zeros(1,10);
   % Each image row is processed one at a time, similar to imagers' readout
   for j = 1 : Hproc,
      % Sum of products between each template row and the current image row
      TI = T * IMGproc(j,[i-4:i+4])';
      % Send the sum of products to the pipeline accumulator
      CM1 = pip9(TI, CM1);
      % Read the output of the accumulator to construct the output image
      IMGnew(j,i) = CM1(10);
   end
end

% Remove the padding and return the output image
IMGout = IMGnew([5:Hin+4],[5:Win+4]);
```

A.1.2 9-stage pipeline accumulator

```
function [CM1] = pip9(X, CM1);
% function [CM1] = pip9(X, CM1);
%
% 9-stage pipeline implementation.
% CM1[2:9] : interface memory
% CM2[1:9] : internal memory
% X[1:9] : input vector

% Compute the values to be saved.
Atmp(1) = X(1);
for i=2:9,
   Atmp(i) = ((i-1)/i) * CM1(i) + (1/i) * X(i);
```

```
    end

    % First step: Load_in
    % CM2_1 = X1
    % CM2_i = (i/i+1)CM1_i + (1/i)Xi

    % The current memories operate on the same clock:
    CM2 = Atmp;

    % Second step: shift on Load_out
    % CM1_i = CM2_{i-1} (i>1)
    % Notes:
    % CM1_1 doesn't exist: set to 0.
    % CM1_10 = output

    CM1 = [ 0 CM2 ];
```

Bibliography

[1] Alexander I. Krymski. A 9-v/lux-s 5000-frames/s 512×512 cmos sensor. *IEEE Transactions on Electron Devices*, 50(1), January 2003.

[2] Stuart Kleinfelder, SukHwan Lim, and Abbas El Gamal. A 10 000 frames/s cmos digital pixel sensor. *IEEE Journal of Solid-State Circuits*, 36(12), December 2001.

[3] A. Ahmed Biyabani, L. Richard Carley, and Takeo Kanade. An analog cmos ic for template matching. In *International Solid-State Circuits Conference*, Digest of Technical Papers, 15-17 February 1999.

[4] Massimo Gottardi and Woodward Yang. A ccd/cmos image motion sensor. In *International Solid-State Circuits Conference*, Digest of Technical Papers, February 1993.

[5] Alberto Pesavento and Christof Koch. Feature detection in analog vlsi. In *Proceedings of the 33rd Asilomar Conference on Signals, Systems and Computers*, October 1999.

[6] Takashi Komuro, Idaku Ishii, Masatoshi Ishikawa, and Atsushi Yoshida. A digital vision chip specialized for high-speed target tracking. *IEEE Transactions on Electron Devices*, 50(1), January 2003.

[7] Christophe Basset, Pietro Perona, Guang Yang, and Bedabrata Pain. Cmos imager with embedded analog early image processor. In *IEEE Workshop on CCDs and Advanced Image Sensors*, May 2003.

[8] Bedabrata Pain, Guang Yang, Monico Ortiz, Kenneth McCarty, Bruce Hancock, Julie Heynssens, Thomas Cunningham, Chris Wrigley, and Charlie Ho. A single-chip programmable digital cmos imager with enhanced low-light detection capability. In *Thirteenth International Conference on VLSI Design*, pages 342–347, 3-7 January 2000.

[9] Hirochika Inoue, Tetsuya Tachikawa, and Masayuki Inaba. Robot vision system with a correlation chip for real-time tracking, optical flow and depth map generation. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1621–1626, 12-14 May 1992.

[10] Andrew Duchon, William H. Warren, and Leslie Pack Kaelbling. Ecological robotics: Controlling behavior with optical flow. In *Proceedings of the 17th Annual Conference of the Cognitive Science Society*, pages 164–169, 22-25 July 1995.

[11] Gary P. Zientara, Pairash Saiviroonporn, Paul R. Morrison, Marvin P. Fried, Stephen G. Hushek, Ron Kikinis, and Ferenc A. Jolesz. Mri monitoring of laser ablation using optical flow. *Journal of Magnetic Resonance Imaging*, 8(6):1306–1318, 1998.

