SCAMP-3: A Vision Chip with SIMD Current-Mode Analogue Processor Array

Piotr Dudek

Abstract In this chapter, the architecture, design and implementation of a vision chip with general-purpose programmable pixel-parallel cellular processor array, operating in single instruction multiple data (SIMD) mode is presented. The SIMD concurrent processor architecture is ideally suited to implementing low-level image processing algorithms. The datapath components (registers, I/O, arithmetic unit) of the processing elements of the array are built using switched-current circuits. The combination of a straightforward SIMD programming model, with digital microprocessor-like control and analogue datapath, produces an easy-to-use, flexible system, with high-degree of programmability, and efficient, low-power, small-footprint, circuit implementation. The SCAMP-3 chip integrates 128×128 pixel-processors and a flexible read-out circuitry, while the control system is fully digital, and currently implemented off-chip. The device implements low-level image processing algorithms on the focal plane, with a peak performance of more than 20 GOPS, and power consumption below 240 mW.

1 Introduction

The basic concept of a 'vision chip' or a 'vision sensor' device is illustrated in Fig. 1. Unlike a conventional computer vision system, which separates image acquisition and image processing, the vision chip performs processing adjacent to the sensors, on the same silicon die, producing as outputs pre-processed images, or even higher-level information, such as lists of features, indicators of presence and locations of specific objects or other information extracted from the visual scene. The advantages of this approach include the increase of sensor/processor data bandwidth, and associated reduction in power consumption and increase in the processing throughput.

P. Dudek (✉)
The University of Manchester, Manchester M13 9PL, UK
e-mail: p.dudek@manchester.ac.uk

Á. Zarándy (ed.), *Focal-Plane Sensor-Processor Chips*,
DOI 10.1007/978-1-4419-6475-5_2,

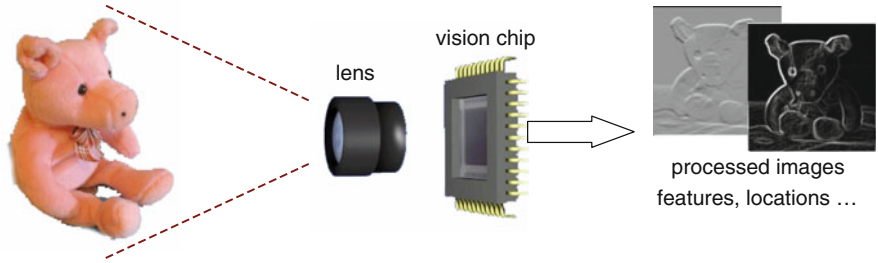


Fig. 1 The concept of a 'vision chip'. Unlike a conventional image sensor, it outputs information extracted from the images

The idea of processing images on the focal plane, and its associated benefits, resulted over the years in many vision chip implementations [1]. From the functionality point of view, two approaches are taken: one is to produce an application-specific (or task-specific) device, performing a particular image processing operation, and the other is to construct a device based on some programmable computing hardware that can be customised to the application by software. While the application-specific devices may be highly optimised at the circuit level for performing a particular task (e.g. optical flow measurements [2], a 3×3 convolution kernel [3]) in practice, their application domain is restricted by their limited functionality. While an application-specific processing circuit may demonstrate performance or power advantages in a particular task, it is rare that only a single image processing operation (e.g. a simple filter) is required in an application. More often, a vision application requires a number of low-level image pre-processing operations, and some higher-level object-based operations, to be performed on each frame of the video stream. Hence, the advantages offered by a custom task-specific device over a more general digital processing hardware (which will usually be included in a complete vision system anyway) are rarely sufficient to merit the use of task-specific vision chips in practical applications. Such devices, despite the demonstrated high performance or efficiency figures, have thus so far largely remained confined to academic exercises and not adopted as real-world engineering solutions.

On the other hand, vision sensors combining on the focal plane, the image sensor array and general-purpose digital processors have been a subject of a number of commercial developments, and found their way into numerous applications [4, 5]. Such devices offer some of the benefits of focal-plane processing (low-power due to near-sensor processing, reduction in I/O bandwidth between the sensor/processor device and the rest of the system), but also flexibility and programmability of a digital microprocessor, and ability to perform a large number of different image processing operations on-chip. However, the current commercial digital vision processors do not make use of the fully pixel-parallel, one processor per pixel architecture, shown in Fig. 2, that conceptually offers the most optimal structure for performing pixel-parallel low-level image processing operations. Instead, they typically integrate a number of processors in a 1D array, one processor per column

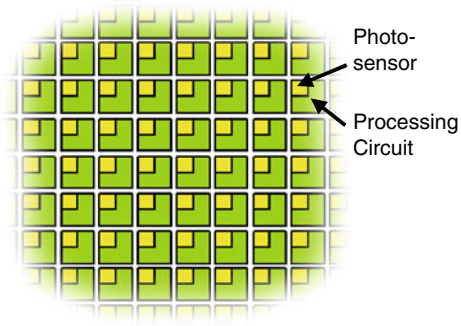


Fig. 2 Pixel-parallel vision chip integrates image processing circuit in each pixel of the image sensor array

of pixels [5, 6]. The reason for this is the difficulty in integrating a complete programmable processor within a small pixel area in a focal-plane array. There have been a few research projects where digital in-pixel processing, having a separate processing element (PE) for each pixel in the array, has been used; from the early devices incorporating rather limited functionality, with a few memory bits per pixel and simple logic operations [7–10], to more recent devices [11], including our developments of asynchronous/synchronous vision chips [12, 13]. The technological progress of CMOS scaling and the availability of vertically stacked 3D integrated devices promise further developments in this area [14]. Nevertheless, the ability to implement reasonably complex processing cores in a small circuit area of a pixel still remains the main challenge. Another challenge is the low-power consumption requirement for a PE that is to be integrated in a massively parallel array of thousands (or millions) of pixel-processors in a single chip. A number of pioneering developments [15–17], including devices described in other chapters of this book, have investigated the issue of integrating analogue processing cores, which would offer better power/performance and area/performance ratios than digital designs, while still offering a degree of programmability usually associated with the digital processing approach.

1.1 SIMD Cellular Processor Arrays

The idea of a Cellular Processor Array – a system integrating a large number of simple PEs, organised in a regular network (typically, especially in the systems considered for image processing, placed on a 2D grid), with nearest-neighbour communication – has been considered from the early days of computing. It can be traced back to von Neumann's work on Cellular Automata [18] and later work on massively parallel arrays of processors by Unger [19] and Barnes [20]. With the introduction of custom-integrated circuits, massive parallelism has become feasible, and the research resulted in the development of machines such as CLIP 4 [21],

DAP [22], MPP [23] and CM-1 [24], which used very simple processors working concurrently performing identical instructions on local data. This mode of operation, known as single instruction multiple data (SIMD), where multiple parallel processing units execute identical program operations on their individual data streams, has been popular in early parallel processor designs. It was used in 'vector' processing units of supercomputers, and has recently evolved as a 'streaming' processing mode used in commodity processors (e.g. in Intel's Streaming SIMD Extensions, IBM/Toshiba/Sony Cell Broadband Engine processor or NVIDIA GPUs). In contrast to 'fine grain' parallelism of SIMD, the 'coarse grain' parallel processing architectures generally adopt a more flexible multiple instruction multiple data (MIMD) style, in which a parallel system consists of multiple independent processing cores, executing independent programs, communicating through a shared memory or a message-passing interface.

Nevertheless, in specific application domains, where a large number of data items undergo identical operations, the SIMD mode provides the most optimal solution, using a multitude of datapath units (PEs), while sharing a single controller that issues instructions to these PEs. Low-level image processing is an example of such application, with inherent massive data parallelism. A typical operation, such as a 3×3 sharpening filter, edge detector, median filter, etc. involve executing identical instructions on every pixel in the image, and producing an output that depends on the value of the pixel, and pixels in its immediate neighbourhood. Such algorithms are easily and naturally mapped onto 2D processor arrays, providing the greatest possible speedup and the simplest control sequence if each processor is associated with one image pixel. The spatial organization of processors can be used to great advantage, minimizing the number of operations required to perform the task. The vision chip design described in this chapter builds upon the ideas of SIMD image processing, providing effective circuit and system level solutions for the design of fine-grain SIMD cellular processor arrays.

1.2 Chapter Overview

The remainder of this book chapter overviews the architecture and design of the SIMD current-mode analogue matrix processor (SCAMP) device that has been designed to 'emulate' the performance of a digital processor array, providing the same degree of flexibility and programming philosophy, while using compact and power-efficient analogue datapath elements. First, the architecture of pixel-parallel cellular processor array is overviewed, explaining how low-level image processing algorithms are mapped onto cellular SIMD arrays. Then the concept of a processor with analogue data path is introduced, its switched-current circuit implementation is explained and its instruction-level operation is described. Finally, the design and implementation details of the SCAMP-3 chip are presented, and application examples are shown.

