TESI DI DOTTORATO

UNIVERSITA' DEGLI STUDI DI NAPOLI "FEDERICO II"

DIPARTIMENTO DI INGEGNERIA ELETTRICA
E DELLE TECNOLOGIE DELL'INFORMAZIONE

DOTTORATO DI RICERCA IN
INGEGNERIA ELETTRONICA E DELLE TELECOMUNICAZIONI

DIGITAL FPGA CIRCUITS DESIGN FOR REAL-TIME VIDEO PROCESSING WITH REFERENCE TO TWO APPLICATION SCENARIOS

GIORGIO LOPEZ

Il Coordinatore del Corso di Dottorato	Il Tutor
Ch.mo Prof. Niccolò RINALDI	Ch.mo Prof. Ettore NAPOLI

A. A. 2014–2015

“Poema loquens pictura, pictura tacitum poema”

– Rhetorica ad Herennium

Table of Contents

List of Figures

List of Tables

Preface

In the present days of digital revolution, image and/or video processing has become a ubiquitous task: from mobile devices to special environments, the need for a real-time approach is everyday more and more evident. Whatever the reason, either for user experience in recreational or internet-based applications or for safety related timeliness in hard-real-time scenarios, the exploration of technologies and techniques which allow for this requirement to be satisfied is a crucial point. General purpose CPU or GPU software implementations of these applications are quite simple and widespread, but commonly do not allow high performance because of the high layering that separates high level languages and libraries, which enforce complicated procedures and algorithms, from the base architecture of the CPUs that offers only limited and basic (although rapidly executed) arithmetic operations. The most practised approach nowadays is based on the use of Very-Large-Scale Integrated (VLSI) digital electronic circuits.

Field Programmable Gate Arrays (FPGAs) are integrated digital circuits designed to be configured after manufacturing, "on the field". They typically provide lower performance levels when compared to Application Specific Integrated Circuits (ASICs), but at a lower cost, especially when dealing with limited production volumes. Of course, on-the-field programmability itself (and re-programmability, in the vast majority of cases) is also a characteristic feature that makes FPGA more suitable for applications with changing specifications where an update of capabilities may be a desirable benefit. Moreover, the time needed to fulfill the design cycle for FPGA-based circuits (including of course testing and debug speed) is much reduced when compared to the design flow and time-to-market of ASICs.

In this thesis work, we will see (Chapter 1) some common problems and strategies involved with the use of FPGAs and FPGA-based systems for Real Time Image Processing and Real Time Video Processing (in the following also

indicated interchangeably with the acronym RTVP); we will then focus, in particular, on two applications.

Firstly, Chapter 2 will cover the implementation of a novel algorithm for Visual Search, known as CDVS, which has been recently standardised as part of the MPEG-7 standard. Visual search is an emerging field in mobile applications which is rapidly becoming ubiquitous. However, typically, algorithms for this kind of applications are connected with a high leverage on computational power and complex elaborations: as a consequence, implementation efficiency is a crucial point, and this generally results in the need for custom designed hardware.

Chapter 3 will cover the implementation of an algorithm for the compression of hyperspectral images which is bit-true compatible with the CCSDS-123.0 standard algorithm. Hyperspectral images are three dimensional matrices in which each 2D plane represents the image, as captured by the sensor, in a given spectral band: their size may range from several millions of pixels up to billions of pixels. Typical scenarios of use of hyperspectral images include airborne and satellite-borne remote sensing. As a consequence, major concerns are the limitedness of both processing power and communication links bandwidth: thus, a proper compression algorithm, as well as the efficiency of its implementation, is crucial.

In both cases we will first of all examine the scope of the work with reference to current state-of-the-art. We will then see the proposed implementations in their main characteristics and, to conclude, we will consider the primary experimental results.

Chapter 1

FPGAs and Real-Time Video Processing

In this Chapter, we will discuss general problems and benefits connected with the use of FPGAs in the design and development of Real Time Video Processing circuits and systems. We will consider common design issues and strategies, as well as typical architecture schemes: we will also focus on a special scenario, namely space applications, of which we will see an example in Chapter 3.

1.1 The role of Field Programmable Gate Arrays in Real-Time Video Processing

Real-Time Video Processing algorithms can be seen at two different abstraction levels, as shown in Figure 1.1. At a higher level, we have the "application-level algorithm": this algorithm determines the sequence of operations or steps that transform the input image into the desired output. At a lower level, on the other hand, each of these steps is described by an "operation-level algorithm": each of these sub-algorithms can be, itself, complex and articulated (see [1]).