[12] J.L. Barron, S.S. Beauchemin, and D.J. Fleet. On optical flow. In *6th International Conference on Artificial Intelligence and Information-Control Systems of Robots*, pages 3–14, September 1994.

[13] Edward H. Adelson and James R. Bergen. Spatiotemporal energy models for the perception of motion. *Journal of Optical Society of America*, 2(2):284–299, February 1985.

[14] James J. Little and Alessandro Verri. Analysis of differential and matching methods for optical flow. In *Proceedings of the IEEE Workshop on Visual Motion*, pages 173–180, 20-22 March 1989.

[15] José L. Martín, Aitzol Zuloaga, Carlos Cuadrado, Jesús Lázaro, and Unai Bidarte. Hardware implementation of optical flow constraint equation using fpgas. *Computer Vision and Image Understanding*, 98(3):462–490, June 2005.

[16] Berthold K. P. Horn and Brian G. Schunck. Determining optical flow. *Artificial Intelligence*, 17(1-3):185–203, August 1981.

[17] Berthold K. P. Horn. *Robot Vision*. MIT Press, 1986.

[18] Behzad Kamgar-Parsi, Behrooz Kamgar-Parsi, and Azriel Rosenfeld. Optimally isotropic laplacian operator. *IEEE Transactions on Image Processing*, 1999.

[19] Berthold K.P. Horn. Hill shading and the reflectance map. In *Proceedings of the IEEE*, volume 69, pages 14–47, January 1981.

[20] Christof Koch, H. Taichi Wang, Roberto Battiti, Bimal Mathur, and Chris Ziomkowski. An adaptive multi-scale approach for estimating optical flow:computational theory and physiological implementation. In *Proceedings of the IEEE Workshop on Visual Motion*, pages 111–122, 7-9 October 1991.

[21] James R. Bergen, Peter J. Burt, Rajesh Hingorani, and Shmuel Peleg. A three-frame algorithm for estimating two-component image motion. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, volume 14, issue 9, pages 886–896, September 1992.

[22] Peter Ramm, Detlef Bonfert, Horst Gieser, Jürg Haufe, Franz Iberl, Armin Klumpp, Andreas Kux, and Robert Wieland. Interchip via technology for vertical system integration. In *Proceedings of the IEEE 2001 International Interconnect Technology Conference*, pages 160–162, June 4-6 2001.

[23] James Burns, Lisa McIlrath, Craig Keast, Craig Lewis, Andrew Loomis, Keith Warner, and Peter Wyatt. Three-dimensional integrated circuits for low-power, high-bandwidth systems on a chip. In *International Solid-State Circuits Conference*, Digest of Technical Papers, pages 268–269,453, February 5-7 2001.

[24] Peter Benkart, Alexander Kaiser, Andreas Munding, Markus Bschorr, Hans-Joerg Pfleiderer, Erhard Kohn, Arne Heittmann, Holger Huebner, and Ulrich Ramacher. 3d chip stack technology using through-chip interconnects. *IEEE Design and Test of Computers*, 22(6):512–518, November - December 2005.

[25] Haruo Kobayashi, Joseph L. White, and Asad A. Abidi. An active resistor network for gaussian filtering of images. *IEEE Journal of Solid-State Circuits*, 26(5), May 1991.

[26] Andreas G. Andreou and Kwabena A. Boahen. A 590,000 transistor 48,000 pixel, contrast sensitive, edge enhancing, cmos imager - silicon retina. In *Conference on Advanced Research in VLSI (ARVLSI'95)*, pages 225–240, 27-29 March 1995.

[27] Jörg Krammer, Rahul Sarpeshkar, and Christof Koch. Pulse-based analog vlsi velocity sensors. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 44(2), February 1997.

[28] Thomas Spirig, Peter Seitz, Oliver Vietze, and Friedrich Heitger. A smart ccd image sensor with real-time programmable parallel convolution capabilities. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, 44(5), May 1997.