2 SCAMP-3 Architecture

To understand the SCAMP-3 system, basic concepts of pixel-parallel SIMD approach to image processing have to be appreciated. They are introduced in the next section, followed by the description of the PE architecture implemented on SCAMP-3 and details of the I/O and control system architecture.

2.1 Pixel-Parallel SIMD Array

The three basic concepts described here are common to all SIMD cellular processor arrays, but will be described introducing the mechanisms and nomenclature used in the SCAMP-3 processor. First is the basic idea of pixel-parallel operation. Second is the idea of a nearest-neighbour communication (a 'NEWS' register). Third is the idea of conditional operation based on a local activity flag (a 'FLAG' register).

2.1.1 Array-Wide Operations

The basic concept of a pixel-parallel SIMD cellular processor array is that operations are performed on all array elements (e.g. image pixels) at once. The architecture of the 2D PE array is shown in Fig. 3. Each PE may contain a number of local *registers* to store data (let us assume these registers can store a real number, such as a grey-level pixel intensity). If we label registers of a PE at location (x, y) as A_{xy}, B_{xy}, C_{xy}, etc., then we can imagine register arrays **A**, **B**, **C**, etc. being formed from corresponding registers taken from all PEs, as shown in Fig. 4.

Each PE is responsible for processing individual array elements, using the arithmetic logic unit (ALU) that represents the PEs capability to perform operations on

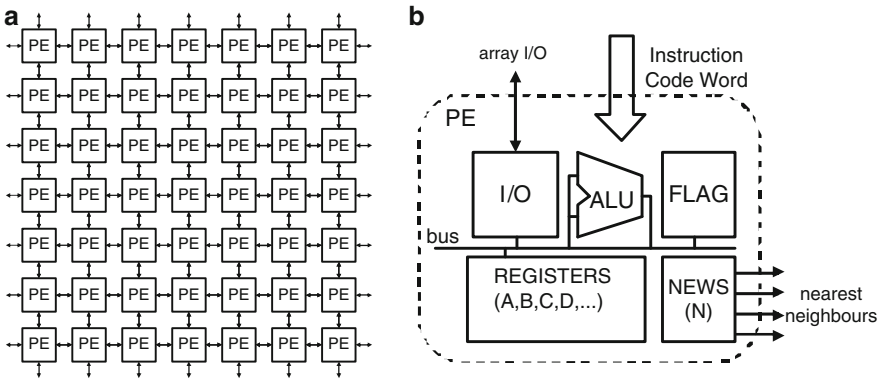


Fig. 3 Architecture of a pixel-parallel SIMD array. (**a**) 2D arrangement of processing elements (PEs), with nearest-neighbour connectivity; (**b**) a single PE

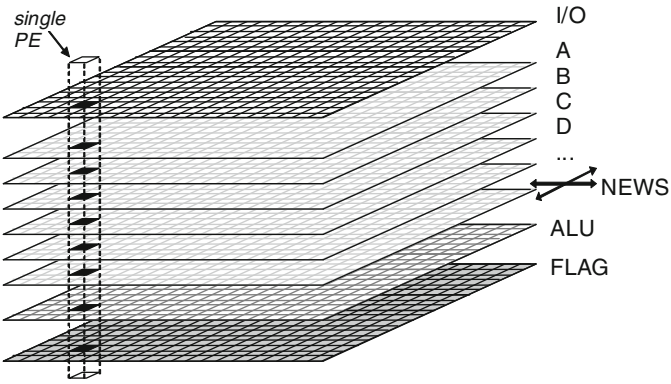
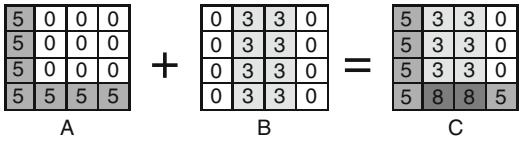


Fig. 4 Array processor view of the SIMD architecture

Fig. 5 Array-wide arithmetic operation



data stored in registers, according to the specific instruction set. As all PEs execute the same program, that is there is one controller in the system, issuing the same 'Instruction Code Words' to all PEs, effectively we obtain array-wide operations. For example, if we load array **A** with one image, and array **B** with another image, a single instruction '$\mathbf{C} = \mathbf{A} + \mathbf{B}$' will add the two images and put the result in register array **C** (see Fig. 5). In the following, we will simply refer to the register arrays **A**, **B**, **C**, etc. as *registers*.

The element-wise array operations are a straightforward concept that will be familiar to anyone, who has used vector/matrix-based programming languages, such as Matlab. It is important to realize though that instead of internally looping through all array elements (as would be the case on a sequential computer), the underlying pixel-parallel SIMD hardware architecture inherently supports such array operations. The individual PE at location (x,y) performs a scalar operation $C_{xy} = A_{xy} + B_{xy}$. This is done concurrently in all PEs, resulting in an array-wide addition operation $\mathbf{C} = \mathbf{A} + \mathbf{B}$.

2.1.2 Nearest-Neighbour Communication

PEs in a cellular processor array can communicate with nearest neighbours on the grid. In the SCAMP-3 system, a 4-neighbour connectivity is used, and the communication is carried out through a special neighbour communication register **N**. The contents of this register can be loaded from any other register, for

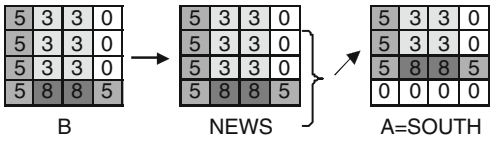


Fig. 6 Neighbour communication operation

example in each PE $N_{xy} = A_{xy}$, but can be accessed by the neighbouring PEs when loading to another register, for example $A_{xy} = N_{x+1,y}$. This operation, when performed in all pixels at once, corresponds to shifting the entire array of data by one column (or one row), that is a one-pixel translation of the image.

By convention, the communication register **N** is called a '**NEWS** register', and when referring to the array operations geographic direction names are used to denote the contents of this register shifted by one pixel, for example assigning $A_{xy} = N_{x+1,y}$ in each array element is denoted as $\mathbf{A} = \mathbf{EAST}$; assigning $A_{xy} = N_{x,y-1}$ corresponds to $\mathbf{A} = \mathbf{NORTH}$, and so on.

As an illustration, assume an image is loaded in register **B**, and consider the following sequence of operations:

NEWS = B
A = SOUTH

The result of this sequence is illustrated in Fig. 6. In the first instruction, the **NEWS** register is loaded with corresponding pixels from image **B**. In the second instruction, register **A** is loaded with the data from the **NEWS** register so that each element takes a value of its **SOUTH** neighbour. As a result, the overall image in **A** appears shifted by one pixel up. (An obvious issue is that of a boundary condition; in the simplest case, the cells that do not have a neighbour are simply loaded with zeros, other solutions may include cyclic or zero flux boundary conditions).

As an example, consider a simple vertical edge detection that can be performed by subtracting the image from its horizontally shifted version. This can be achieved by the following program:

NEWS = B
A = B-EAST

The operation of this program (assuming zero is shifted from the boundary) is illustrated in Fig. 7.

Using the neighbour data transfer concept, filters based on convolution kernels can be easily implemented. It should be noted that many convolution kernels can be implemented in a compact way on the pixel-parallel processor arrays, in a few instructions, exploiting the kernel symmetry and decomposition possibilities. For example, consider convolution $\mathbf{A} = \mathbf{k}^{*}\mathbf{P}$ of the image **P** with a horizontal Sobel edge detection kernel **k**:

$$A_{xy} = \sum_{j=1}^{3}\sum_{k=1}^{3} k_{jk} P_{x+j-2,y+k-2}; \quad \mathbf{k} = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \tag{1}$$

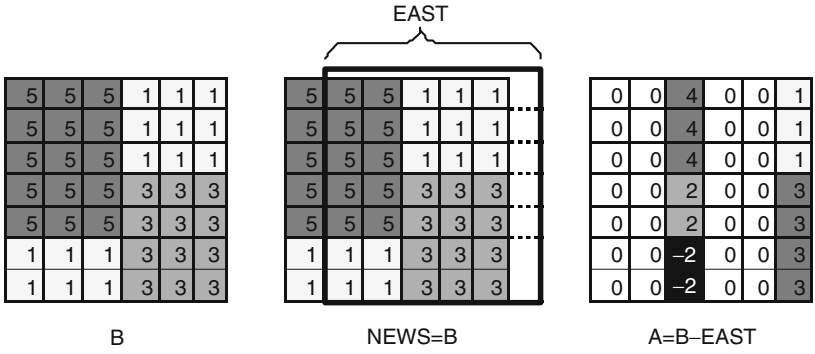


Fig. 7 Vertical edge detection example

To implement this filter, the following operations can be executed in a pixel-parallel way:

NEWS = P
A = 2*P + EAST + WEST
NEWS = A
A = SOUTH − NORTH

Register **A** is first used as a temporary array, and ultimately it stores the result of the convolution operation. The first two instructions implement the convolution of the image with a kernel [1 2 1], the following two instructions take the result and apply the kernel $[-1\ 0\ 1]^{\mathrm{T}}$, producing the final result. (Note that the actual machine-level instruction implementation of the above procedure on the SCAMP-3 system has to take into account specific instruction set restrictions, such as the fact that only one neighbour access is allowed in one instruction, every elementary instruction leads to value negation and carries an offset error that has to be cancelled out, no multiplication is available, etc. The procedure shown above has to be thus compiled into assembly code of about 30 machine-level instructions.)