From this characteristic, upon every image processing algorithm descends an inherent parallelism that is expressed mostly in the following two aspects:

- The decomposition of a RTVP algorithm into a sequence of image processing operations is a form of *temporal parallelism*. As a consequence, a separate processing unit may operate each transformation, in

a pipelined architecture, as shown in Figure 1.2a. If pipeline stages all have approximately the same latency, so that waiting times between adjacent stages are minimized, throughput can be greatly enhanced. Two difficulties can arise when using pipelining to implement image processing algorithms. The first is connected with multiple parallel paths: when, as an example, Processor 4 in Figure 1.2a takes as input also data from Processor 1, then proper synchronization logic must be set up. The complexity of this logic may range from a simple delay line to more complex structures when the intermediate stages have variable latency. A greater difficulty is connected with feedback handling, when data from subsequent stages affects parameters of earlier processors, for example in adaptive algorithms.

- In many RTVP operation-level algorithms there is a form of parallelism in terms of loops. A common example is when, typically in the outermost loop of each operation, all the pixel in an image are iterated for the same operation to be applied independently on them. This is a form of *spatial parallelism*, which may be exploited by partitioning the image and performing the operation, on each separate block, using a separate processor, like in Figure 1.2b. For video processing, the partitioning can

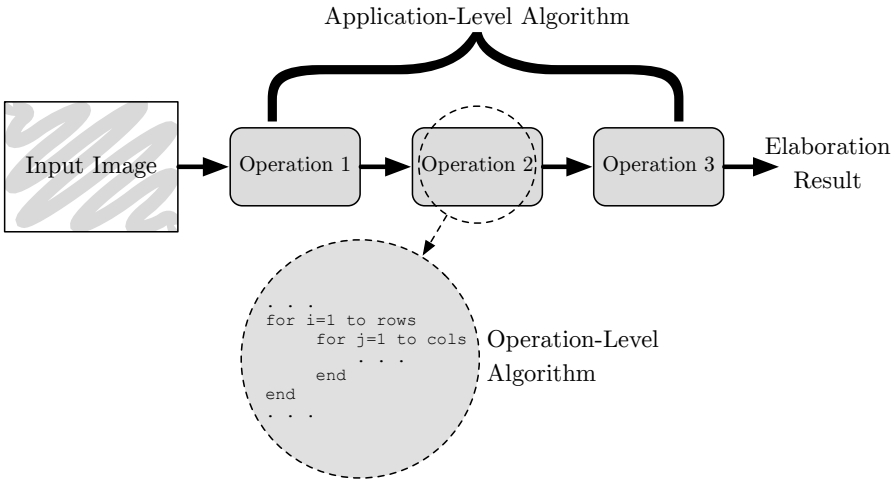


Figure 1.1: Application-Level Algorithms and Operation-Level Algorithms

be achieved also in time domain. The two main problems with such a parallelism are connected respectively with overhead in the image data distribution among the processors and with uneven workload with different partitions. With reference to the latter, this arises when processing time for each partition may vary significantly with dependence on the content of the image inside of that particular partition. In such a case, a