[29] Shang-Yi Lin, Mei-Hui Chen, and Tzi-Dar Chiueh. Neuromorphic vision processing system. *Electronic Letters*, 33:1039–1040, June 1997.

[30] Piotr Dudek and Peter J. Hicks. A general-purpose cmos vision chip with a processor-per-pixel simd array. In *European Solid-State Circuits Conference*, 18-20 September 2001.

[31] Ania Mitros. *A compact system for self-motion estimation.* PhD thesis, California Institute of Technology, 2006.

[32] Alan V. Oppenheim and Ronald W. Schafer. *Discrete-Time Signal Processing.* Signal-Processing Series. Prentice-Hall, 1989.

[33] Carver Mead. *Analog VLSI and Neural Systems.* VLSI Systems Series, Computation and Neural Systems Series. Addison-Wesley, 1989.

[34] Peter M. Athanas and A. Lynn Abbott. Real-time image processing on a custom computing platform. *IEEE Computer*, 28(2):16–25, February 1995.

[35] Arrigo Benedetti, Andrea Prati, and Nello Scarabottolo. Image convolution on fpgas: the implementation of a multi-fpga fifo structure. In *Proceedings of the 24th Euromicro Conference*, volume 1, pages 123–130, August 1998.

[36] Vivian Ward and Marek Syrzycki. Vlsi implementation of receptive fields for smart vision sensors. In *Canadian Conference on Electrical and Computer Engineering*, volume 1, pages 1184–1187, 14-17 September 1993.

[37] Vladimir Brajovic and Takeo Kanade. Computational sensor for visual tracking with attention. *IEEE Journal of Solid-State Circuits*, 33(8), August 1998.

[38] Abbas El Gamal, David X. Yang, and Boyd Fowler. Pixel-level processing: why, what, and how? In Nitin Sampat and Thomas Yeh, editors, *Sensors, Cameras, and Applications for Digital Photography*, volume 3650, pages 2–13, March 1999.

[39] Yoshinori Muramatsu, Susumu Kurosawa, Masayuki Furumiya, Hiraoki Ohkubo, and Yasutaka Nakashiba. A signal-processing cmos image sensor using a simple analog operation. *IEEE Journal of Solid-State Circuits*, 38(1):101–106, January 2003.

[40] Viktor Gruev and Ralph Etienne-Cummings. Implementation of steerable spatiotemporal image filters on the focal plane. *IEEE Transactions on Circuits and Systems-II*, 49(4), April 2002.

[41] Brent Buchanan and Martin Brooke. An experimental evaluation of error spectrum shaping applied to mixed-signal image convolutions. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 50(12):950–962, December 2003.

[42] Achim Graupner, Jörg Schreiter, Stefan Getzlaff, and René Schüffny. Cmos image sensor with mixed-signal processor array. *IEEE Journal of Solid-State Circuits*, 38(6), June 2003.

[43] Ming-Ting Sun, Ting-Chung Chen, and Albert M. Gottlieb. Vlsi implementation of a 16×16 discrete cosine transform. *IEEE Transactions on Circuits and Systems*, 36(4), April 1989.

[44] Erdem Hokenek, Robert K. Montoye, and Peter W. Cook. Second-generation risc floating point with multiply-add fused. *IEEE Journal of Solid-State Circuits*, 25(5), October 1990.

[45] Jian Liang, Russel Tessier, and Oskar Mencer. Floating point unit generation and evaluation for fpgas. In *IEEE Symposium on Field-Programmable Custom Computing Machines*, volume 4, pages 184–194, 9-11 April 2003.

[46] Taek-Jun Kwon, Jeff Sondeen, and Jeff Draper. Design trade-offs in floating-point unit implementation for embedded and processing-in-memory systems. In *International Symposium on Circuits and Systems*, volume 4, pages 3331–3334, 23-26 May 2005.

[47] R Meyer, R Reng, and K Schwarz. Convolution algorithms on dsp processors. In *International Conference on Acoustics, Speech, and Signal Processing*, volume 3, pages 2193–2196, April 1991.