2.1.3 Local Activity Flag

The third important concept of SIMD cellular processing is that of local activity flag. While all PEs in the array receive the same instruction stream, it is often required to provide some degree of local autonomy, so that different operations can be performed on different elements of the array, in data-dependent fashion.

For example, consider the thresholding operation. The output of this operation can be described as:

$$A_{xy} = \begin{cases} 1 & \text{if } P_{xy} > T \\ 0 & \text{if } P_{xy} \leq T \end{cases} \tag{2}$$

that is, the output register **A** should be loaded with '1' in these PEs that have their register **P** value above the threshold value T, and loaded with '0' in the other PEs. We could imagine that a following program executing in each PE would implement this operation:

```
if P > T then A = 1
else A = 0
end if
```

After the conditional instruction '**if**' the program branches, some PEs need to perform the assignment instruction '**A = 1**', others have to perform '**A = 0**'. However, the SIMD architecture stipulates that a single controller issues the same instructions to all PEs in the array, that is each PE receives exactly the same instruction to be executed at a given time. It is hence impossible to branch and perform different instructions in different PEs of the array at the same time.

To solve this problem, and implement conditional branching operations, a concept of a *local activity flag* is introduced. Each PE contains a one-bit FLAG register. As long as this FLAG is set, the PE is active (enabled) and executes the instructions issued by the controller. When the FLAG is reset, the PE becomes inactive (disabled), and simply ignores the instruction stream. The FLAG can be set or reset in a single conditional data-dependent instruction, and the instructions setting/reseting the FLAG are the only ones that are never ignored (so that the inactive PEs processors can be activated again).

The example thresholding operation can be thus implemented as follows:

```
A = 1
if (P > T) reset FLAG
A = 0
set FLAG
```

The execution of this program is illustrated in Fig. 8. The same instructions are delivered to every PE in the array; however, the instruction '**A = 0**' is only executed in those PEs that have the FLAG still activated after the conditional reset.

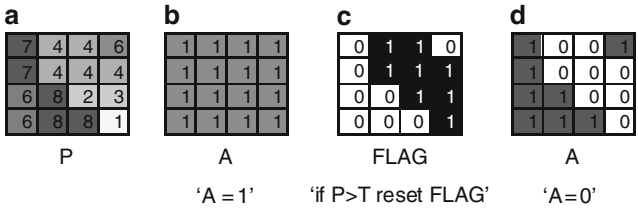


Fig. 8 Conditional code execution (processor autonomy) using local activity flag: (**a**) register P contains the input data, (**b**) assuming all PEs are active (FLAG = 1), register A of all PEs is loaded with 1, (**c**) comparison operation, PEs that have P > T (assume T = 5 in this example) become inactive (FLAG = 0), (**d**) register A is loaded with 0 only in active PEs

It should be noted that no additional flexibility is allowed in terms of conditional branching in SIMD arrays. In particular, if the control program contains loops or jumps, these cannot be conditional on local PE data, as the same instruction stream has to be applied to all PEs. The only local autonomy is possible by masking individual PEs using the FLAG register, so that they do not execute certain instructions in the program.

2.2 Processing Element

The block diagram of the PE implemented on the SCAMP-3 chip is shown in Fig. 9 The classical SIMD mechanism of array-wide operation, neighbour communication, and local activity flag, described above, form the basis of the SCAMP-3 architecture. The PE contains nine registers, capable of storing analogue data values (e.g. pixel intensity values, edge gradient information, gray-level filtering results). The general-purpose registers are labeled A, B, C, D, H, K, Q and Z. The registers are connected to a local bus. The NEWS register can be also connected to local buses of the four neighbours.

The PE contains a one-bit local activity FLAG. Control signals that form the instruction code words (ICWs) are broadcast to the PEs from the external controller. The ICWs determine the type of data operation, and select individual registers to be read-from or written-to. The write control signals for all registers are gated by the contents of the FLAG, so that no register write operation is performed when FLAG is reset. This is sufficient to implement the local autonomy, as gating write control signals ensures that the state of the PE does not change as a result of broadcast instructions when the PE is in the disabled state.

The PIX register contains the pixel intensity value acquired by the photosensor intergrated within the processor. A single PE in the array is associated with a single pixel of the image sensor, that is the SCAMP-3 vision chip implements a fully pixel-parallel fine grain SIMD focal plane processor array.

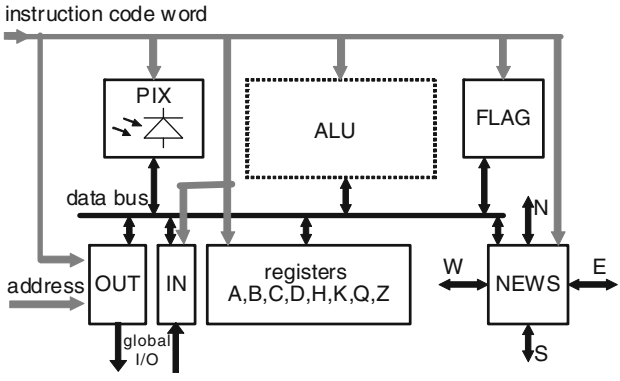


Fig. 9 Processing element of the SCAMP-3 chip

The ALU block represents the instruction-set of the PE. The PE is capable of the following basic operations: negation, addition/summation, division by two (and three), conditional FLAG reset based on comparison with zero, and unconditional FLAG set. The results of the operations are stored in registers, and the arguments of the operations are registers, or register-like components (PIX, IN) representing the pixel-parallel input to the array. The IN register represents the global input, for example immediate argument for a register load instruction.

2.3 Array Structure and Global I/O

The overall system integrated on the chip contains an array of PEs, read-out circuits and control-signal distribution circuits, as shown in Fig. 10. The PE array supports random addressing. It is organized in rows and columns, and the address decoders are placed on the periphery of the array. The PE can output data via the global I/O port to the array column bus, when addressed by the row select signal, and then to the output port as selected by the column address signal. The output data is a register value (contents of any register can be read-out to the column line), or the FLAG register bit. The array can be scanned to read-out the entire register array (e.g. the output image).

A further computational capability is offered by the global read-out operations. When multiple PEs are addressed and output their data to the column lines, the PE array performs the summation operation. In particular, addressing all pixels in the array simultaneously results in the calculation of the global sum of all values in a particular array register. This can be used, for example, to perform pixel

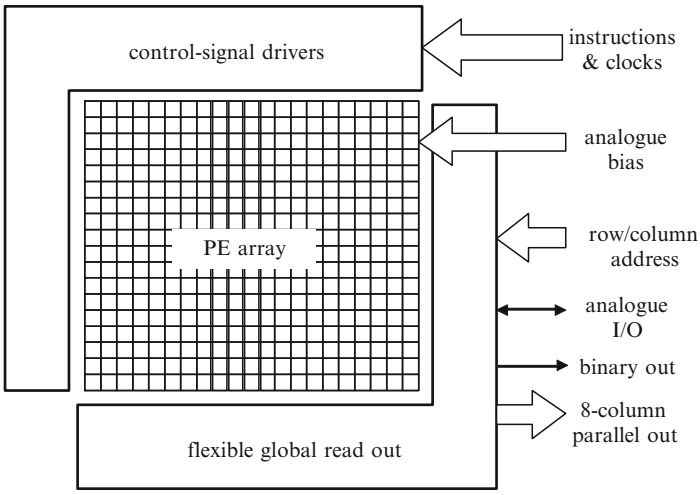


Fig. 10 SCAMP-3 chip

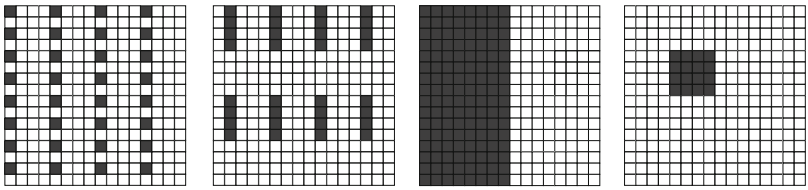


Fig. 11 Flexible addressing scheme utilising *don't care* row/column addresses for selecting multiple PEs at the same time. The address (0, 0) points to the *bottom left* corner of the array, addressed PEs are *shaded*

counting operations (e.g. for computing histograms) or calculating some other global measures (e.g. the average brightness of the image for gain control, or overall 'amount' of motion in the image). Indeed, it is expected that when the device is used in a machine vision system, the results of global operations will often be the only data that is ever transmitted off chip, as these results will represent some relevant information extracted from the image.