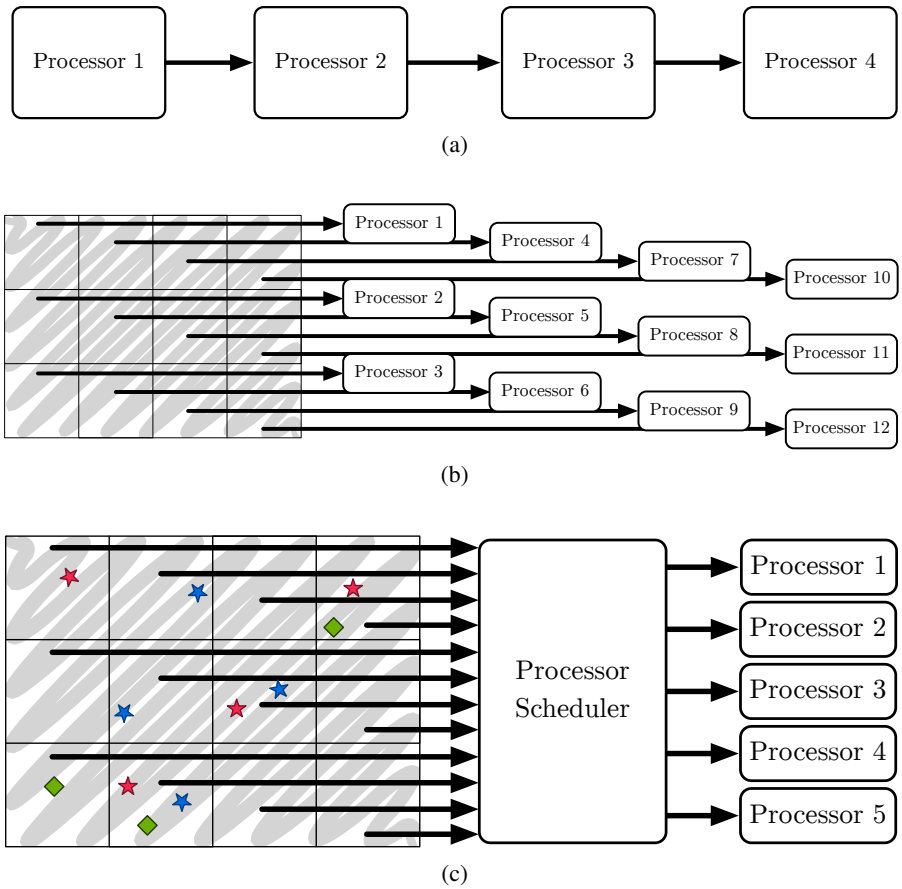


Figure 1.2: Structures for parallelism exploitation: (a) shows pipelining for exploitation of temporal parallelism, (b) shows partitioning for exploitation of spatial parallelism, (c) shows dynamic scheduled partitioning to optimize situation in which the workload is unevenly distributed in the image partitions

better throughput may be obtained with a dynamic scheduling of partitions to the processors, as Figure 1.2c shows.

As opposed to general purpose CPUs, which operate sequentially on the instructions that constitute an algorithm, FPGAs are inherently parallel. This makes them at the same time fast because of the nature of an hardware design and suitable to exploit the parallelism mechanisms that have now been exposed. For a pipelined architecture, specific hardware is built for each operation-level algorithm: if operations are synchronous, data is simply passed between stages. In cases in which stages do not operate synchronously, appropriate buffers may be needed. Partitioning is also easily implemented on FPGAs by instantiating multiple copies of the processing hardware and dispatching, either statically or dynamically, image partitions to each copy.

Real Time Video Processing on FPGAs, as a consequence of these argumentations, is a very discussed field in the study of RTVP systems. Particularly notable examples in literature are [2, 3, 4, 5, 6].

1.2 FPGAs for RTVP: Typical System Architectures

In the process of developing FPGA systems for RTVP, a smart coupling of the algorithm and the implementation is vital for taking full advantage of the capabilities of hardware: as [7] states, the development of complex, high performance algorithms on FPGAs is "unusually sensitive to the implementation's quality". It is not sufficient to port an algorithm from software onto an FPGA. The algorithm and the computational architecture must be well matched. Of course there is not a unique solution, as development, with particular reference to algorithm development, usually follows heuristic principles ([8]).

The architecture of an FPGA-based RTVP system can be considered at two levels. At a higher hierarchical level, system level architecture determines the interaction between the main components of the system: at a lower level, computational architecture determines the means by which the computation of the algorithm is performed.

With software-based image processing, the computational architecture is fixed (the datapath of the CPU), and the system architecture is generally predefined (classical models). On FPGA platforms, conversely, the whole architecture must be designed and developed. This is of course a burden, but gives the designer the possibility to strategically configure an architecture which exploits the algorithm intrinsic parallelism features to the maximum.

Taking as a starting point the taxonomies defined by [9, 10] we can identify the following examples of system level architecture, which are also depicted in Figure 1.3

- Standalone FPGA: the FPGA operates without the aid of a CPU: an external (DRAM) memory may or may not be present (Figure 1.3a). This is most indicated when the algorithm is data-intensive and of relatively low control complexity.
- Multiprocessor arrangement: in this scheme, as shown in Figure 1.3b, a CPU is present, and the FPGA behaves as an additional processor in