[48] Jagadeesh Sankaran, William Pervin, and Cyrus D. Cantrell. Preferred strategies for optimizing convolution on vliw dsp architectures. In *International Conference on Compilers, Architectures, and Synthesis for Embedded Systems*, pages 41–51, September 2004.

[49] Piotr Brudlo and Krzysztof Kuchcinski. Parallel spatio-temporal convolution scheme oriented for hardware real-time implementation. In *Proceedings of the 23rd Euromicro Conference*, pages 190–195, September 1997.

[50] Valentine C. Aikens II, José G. Delgado-Frias, Gerald G. Pechanek, and Stamatis Vassiliadis. A parallel pipelined dsp processor core. In *IEEE 39th Midwest symposium on Circuits and Systems*, volume 1, pages 81–84, August 1996.

[51] Marcelo Alves de Barros and Mohamed Akil. Low level image processing operators on fpga: implementation examples and performance evaluation. In *Proceedings of the 12th IAPR International Conference on Pattern Recognition*, volume 3, pages 162–167, October 1994.

[52] Linda Kaouane, Mohamed Akil, and Yves Sorel. An automated design flow for optimized implementation of real-time image processing applications onto fpga. In *EUROCON 2003*, volume 1, pages 71–75, Septembre 2003.

[53] A.K Ratha, N.K.; Jain. Computer vision algorithms on reconfigurable logic arrays. In *IEEE Transactions on Parallel and Distributed Systems*, volume 10, pages 29–43, January 1999.

[54] Viktor Hongtu Jiang; Owall. Fpga implementation of real-time image convolutions with three level of memory hierarchy. In *IEEE International Conference on Field-Programmable Technology (FPT)*, pages 424–427, December 2003.

[55] Kenneth W. Mai, Toshihiko Mori, Bharadwaj S. Amrutur, Ron Ho, Bennett Wilburn, Mark A. Horowitz, Isao Fukushi, Tetsuo Izawa, and Shin Mitarai. Low-power sram

design using half-swing pulse-mode techniques. *IEEE Journal of Solid-State Circuits*, 33(11), November 1998.

[56] Lisa G. McIlrath, Vincent S. Clark, Peter K. Duane, R. Daniel McGrath, and William D. Waskurak. Design and analysis of a 512×768 current-mediated active pixel array image sensor. *IEEE Transactions on Electron Devices*, 44(10), October 1997.

[57] Behzad Razavi. *Design of Analog CMOS Integrated Circuits*. McGraw-Hill, preview edition, 2000.

[58] Paul R. Gray and Robert G. Meyer. *Analysis and design of Analog Integrated Circuits*. Wiley, third edition, 1993.

[59] Marvin H. White, Donald R. Lampe, Franklyn C. Blaha, and Ingham A. Mack. Characterization of surface channel ccd image arrays at low light levels. *IEEE Journal of Solid-State Circuits*, SC-9(1), February 1974.

[60] Nicolas Moeneclaey and Andreas Kaiser. Measurement of the main limitations of current memory cells. In *37th Midwest Symposium on Circuits and Systems*, volume 1, pages 54–57, 3-5 August 1994.

[61] Steven J. Daubert, David Vallancourt, and Yannis P. Tsividis. Current copier cells. *Electronic Letters*, 24(25):1560–1562, December 1988.

[62] Barrie Gilbert. A precise four-quadrant multiplier with subnanosecond response. *IEEE Journal of Solid-State Circuits*, SC-3(4), December 1968.

[63] I Házman. Four-quadrant multiplier using mosfet differential amplifiers. *Electronic Letters*, 8:63–65, February 1972.

[64] N. I. Khachab and M. Ismail. Mos multiplier/divider cell for analog vlsi. *Electronic Letters*, 25:1550–1552, November 1989.

[65] Peter B. Denyer, John Mavor, and John W. Arthur. Miniature programmable transveral filter using ccd/cmos technology. In *Proceedings of the IEEE*, volume 67, pages 42–50, January 1979.