Global logic operations are also supported. When reading-out the state of the FLAG register, and addressing multiple PEs at the same time, a logic OR operation is performed. Such an operation can be effectively used, for example to test the image for existence of a certain feature, or to detect a stop condition that controls the number of iterations in an algorithm.

Multiple PE selection for read-out is implemented with addressing scheme that uses *don't care* symbols as a part of the row/column address, as illustrated in Fig. 11. In practice, this is achieved by using two words to produce the address, one sets the actual address bits (e.g. '00110000') the second sets the *don't care* attribute for individual address bits (e.g. '00001111') producing together the combined word that addresses many pixels at once ('0011XXXX' in this example). The addressing of groups of pixels can be utilized in various ways. An example is the pixel-address finding routine that starts by addressing half of the array, performing a global OR operation to determine whether the pixel is in the selected region, then selecting the quarter of the array (in the half that contains the pixel, as determined in the previous step), then 1/8th, and so on, finally homing on the pixel of interest. In this way, an address of an active pixel can be determined in just a few bisection steps, without the need for reading-out (scanning) the entire image from the chip [25].

2.4 Control System

The PE array needs to be provided with the sequence of instructions to be executed. These are broadcast to all PEs in the array from a single controller. The control system is also responsible for read-out addressing, interfacing to external devices, and

```
C ←A                ; using C as a temporary register…
B ← C               ; …transfer array A (original image) to array B
load s, 10          ; load control variable s with 10
: loop              ; address label
NEWS ← B            ; transfer array B to the NEWS array
B ← EAST            ; transfer EAST to B (shift NEWS by one pixel)
sub s, 1            ; subtract 1 from s
jump nz, loop       ; loop, unless s is zero (counts 10 iterations)
C ←A + B            ; add A (original image) and B (shifted by 10 pixels)
```

Fig. 12 An example SIMD program. The program shifts an image by 10 pixels to the left, and superimposes the two images (adds original and shifted together). The operations in *italic* type are executed in the controller CPU. The operations shown in **bold** type are executed in the PE array – the controller broadcasts these instructions to the PE array. A, B and NEWS are array registers, operated in the PE datapath, while variable s is mapped to a scalar register of the controller CPU datapath

other control duties (setting up biases, etc.). A digital controller could be included on the same chip as the SIMD array; however on the SCAMP-3 prototype device, this is an external circuit, implemented on an FPGA, using a modified microcontroller core [26].

The main function of the controller is to supply the sequence of ICWs to the PEs in the array. The controller can also execute conditional and unconditional jumps, loops and arithmetic/logic operations on a set of its own scalar registers (e.g. in order to execute variants of the code based on user settings or other inputs to the system, or, for example, processing the result of a global array operation, and performing conditional program flow control based on this result). It has to be remembered that the program flow control is global. All PEs will be executing the same instructions, with the FLAG mechanism available to mask parts of the array. In practice, the program is thus a mixture of *array* operations, which are executed in the massively parallel PE datapath, and *control* operations, which are executed in the controller CPU. A simple example program, consisting of both types of operations is shown in Fig. 12.

3 The Analogue Processor

The pixel-parallel SIMD processor architecture, outlined above, provides the framework for designing a general-purpose ‘vision chip’ device. The integration of physical image sensor in each PE makes the incident image pixel data immediately accessible to the PE. A variety of low level image processing algorithms can be implemented in software. The main challenge, however, lies in the efficient implementation of the 2D PE array system in a silicon device. The solution adopted in the design of the SCAMP-3 chip, based on our idea of an analogue sampled-data processor [27], is outlined in this section.

If a reasonably high resolution device is to be considered, then the individual PE circuitry has to be designed to a very stringent area and power budget. We are interested in vision sensor devices of resolutions in the range of 320×240 pixels, with a pixel pitch below $50\,\mu$m, and typical power consumption in the milliwatt range. A high-performance low-power devices with lower resolutions (e.g. 128×128 or 64×64 pixels) could still be useful for many applications (e.g. consumer robotics and automation, toys, 'watchdog' devices in surveillance). Pixel sizes in the range of $50\,\mu$m have further applications in read-out integrated circuits for infrared focal plane arrays. While using deep sub-micron technologies makes it feasible to design digital processors to these specifications (typically using a bit-serial datapath), we have proposed to use analogue circuitry to implement the PE [27, 28], which can be implemented with a very good power/performance ratio in an inexpensive CMOS technology. It is well known that analogue signal processing circuits can outperform equivalent digital ones. However, while exploiting the analogue circuit techniques for superior efficiency, we still want to retain the advantages of a software-programmable general-purpose computing system.

The basic concept that allows the achievement of this goal is based on the insight that the stored-program computer does not have to operate with a digital datapath. We can equally well construct a universal machine, that is a device operating on data memory, performing data transformations dictated by the sequence of instructions fetched from the program memory, using an analogue datapath. In this scenario, the data memory is stored in analogue registers, transmitted through analogue data buses, and transformed using analogue arithmetic circuits, while the architecture remains microprocessor-like, with a fixed instruction set, and the functionality determined by a software program, and the control (program memory, instruction fetch, decode and broadcast) implemented using digital circuits. We called such a device 'the analogue microprocessor' [27].

3.1 Switched-Current Memory

An efficient way to design an analogue processor is to use switched-current circuit techniques [29]. A basic SI memory cell is illustrated in Fig. 13. When an MOS transistor works in the saturation region its drain current I_{ds} can be, in a first-order approximation, described by the equation:

$$I_{ds} = K(V_{gs} - V_t)^2 \tag{3}$$

where K is the transconductance factor, V_t is the threshold voltage and V_{gs} is the gate-source voltage.

The SI memory cell remembers the value of the input current by storing charge on the gate capacitance C_{gs} of the MOS transistor. The operation of the memory cells is as follows. When writing to the cell, both switches, S and W, are closed (Fig. 13b). The transistor is diode-connected and the input current i_{in} forces the gate-source

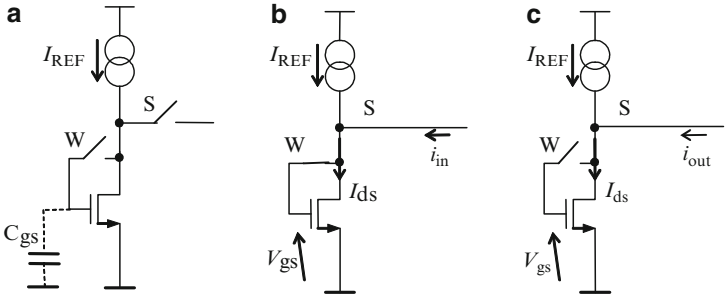


Fig. 13 Basic SI memory cell. (**a**) during storage both switches are open, (**b**) both switches are closed when the cell is 'written to', (**c**) only switch 'S' is closed when the cell is 'read-from'

voltage V_{gs} of the transistor to the value corresponding to the drain current $I_{ds} = I_{\text{REF}} + i_{\text{in}}$, according to (3). At the end of the write phase, the switch W is opened and thus the gate of the transistor is disconnected, that is put into a high impedance state. Due to charge conservation on the capacitor C_{gs}, the voltage at the gate V_{gs} will remain constant.

When reading from the memory cell, the switch S is closed and the switch W remains open (Fig. 13c). Now, the transistor acts as a current source. As the gate-source voltage V_{gs} is the same as one that was set during the write phase, the drain current I_{ds} has to be the same (provided that the transistor remains in the saturation region), and hence the output current i_{out} is equal to

$$i_{\text{out}} = I_{ds} - I_{\text{REF}} = i_{\text{in}} \tag{4}$$

Therefore, the SI memory cell is, in principle, capable of storing a continuous-valued (i.e. real) number, within some operational dynamic range, and subject to accuracy limitations that will be discussed later in this chapter.