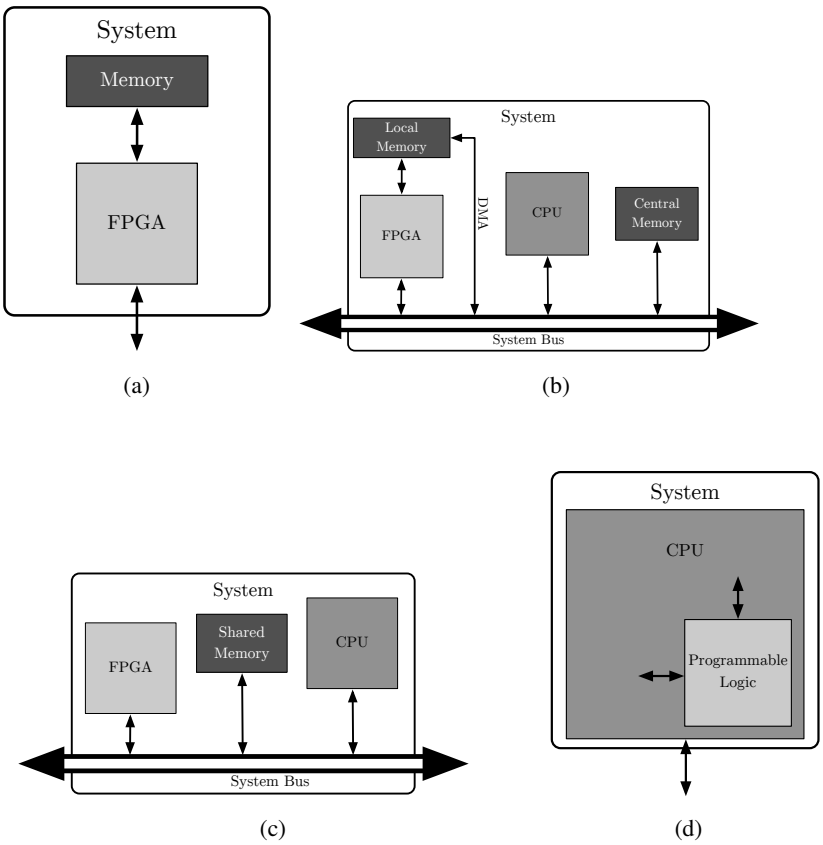


Figure 1.3: Typical System Level Architectures: (a) Standalone FPGA; (b) Multiprocessor Arrangement; (c) Coprocessor Scheme; (d) Integrated Scheme

a multiprocessor system. The FPGA resides on the system bus and receives input data by the CPU via DMA on a local memory, transferring back the results either through DMA or through mapped registers, depending on the requested processing. In this way, the control flow can be managed in software by the CPU while computationally dense portions of the algorithm that benefit from FPGA implementation can be accelerated via hardware implementation. Of course, the CPU may be part of the FPGA itself, either as a hard core CPU (like the ones implemented in SoC-oriented FPGAs, [11, 12, 13]) or as soft-core IP blocks implementing standard or custom processors ([14, 15, 16, 17]).

- Coprocessor scheme: a more tightly coupled variation of the previous scheme, in which the FPGA has access to a memory that is shared with the CPU. The hardware computation is, also in this case, triggered by the CPU after the data has been copied in the shared memory, and proceeds independently from the CPU instruction flow, returning output data with similar mechanisms to those relative to a multiprocessor system. This scheme is depicted in Figure 1.3c.

- Integrated scheme: this is the most tightly coupled scheme, as the programmable logic is integrated inside the processor, often in terms of custom instructions. Input and output data, in this scheme, is usually passed through shared registers: as a consequence a limitation is the restriction in the amount of data that may be passed with each instruction. Furthermore, during the processing the main processor is usually stalled, waiting for the custom instruction to complete, and therefore generating also a practical limitation in the time that is available to complete the processing. Figure 1.3d schematises this arrangement.

1.3 From Algorithm to Hardware: Computational Level Architectures

As we have seen before, at a lower hierarchical level, architectural choices define how an operation-level algorithm is mapped to an hardware implementation. Different algorithm structures may need different computational architectures, with the main schemes being summed up as follows:

- Stream Processing: in an algorithm that is of mostly serial nature, in

which different image transformation are applied sequentially, the main bottleneck of software implementations is that it is memory bound. The CPU needs to read the pixel values from the memory, process them, and then store the processed pixel value in memory again. In a stream processing scheme, this bottleneck can be avoided by pipelining operations, therefore exploiting temporal parallelism. Input is fed from the camera sensor (or read from memory, or received from a communication channel) to the first stage of the pipeline. Results from each stage are immediately passed to the next one. As much processing as possible should be performed before writing to memory the data even once. Ideally, all the application level algorithm can be performed without using any buffer to store the image data.

This architectural scheme suits particularly well RTVP algorithms that can be executed completely during a raster scan, like point operations (operations that require only input data from a single pixel), or local filters. If an operation requires data from more than one pixel, local buffers must be added.

The major constraint of stream processing is that the throughput is fixed and bound either to the input (e.g: the acquisition speed of the camera sensor) or to the output (e.g: the display frame rate). As a consequence, stream processing gives place to synchronous systems.