[66] Klaas Bult and Hans Wallinga. A cmos four-quadrant multiplier. *IEEE Journal of Solid-State Circuits*, SC-21(3), June 1986.

[67] Francis J. Kub, Keith K. Moon, Ingham A. Mack, and Francis M. Long. Programmable analog vector-matrix multipliers. *IEEE Journal of Solid-State Circuits*, 25(1), February 1990.

[68] Shen-Iuan Liu and Yuh-Shyan Hwang. Cmos four-quadrant multiplier using bias feedback techniques. *IEEE Journal of Solid-State Circuits*, 29(6), June 1994.

[69] Gang Li and Brent Maundy. A novel four quadrant cmos analog multiplier/divider. In *International Symposium on Circuits and Systems*, volume 1, pages 1108 – 1111, 23-26 May 2004.

[70] C.M. Edwards. Survey of analog multiplication schemes. *Journal of the ACM*, 1(1):27–35, 1954.

[71] Gunhee Han and Edgar Sánchez-Sinencio. Cmos transconductance multipliers: A tutorial. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 45(12), December 1998.

[72] Gin-Kou Ma and Fred J. Taylor. Multiplier policies for digital signal processing. *IEEE ASSP Magazine*, 7(1):6–20, January 1990.

[73] Andrew G. Dempster and Macleod Malcom D. Use of minimum-adder multiplier blocks in fir digital filters. *IEEE Transactions on Circuits and Systems*, 42(9), September 1995.

[74] Issam S. Abu-Khater, Abdellatif Bellaouar, and M. I. Elmasry. Circuit technique for cmos low-power high performance multipliers. *IEEE Journal of Solid-State Circuits*, 31(10), October 1996.

[75] Pedram Mokrian, Majid Ahmadi, Graham Jullien, and W.C. Miller. A reconfigurable digital multiplier architecture. In *Canadian Conference on Electrical and Computer Engineering*, volume 1, pages 125–128, 4-7 May 2003.

[76] Kwen-Siong Chong, Bah-Hwee Gwee, and Joseph S. Chang. A micropower low-voltage multiplier with reduced spurious switching. In *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, volume 13, pages 255–265, February 2005.

[77] Alfred Fettweis. On the connection between multiplier word length limitation and roundoff noise in digital filters. *IEEE Transactions on Circuit Theory*, 19(5), September 1972.

[78] Robert H. Walden. Analog-to-digital converter survey and aalysis. *IEEE Journal on Selected Areas in Communications*, 17(4):539–550, April 1999.

[79] Andrew M. Abo and Paul R. Gray. A 1.5-v, 10-bit, 14.3-ms/s cmos pipeline analog-to-digital converter. *IEEE Journal of Solid-State Circuits*, 34(5):599–606, May 1999.

[80] David X. D. Yang, Boyd Fowler, and Abbas El Gamal. A nyquist-rate pixel-level adc for cmos image sensors. *IEEE Journal of Solid-State Circuits*, 34(3):348–356, March 1999.

[81] Kambiz Kaviani, Ömer Oralkan, Pierre Khuri-Yakub, and Bruce A. Wooley. A multichannel pipeline analog-to-digital converter for an integrated 3-d ultrasound imaging system. *IEEE Journal of Solid-State Circuits*, 38(7):1266–1270, July 2003.

[82] Shu-Yuan Chin and Chung-Yu Wu. A 10-b 125-mhz cmos digital-to-analog converter (dac) with threshold-voltage compensated current sources. *IEEE Journal of Solid-State Circuits*, 29(11):1374–1380, November 1994.

[83] J. Jacob Wikner and Nianxiong Tan. Modeling of cmos digital-to-analog converters for telecommunication. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 46(5):489–499, May 1999.

[84] John Hyde, Todd Humes, Chris Diorio, Mike Thomas, and Miguel Figueroa. A 300-ms/s 14-bit digital-to-analog converter in logic cmos. *IEEE Journal of Solid-State Circuits*, 38(5):734–740, May 2003.