3.2 Arithmetic Operations

In addition to data storage registers, the PE needs to provide a set of basic arithmetic operations. These can be achieved with a very low area overhead in the current-mode system. Consider a system consisting of a number of registers implemented as SI memory cells, connected to a common *analogue bus*. Each memory cell can be configured (using corresponding switches S and W) to be read from, or written to.

The basic operation is the transfer operation, as shown in Fig. 14a. Register A (configured for reading) provides the current i_{A} to the analogue bus. This current is consumed by register B (configured for writing); register C is not selected, and hence it is disconnected from the bus. Therefore, the analogue (current) data value is transferred from A to B. This transfer is denoted as $\mathbf{B} \leftarrow \mathbf{A}$. According to the current

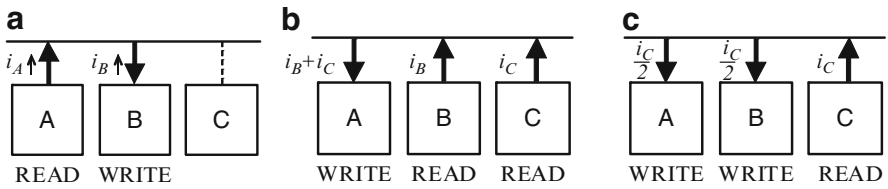


Fig. 14 Basic operations in a current-mode processor: (**a**) register data transfer, (**b**) addition, (**c**) division by two

memory operation, register B will produce the same current when it is read-from (at some later instruction cycle). If we consider that the data value is always the one provided *into the analogue bus*, it can be easily seen that $i_{\mathrm{B}} = -i_{\mathrm{A}}$, that is the basic transfer operation includes negation of the stored data value.

Addition operation (and in general current summation of a number of register values) can be achieved configuring one register for writing, and two (or more) registers for reading. The currents are summed, according to Kirchhoff's current law, directly on the analogue bus. For example, situation shown in Fig. 14b produces operation $\mathbf{A} \leftarrow \mathbf{B} + \mathbf{C}$.

Division operation is achieved by configuring one register cell for reading and many (typically two) registers for writing. The current is split, and if the registers are identical then it is divided equally between the registers that are configured for writing, producing a division by a fixed factor. For example, in Fig. 14c both registers **A** and **B** will store current equal to half of the current provided by register **C**. We denote this instruction as $\mathbf{DIV}(\mathbf{A} + \mathbf{B}) \leftarrow \mathbf{C}$.

This completes the basic arithmetic instruction set of the analogue processor. Multiplication and division by other factors can be simply achieved by multiple application of add or divide by two operations. Subtraction is performed by negation followed by addition. Other instructions (e.g. full-quadrant multiplication) could of course be implemented in hardware using current mode circuits; however, in a vision chip application, silicon area is at premium and a simple instruction set that is sufficient for the intended application should be used. This is achieved here by the basic arithmetic instructions of negation, addition, and divide by two, which are performed entirely in the register/bus system, with no additional circuits. Consequently, more silicon area can be devoted to the register circuits, increasing the amount of local memory (or improving the accuracy of computations, which largely depends on device matching, i.e. device area).

The 'ALU' in the analogue processor, shown as a separate block in Fig. 9 is thus virtual. However, to enable comparison operations, a current comparator (detecting current provided to the analogue bus) should be provided. A combination of arithmetic operations, comparison operations (with programmable threshold) and conditional code execution can be used to perform logic operations in the analogue registers as well.

3.3 Accuracy of SI Circuits

The above descriptions are somewhat simplified, since they ignore the many error effects that affect the result of computation in the analogue processor. Basic switched-current memory storage operation is not entirely accurate, due to errors that are caused by the output conductance of the transistor and the current source, capacitive coupling from the bus to the memory capacitor C_{gs}, clock feedthrough effects due to charge injection onto the memory capacitor from the switches (implemented as MOS transistors) and capacitive coupling from the control signals, and the many noise sources in the circuit. These error effects can be to some extent mitigated using more sophisticated circuits. Extensions of the basic SI technique allow to implement more accurate circuits, with some area overhead. In particular, the S^2I technique proposed by Hughes et al. [30] provides a good trade-off between the accuracy and circuit area.

Overall, the errors cannot be entirely eliminated though, and the current read from the memory cell i_{out} is not exactly the same as the current during the write operation i_{in}, but can be represented as

$$i_{\text{out}} = i_{\text{in}} + \Delta_{\text{SI}} + \varepsilon_{\text{SD}}(i_{\text{in}}) + \varepsilon(^*) \tag{5}$$

where Δ_{SI} is the signal-independent (offset) error, $\varepsilon_{\text{SD}}(i_{\text{in}})$ is the signal-dependent component of the error, and $\varepsilon(^*)$ is the overall random noise error. The actual situation is even more complicated, as the signal dependent error ε_{SD} depends not only on the current value written to the cell, but also on the analogue bus state during the read operation. Nevertheless, through the combination of circuit techniques and careful design, the errors can be made small enough to yield useful circuits in a reasonable circuit area, with accuracies corresponding to perhaps about 8-bit digital numbers.

It is important to point out that, on the one hand, the signal-independent offset errors are very easy to compensate for algorithmically, subtracting offsets during the operation of the program. The signal-dependent errors, are also systematic, and can be to some extent also accounted for by the system, although they are more cumbersome to handle. The random errors, on the other hand, put the limit on the achievable accuracy. The random noise has a temporal component, that is the value of $\varepsilon(^*)$ changes on each memory operation (thermal noise, shot noise, etc.) and a spatial component which represents the variation in the offset and signal-dependent errors in each memory cell in the system due to the component variability. It has to be mentioned that ultimately these random errors limit the accuracy of the analogue processor techniques and scalability of the designs to finer scale CMOS technology nodes.

As shown by (5), each storage operation (and consequently each transfer and arithmetic operation) is performed with some errors. In general, these errors depend on the signal value and type of operation both during register write and subsequent read. In general, most of these errors are small, and simply have to be accepted, as long as they do not degrade the performance of the processor beyond the useable

Table 1 Compensation of signal-independent offset errors

Operation	Instructions	Current transfers showing signal-independent error
Transfer	$C \leftarrow A$	$i_C = -i_A + \Delta_{SI}$
$B := A$	$B \leftarrow C$	$i_B = -i_C + \Delta_{SI} = i_A$
Addition	$D \leftarrow A + B$	$i_D = -(i_A + i_B) + \Delta_{SI}$
$C := A + B$	$C \leftarrow D$	$i_C = -i_D + \Delta_{SI} = i_A + i_B$
Subtraction	$D \leftarrow A$	$i_D = -i_A + \Delta_{SI}$
$C := A–B$	$C \leftarrow B + D$	$i_C = -(i_B + i_D) + \Delta_{SI} = i_A - i_B$
Negation	$C \leftarrow$	$i_C = \Delta_{SI}$
$B := -A$	$B \leftarrow A + C$	$i_B = -(i_A + i_C) + \Delta_{SI} = -i_A$

limit. There are exceptions, however, where simple compensation techniques may significantly improve the accuracy at a small cost in the size of the program.

First, consider the offset error, denoted as Δ_{SI} in (5). This is typically a relatively large systematic error (e.g. several percent of the maximum signal value). However, it can be very easily cancelled out. Since every instruction introduces current negation, the offset error is eliminated using double-instruction macros for transfer, addition, subtraction and negation operations, as illustrated in Table 1.

Second, consider the mismatch error associated with the division instruction. It should be noted that mismatch errors associated with register transfer/storage operations are very small since in the SI memory cell the same transistor is used to copy the current from input cycle to output cycle, and hence it does not matter much if the individual registers are not exactly matched. However, the accuracy of the current splitting in two (as shown in Fig. 14c) relies on matching of the transistors in different cells. In practical designs, the current mismatch of these transistors, given the same terminal voltages, can be as large as several percent. Assuming the mismatch error ε, and ignoring the Δ_{SI} error (that can always be compensated in the way described in Table 1), the division such as $\mathbf{DIV\,A + B \leftarrow C}$ results in

$$i_A = -i_C(1+\varepsilon)/2 \tag{6}$$

$$i_B = -i_C(1-\varepsilon)/2. \tag{7}$$

If an accurate division is required, then the following five-step compensation algorithm should be used. This is based on the scheme proposed in [31], and works by finding the actual error resulting from the division result (note that after $\mathbf{DIV\,A + B \leftarrow C}$ we get $i_B - i_A = \varepsilon i_C$), adding that to the original dividend, and performing mismatched division again. For example, to perform an accurate $\mathbf{A = C/2}$ operation the program shown below should used:

$$\begin{aligned}
\mathbf{DIV\,A + B \leftarrow C};\ & i_A = -i_C(1+\varepsilon)/2 + \Delta, i_B = -i_C(1-\varepsilon)/2 + \Delta \\
\mathbf{H \leftarrow B + C};\ & i_H = -(i_B + i_C) + \Delta = -i_C(1+\varepsilon)/2 \\
\mathbf{D \leftarrow H + A};\ & i_D = -(i_H + i_A) + \Delta = i_C(1+\varepsilon) \\
\mathbf{DIV\,A + B \leftarrow D};\ & i_B = -i_D(1-\varepsilon)/2 + \Delta = -i_C(1+\varepsilon)(1-\varepsilon)/2 + \Delta \\
\mathbf{A \leftarrow B};\ & i_A = -i_B + \Delta = i_C(1-\varepsilon^2)/2.
\end{aligned}$$

At a cost of a few extra instructions and additional local memory usage (in the above, the temporary register D can be replaced by C if its exact value is no longer needed), a significant division error improvement is achieved, theoretically from ε mismatch down to ε^2 error. In practice, secondary error effects influence the result and, for example, on the SCAMP-3 processor the improvement in division accuracy from mismatch-limited value of 2.3% to below 0.2% is achieved.

4 SCAMP-3 Circuit Implementation

The analogue microprocessor idea, combined with the SIMD pixel-parallel architecture, provides foundations for the implementation of an efficient vision chip device. The PE design on the SCAMP-3 chip follows the switched-current processor design concept outlined in the previous section. A simplified schematic diagram of the complete PE is shown in Fig. 15.

4.1 Registers

The register cells have been implemented using the S^2I technique [30], with additional transistors to manage power consumption (switching the DC currents off when a register is not used) and provide conditional 'write' operation (to implement the activity flag operation). The detailed schematic diagram of the register cell is

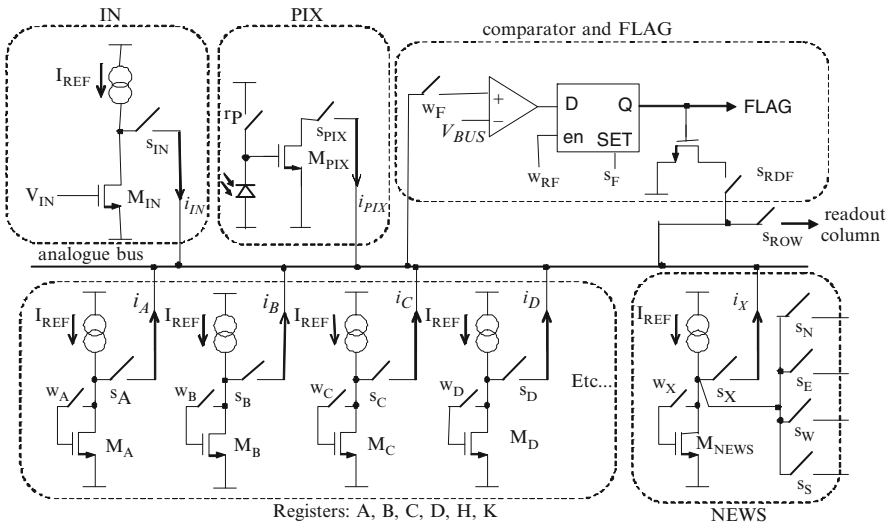


Fig. 15 Simplified schematic diagram of the PE

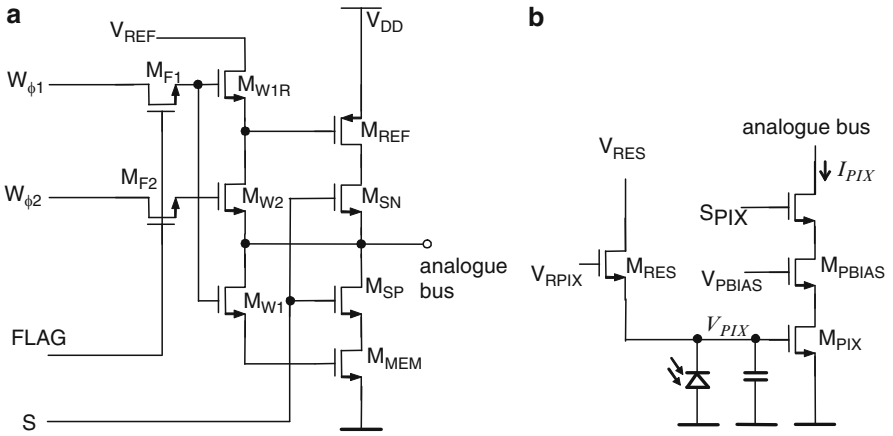


Fig. 16 Schematic diagram of the cells implemented on the SCAMP-3 chip (**a**) register cell, (**b**) pixel circuit

shown in Fig. 16a. M_{MEM} is the storage transistor, M_{REF} is the current source/error storage transistor following the S^2I technique, and M_{W1}, M_{W2} and M_{W1R} are the switches controlling writing/error correction. M_{SN} and M_{SP} connect the register to the analogue bus and shut-down the current path if register is not selected. M_{F1} and M_{F2} are used to gate write-control signals by the FLAG signal.

The NEWS register is build in a similar way to a regular register, but providing output current to the analogue buses of the neighbouring PEs. Furthermore, the four neighbour communication network provides means for executing a fast array-wide diffusion operation (e.g. to low-pass filter or 'blur' the image).

4.2 Photodetector

The photodetector circuit is shown in Fig. 16b. In integration mode, M_{RES} provides a reset signal that charges the capacitance so that $V_{PIX} = V_{RES}$. During photocurrent integration, the capacitance is discharged. The voltage V_{PIX} is transformed to current I_{PIX} (using M_{PIX} biased in linear region with cascode transistor M_{PBIAS}), which is provided to the analogue bus at a required time selecting the PIX register for read-out, that is switching S_{PIX}. Therefore, instruction such as $\mathbf{A} \leftarrow \mathbf{PIX}$ can be executed to sample the pixel current I_{PIX} at some time after the reset. This can be done multiple times during the integration time, and also just after reset (e.g. to perform differential double sampling in order to reduce fixed pattern noise of the imager). A typical frame loop, overlapping integration and program execution is shown in Fig. 17. However, it has to be noted that other possibilities exist (e.g. sampling at multiple times for enhanced dynamic range [32], dynamically adjusting integration time for achieving constant overall brightness, locally adjusting integration time for adaptive sensing).

```
: start
B←PIX          ; sample PIX value to B
RESPIX         ; reset PIX
A←B + PIX      ; subtract the acquired value from the reset value
...            ; user program here
OUT Z          ; output results
: loop start
```

Fig. 17 A typical frame loop. The pixel value is sampled first (assuming it was reset in the previous frame), then the photodetector is reset and the reset level subtracted from the sampled value. The user program is executed, while the photocurrent is integrated for the next frame. The processing results are output (this forces the controller to halt the program and execute a read-out sequence, it can also wait at this point for the next frame-synchronisation pulse, if operation at a fixed frame rate is required)

The pixel circuit also enables a 'logarithmic mode' sensing, which is instantaneous (not integrating). In this mode, voltage V_{RPIX} is constantly applied, and the sub-threshold current of $\mathrm{M_{RES}}$ produces a voltage V_{PIX} (note that V_{RPIX}–V_{PIX} is a gate-source voltage that varies exponentially with current for a transistor in weak inversion).

4.3 Comparator and FLAG

A comparator, connected to the analogue bus, has the capability to determine the overall sign of the current provided to the analogue bus by registers connected to the bus. The result of the comparison operation is latched in the **FLAG** register, build as a static D-type latch. The value stored in the **FLAG** register is then used to gate the write-control signals of all registers (this is done using a single-transistor gates $\mathrm{M_{F1}}$ and $\mathrm{M_{F2}}$, as shown in Fig. 16a), thus ensuring that the state of the processor does not change in response to broadcast instructions when **FLAG** signal is low. The D-latch can be set by a global signal to activate all PEs.

4.4 Input Circuit

The **IN** circuit (see Fig. 15) provides input current to the analogue bus of the PE. The value of this current is set by transistor $\mathrm{M_{IN}}$ driven by a global signal V_{IN} that has to be set accordingly. Currently, an off-chip digital-to-analogue converter and a lookup table are used in the system, to set this value to achieve the desired input current. The **IN** register is used in instructions such as **A** ← **IN**(**35**), which load all registers in the array with a constant value (by convention, numerical value of 35 used in this example corresponds to 35% of the reference current of approximately 1.7 µA).