[85] Andrea Baschirotto, Nicola Ghittori, Piero Malcovati, and Andrea Vigna. Design trade-offs for a 10 bit, 80 mhz current steering digital-to-analog converter. In *2nd Annual IEEE Northeast Workshop on Circuits and Systems*, pages 249–252, 20-23 June 2004.

[86] Ying Yao, Mayurachat N. Gulari, Jamille F. Hetke, and Kensall D. Wise. A low-profile three-dimensional neural stimulating array with on-chip current generation. In *26th Annual International Conference of the Engineering in Medicine and Biology Society*, volume 3, pages 1994–1997, 1-5 September 2004.

[87] Glenn E.R. Cowan, Robert C. Melville, and Yannis P. Tsividis. A vlsi analog computer/digital computer accelerator. *IEEE Journal of Solid-State Circuits*, 41(1):42–53, January 2006.

[88] Bing J. Sheu and Chenming Hu. Switch-induced error voltage on a switched capacitor. *IEEE Journal of Solid-State Circuits*, sc-19(4), August 1984.

[89] George Wegmann, Eric A. Vittoz, and Fouad Rahali. Charge injection in analog mos switches. *IEEE Journal of Solid-State Circuits*, sc-22(6), December 1989.

[90] Christoph Eichenberger and Walter Guggembühl. Dummy transistor compensation of analog mos switches. *IEEE Journal of Solid-State Circuits*, 24(4), August 1989.

[91] Hui Tian, Xinqiao Liu, SukHwan Lim, Stuart Kleinfelder, and Abbas El Gamal. Active pixel sensors fabricated in a standard $0.18\mu m$ cmos technology. In John Blouke, Morley M.and Canosa and Nitin Sampat, editors, *Proceedings of SPIE. Sensors and Camera Systems for Scientific, Industrial, and Digital Photography Applications II*, volume 4306, pages 441–449, May 2001.

[92] Satoshi Yoshihara, Yoshikazu Nitta, Masaru Kikuchi, Ken Koseki, Yoshiharu Ito, Yoshiaki Inada, Souichiro Kuramochi, Hayato Wakabayashi, Masafumi Okano, Hiromi Kuriyama, Junichi Inutsuka, Akari Tajima, Tadashi Nakajima, Yoshiharu Kudoh, Fumihiko Koga, Yasuo Kasagi, Shinya Watanabe, and Tetsuo Nomoto. A 1/1.8-inch 6.4 mpixel 60 frames/s cmos image sensor with seamless mode change. *IEEE Journal of Solid-State Circuits*, 41(12):2998–3006, December 2006.

[93] Bedabrata Pain and Bruce R. Hancock. Accurate estimation of conversion gain and quantum efficiency in cmos imagers. In Nitin Blouke, Morley M.and Sampat and Ricardo J. Motta, editors, *Proceedings of SPIE. Sensors and Camera Systems for Scientific, Industrial, and Digital Photography Applications IV*, volume 5017, pages 94–103, May 2003.

[94] James C. Mullikin, Lucas J. van Vliet, Hans Netten, Frank R. Boddeke, G. van der Feltz, and Ian T. Young. Methods for ccd camera characterization. In Helen C. Titus and Amir Waks, editors, *Image Acquisition and Scientific Imaging Systems*, volume 2173, pages 73–84, May 1994.

[95] Abbas El Gamal, Boyd Fowler, Hao Min, and Xinqiao Liu. Modeling and estimation of fpn components in cmos image sensors. In Morley M. Blouke, editor, *Solid State Sensor Arrays: Development and Applications II*, volume 3301, pages 168–177, April 1998.

[96] Boyd Fowler, Abbas El Gamal, David X. D. Yang, and Hui Tian. Method for estimating quantum efficiency for cmos image sensors. In Morley M. Blouke, editor, *Solid State Sensor Arrays: Development and Applications II*, volume 3301, pages 178–185, April 1998.

[97] Hui Tian, Boyd Fowler, and Abbas El Gamal. Analysis of temporal noise in cmos photodiode active pixel sensor. *IEEE Journal of Solid-State Circuits*, 36(1):92–101, January 2001.