As all PEs receive the same voltage V_{IN}, the mismatch of transistors M_{IN} and any systematic voltage drops across the array will affect the accuracy of the input current. This can be to some extent reduced using a differential scheme, subtracting the IN(0) value from the desired value, over two cycles, as shown below:

$$\mathbf{A} \leftarrow \mathbf{IN}(\mathbf{0})$$
$$\ldots$$
$$\mathbf{B} \leftarrow \mathbf{A} + \mathbf{IN}(\mathbf{35})$$

The first line loads register A with the input circuit offset error (and the Δ_{SI} offset). The second line subtracts the offsets from the desired input value. Register B is thus loaded with 35% of the bias current.

Although loading individual PEs with different values is possible (through setting FLAG registers of individual PEs in turn, via read-out selection circuit, and then using global input), this mode is not practical and the optical input via the **PIX** circuit remains the primary data source for the chip.

4.5 Output Circuits

The analogue bus of the selected PE (selected through a row select signal) can be connected to a read-out column, and then (depending on the column select signal) to the output of the chip. The current from any of the registers can be thus read-out. If multiple PEs are selected, output currents are summed.

The output of the FLAG register can also be used to drive the read-out column line. An nMOS only drive, with precharged read-out bus, enables OR operation if multiple PEs are selected. In addition, an 8-bit parallel output (reading out eight PEs in one row concurrently) is also provided.

4.6 Silicon Implementation

We have implemented the 128×128 pixels SCAMP-3 chip [33], shown in Fig. 18a. The layout plot showing floorplan of a single PE is shown in Fig. 18b. The device has been fabricated in an inexpensive 0.35 µm 3-metal layers CMOS technology, with PE pitch below 50 µm. The chip size is 54 mm^2, and it comprises 1.9 M transistors. The basic parameters of the device are included in Table 2.

When clocked at 1.25 MHz, the processing array provides over 20 GOPS (Giga operations per second), with individual 'operation' corresponding to one analogue instruction, such as summation or division by two. The chip dissipates a maximum of 240 mW (when continuously executing instructions). When processing video streams at typical frame rates of 30 fps, the power dissipation depends on the length of the algorithm, but can be as low as a few mW for simple filtering or target tracking tasks.

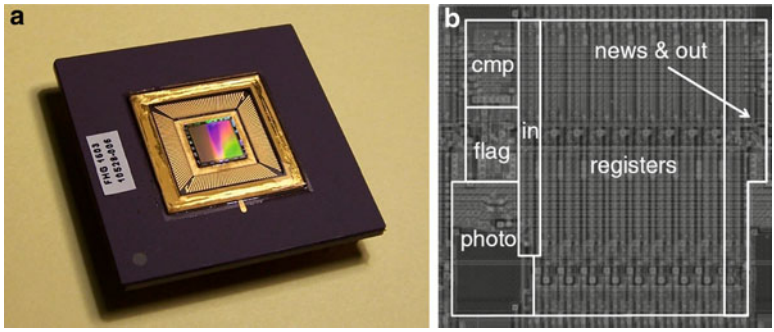


Fig. 18 SCAMP-3 device implementation: (**a**) fabricated integrated circuit, (**b**) PE layout

Table 2 SCAMP-3 chip specifications

Number of parallel processors:	16,384
Performance:	20 GOPS
Power consumption:	240 mW max
Supply voltage:	3.3 V and 2.5 V (analogue)
Image resolution:	128×128
Sensor technology:	Photodiode, active pixel, 0.35 µm CMOS
Pixel size:	49.35 µm × 49.35 µm (includes processor)
Pixel fill-factor:	5.6%
Imager fixed pattern noise:	1%
Accuracy of analogue processors (instruction error):	
Storage linearity:	0.52%
Storage error fixed pattern noise:	0.05% rms
Division-by-2 fixed pattern noise:	0.12% rms
Random noise:	0.52% rms
Image processing performance benchmarks (execution time @ 1 MHz clock):	
Sharpening filer 3×3 convolution:	17 µs
Sobel edge detection:	30 µs
3×3 median filter:	157 µs

5 Applications

The SCAMP-3 applications are developed using a PC-based simulator (Fig. 19a) that models the instruction set of the processor array and the control processor, including behavioural models of analogue errors. A development system, allowing operation and debugging of SCAMP-3 hardware in a PC-based environment (including user front-end Graphical User Interface, and a hardware system with a USB interface, as shown in Fig. 19b), has been developed. Ultimately, the target applications for the device are in embedded systems.

The general-purpose nature of the SIMD array allows implementation of a wide range of image processing algorithms. Some examples of filtering operations are

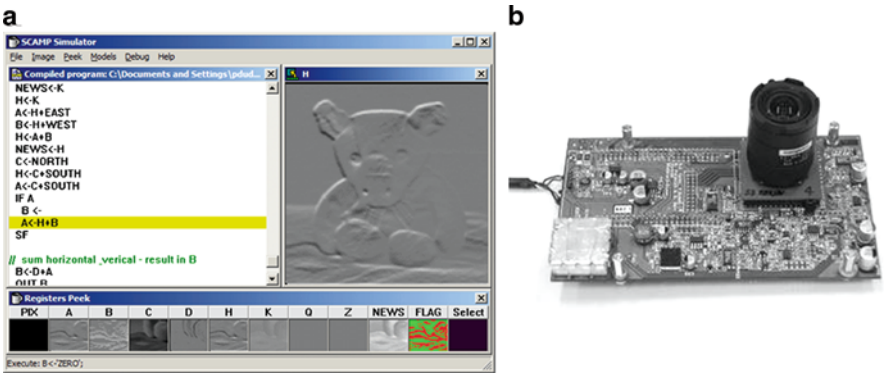


Fig. 19 SCAMP-3 development environment: (**a**) simulator software, (**b**) hardware development kit

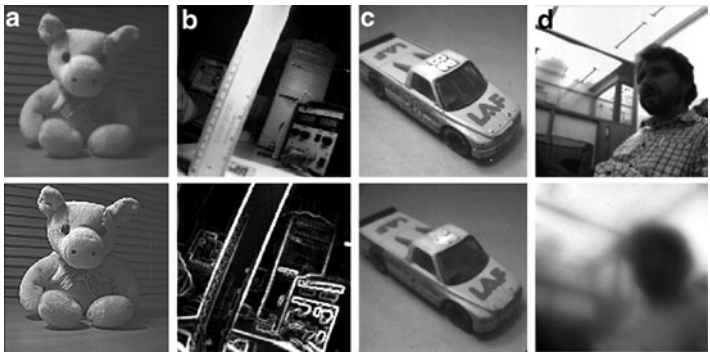


Fig. 20 Image processing algorithms executed on SCAMP-3. *Top*: original image, *Bottom*: processing result; (**a**) Sharpening filter, (**b**) Sobel edge detection, (**c**) Median filter, (**d**) Blur (diffusion)

shown in Fig. 20. Linear and non-linear filters presented in that figure require between 17 and 157 program instructions (with the exception of blur, which uses a resistive grid scheme and only requires two instructions), with maximum execution speed of 0.8 µs per instruction (higher instruction rates are possible, but with increased errors [33]).

More complex algorithms, for example adaptive thresholding, in-pixel A/D conversion and wide dynamic-range sensing [32], various cellular automata models [34], skeletonisation, object detection and counting, can also be executed at video frame rates. An example of image segmentation via active contours (using a Pixel Level Snakes algorithm [35]) is shown in Fig. 21. In this example, all processing is done on the vision chip. For illustration purposes, the frames shown are the gray-level images that are read-out from the device, where the results are superimposed on the input images on chip, although in practical applications, only the extracted information (i.e. binary contours in this case) would be read-out. We have considered

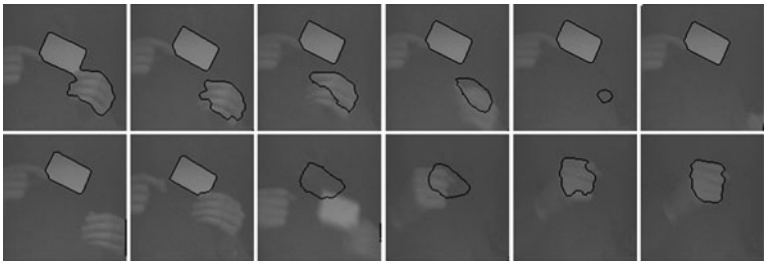


Fig. 21 Results of executing the Pixel level snakes algorithm on SCAMP-3. The contours evolve shifting at most 1 pixel at a time. Every 30th frame of the video sequence is shown

the application of the chip in several more complex tasks such as video compression [26], comparisons using wave distance metric [36], angiography image segmentation [37], binocular disparity [38] and neural network models [39].

6 Conclusions

This chapter presented the design of a SCAMP-3 pixel-parallel vision chip. The chip operates in SIMD mode, and the PEs are implemented using analogue current-mode scheme. This combination of digital architecture and analogue PE implementation provides flexibility, versatility and ease of programming, coupled with sufficient accuracy, high performance, low-power consumption and low cost of the implementation. The main applications of this technology are in power-sensitive applications such as surveillance and security, autonomous robots, toys, as well as other applications requiring relatively high computing performance in machine vision tasks, at low power consumption and low cost. The 128×128 prototype has been fabricated in 0.35 µm CMOS technology, and reliably executes a range of image processing algorithms. The designed processor-per-pixel core can be easily integrated with read-out/interfacing and control circuitry, and a microcontroller IP core, for a complete system on a chip solution for embedded applications.

Acknowledgement This work has been supported by the EPSRC; grant numbers: EP/D503213 and EP/D029759. The author thanks Dr Stephen Carey and Dr David Barr for their contributions to testing and system development for the SCAMP-3 device.

References

1. A. Moini, Vision chips, Kluwer, Boston, 2000
2. A.A. Stocker, Analog integrated 2-D optical flow sensor, Analog Integrated Circuits and Signal Processing, vol 46(2), pp 121–138, Springer, Heidelberg, February 2006

3. V. Gruev and R. Etienne-Cummings, Implementation of steerable spatiotemporal image filters on the focal plane, IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing, 49(4), 233–244, April 2002
4. R.P. Kleihorst et al., X et al: A low-power high-performance smart camera processor, IEEE International Symposium on Circuits and Systems, ISCAS 2001, May 2001
5. L. Lindgren et al., A multiresolution 100-GOPS 4-Gpixels/s programmable smart vision sensor for multisense imaging. IEEE Journal of Solid-State Circuits, 40(6), 1350–1359, 2005
6. S. Kyo and S. Okazaki, IMAPCAR: A 100 GOPS in-vehicle vision processor based on 128 Ring connected four-way VLIW processing element, Journal of Signal Processing Systems, doi:10.1007/s11265–008–0297–0, Springer, Heidelberg, November 2008
7. J.E. Eklund, C. Svensson, and A. Åström, VLSI implementation of a focal plane image processor – A realisation of the near-sensor image processing concept, IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol 4(3), pp 322–335, September 1996
8. F. Paillet, D. Mercier, and T.M. Bernard, Making the most of $15k\lambda^2$ silicon area for a digital retina, Proc. SPIE, vol 3410, Advanced Focal Plane Arrays and Electronic Cameras, AFPAEC'98, 1998
9. M. Ishikawa, K. Ogawa, T. Komuro, and I. Ishii, A CMOS vision chip with SIMD processing element array for 1ms image processing, Proc. International Solid State Circuits Conference, ISSCC'99, TP 12.2, 1999
10. J.C. Gealow and C.G. Sodini, A pixel-parallel image processor using logic pitch-matched to dynamic memory, IEEE Journal of Solid-State Circuits, 34(6), 831–839, June 1999
11. M. Wei et al., A programmable SIMD vision chip for real-time vision applications. IEEE Journal of Solid-State Circuits, 43(6), 1470–1479, 2008
12. A. Lopich and P. Dudek, ASPA: Focal plane digital processor array with asynchronous processing capabilities, IEEE International Symposium on Circuits and Systems, ISCAS 2008, pp 1592–1596, May 2008
13. A. Lopich and P. Dudek, An 80×80 general-purpose digital vision chip in $0.18\,\mu m$ CMOS technology, IEEE International Symposium on Circuits and Systems, ISCAS 2010, pp 4257–4260, May 2010
14. P. Dudek, A. Lopich, and V. Gruev, A pixel-parallel cellular processor array in a stacked three-layer 3D silicon-on-insulator technology, European Conference on Circuit Theory and Design, ECCTD 2009, pp 193–197, August 2009
15. A. Dupret, J.O. Klein, and A. Nshare, A DSP-like analogue processing unit for smart image sensors, International Journal of Circuit Theory and Applications, 30, 595–609, 2002
16. G. Liñán, S. Espejo, R. Domínguez-Castro, and A. Rodríguez-Vázquez, Architectural and basic circuit considerations for a flexible 128×128 mixed-signal SIMD vision chip, Analog Integrated Circuit and Signal Processing, vol 33, pp 179–190, 2002
17. M. Laiho, J. Poikonen, P. Virta, and A. Paasio, A 64×64 cell mixed-mode array processor prototyping system, Cellular Neural Networks and Their Applications, 2008. CNNA 2008, July 2008
18. J. von Neumann, A system of 29 states with a general transition rule, A.W. Burks (Ed.), Theory of Self-reproducing Automata, University of Illinois, IL, 1966
19. S.H. Unger, A computer oriented to spatial problems, Proc. IRE, vol 46, pp 1744–1750, 1958
20. G.H. Barnes, R.M. Brown, M. Kato, D.J. Kuck, D.L. Slotnick, R.A. Stokes, The ILLIAC IV computer, IEEE Transactions on Computers, 17(8), 746–757, August 1968
21. M.J.B. Duff, Review of the CLIP image processing system, Proc. National Computer Conference, pp 1055–1060, 1978
22. S.F. Reddaway, The DAP approach, Infotech State of the Art Report on Supercomputers, 2, 309–329, 1979
23. K.E. Batcher, Design of a massively parallel processor, IEEE Transactions on Computers, 29(9), 837–840, September 1980
24. D. Hillis, The connection machine, MIT, Cambridge, MA, 1985
25. P. Dudek, A flexible global readout architecture for an analogue SIMD vision chip, IEEE International Symposium on Circuits and Systems, ISCAS 2003, Bangkok, Thailand, vol III, pp 782–785, May 2003

26. D.R.W. Barr, S.J. Carey, A. Lopich, and P. Dudek, A control system for a cellular processor array, IEEE International Workshop on Cellular Neural Networks and their Applications, CNNA 2006, Istanbul, pp 176–181, August 2006
27. P. Dudek and P.J. Hicks, A CMOS general-purpose sampled-data analogue processing element, IEEE Transactions on Circuits and Systems – II: Analog and Digital Signal Processing, 47(5), 467–473, May 2000
28. P. Dudek and P.J. Hicks, A general-purpose processor-per-pixel analog SIMD vision chip, IEEE Transactions on Circuits and Systems – I, 52(1), 13–20, January 2005
29. C. Toumazou, J.B. Hughes, and N.C. Battersby (Eds.), Switched-currents: An analogue technique for digital technology, Peter Peregrinus, London, 1993
30. J.B. Hughes and K.W. Moulding, S^2I: A switched-current technique for high performance, Electronics Letters, 29(16), 1400–1401, August 1993
31. J.-S. Wang and C.-L. Wey, Accurate CMOS switched-current divider circuits, Proc. ISCAS'98, vol I, pp 53–56, May 1998
32. P. Dudek, Adaptive sensing and image processing with a general-purpose pixel-parallel sensor/processor array integrated circuit, International Workshop on Computer Architectures for Machine Perception and Sensing, CAMPS 2006, pp 18–23, September 2006
33. P. Dudek and S.J. Carey, A general-purpose 128×128 SIMD processor array with integrated image sensor, Electronics Letters, 42(12), 678–679, June 2006
34. M. Huelse, D.R.W. Barr, and P. Dudek, Cellular automata and non-static image processing for embodied robot systems on a massively parallel processor array, Automata-2008, Theory and Applications of Cellular Automata, pp 504–513, Luniver Press, 2008
35. P. Dudek and D.L. Vilarino, A cellular active contours algorithm based on region evolution, IEEE International Workshop on Cellular Neural Networks and their Applications, CNNA 2006, pp 269–274, Istanbul, August 2006
36. D. Hillier and P. Dudek, Implementing the grayscale wave metric on a cellular array processor chip, IEEE Workshop on Cellular Neural Networks and their Applications, CNNA 2008, pp 120–124, July 2008
37. C. Alonso-Montes, D.L. Vilariño, P. Dudek, and M.G. Penedo, Fast retinal vessel tree extraction: A pixel parallel approach, International Journal of Circuit Theory and Applications, 36(5–6), 641–651, July–September 2008
38. S. Mandal, B. Shi, and P. Dudek, Binocular disparity calculation on a massively-parallel analog vision processor, IEEE Workshop on Cellular Nanoscale Networks and Applications, CNNA 2010, Berkeley, pp 285–289, February 2010
39. D.R.W. Barr, P. Dudek, J. Chambers, and K. Gurney, Implementation of multi-layer leaky integrator networks on a cellular processor array, International Joint Conference on Neural Networks, IJCNN 2007, Orlando, FL, August 2007