- Array Processors: in this architectural scheme spatial parallelism is exploited by allocating multiple "Processing Elements" (PEs) that are coordinated by a common control unit. The control unit performs scalar operations autonomously, while dispatching vector or matrix operations to the PEs. An important feature of array processors is the interconnection scheme that allows communications between the different PEs and between PEs and the shared memory, if any. Typically, this scheme is either in mesh or hypercube form when memory constitutes a single, distributed block, or in a crossbar structures when memory is shared between PEs.

- Systolic Arrays: a particular kind of array processors that are composed by a set of identical nodes, each including a dedicated memory. The nodes are interconnected into basic structures (trees, meshes, etc.) that match the computational graph in order to maintain inter-node communications to the minimum. Data travels from the central unit memory to

the processing nodes to return again to the central memory (in analogy with blood circulation, hence the name). This can be considered a combination of array processing and stream processing, because data flows between adjacent blocks (like in pipelined architectures) but, while in stream processing the connected blocks realize different operations, in systolic arrays all nodes are replicas of the same processing element.

- Buffered Processing: in some cases, pixels must be accessed from everywhere in the memory during the algorithm computation as there are no constraints in the pattern of data access (the scheme is also referred to as "random access processing"). As a consequence, the whole image must be available in a frame buffer, which will likely reside on external memory, except in the case in which image size is very small. In a sense, buffered processing is most similar to software processing, and it is easier to map a software algorithm onto such an architecture. Simply "porting" an algorithm to hardware with buffered processing will give limited advantage over a software implementation because of memory bandwidth limitations. Where possible, local buffers can substitute a frame buffer to improve external memory access. Another consequence of the adoption of such a scheme is that the hard timing constraints of stream processing are relaxed: the output may not be regular, and there may be a variable number of clock cycles to produce each pixel in the output. This will, however, generate difficulties in the synchronization between algorithm steps.

Of course, hybrid solutions exist in between these schemes: ideally, the best performance is usually achieved when the whole algorithm is implemented during a single raster scan, and the use of buffering is minimized. In some applications, on the other hand, this is not possible and, while the use of frame buffers can be limited to a minimum, an hybrid approach is the only pursuable solution.

In other cases, the algorithm can be remodelled by substituting one or more processing steps with other equivalent or similar operations: in some cases, substituting an operation may slightly change the obtained results, and some functionality verification with respect to the original algorithms must be performed.

1.3.1 Design Challenges: Control and Synchronization

One of the most challenging aspects of designing circuits for RTVP is connected to algorithm control flow and and data stream management. Either with streaming processing, which usually requires small sliding window buffers, or with larger buffers that must be properly managed and accessed, synchronization and control logic are crucial points.

An example is related to synchronization between asynchronous algorithm steps. When different processors have variable processing times, intermediate buffers are necessary to arrange the situations in which either the downstream unit is not able to accept incoming data or the upstream unit is not ready to produce output data. In some cases FIFOs (or dual clock FIFOs, when the two adjacent units operate at different clock frequencies) may be sufficient, but sometimes entire frame buffers are needed. Figure 1.4 shows a typical synchronization scheme, in which a FIFO provides basic handshaking logic.

Another problem arises when spatial parallelism is exploited in an image processing algorithm: different parts of the image are allocated to separate processors. In this case, however, it is also fundamental to consider the overhead generated by the transfer of data between central memory (e.g: a frame buffer) to local processing memory and vice versa. In other cases, some strategies of bank switching, like Ping-Pong buffering, may be needed to avoid idle times in the schedule of the processing elements.

More generally, one of the major issues in the process of mapping algorithms to embedded RTVP processing is meeting the various constraints. A common trade-off occurs between speed and accuracy ([18]), and must be resolved in the observance of three main typologies of constraint: timing con-

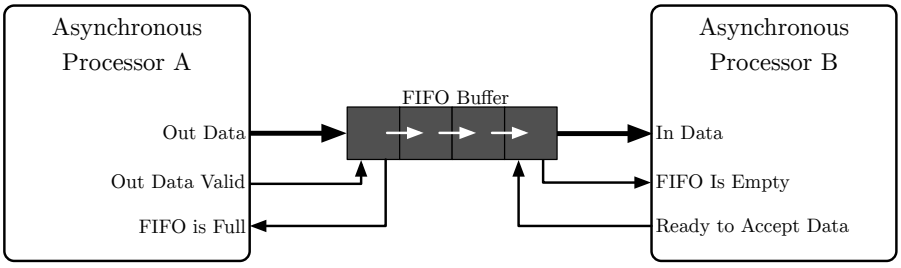


Figure 1.4: A FIFO buffer can be used to provide handshaking logic between asynchronous processing blocks

straints, memory bandwidth constraints and resource constraints.

Two types of timing constraints are typical of every real-time application: throughput and latency. Throughput is typical of stream processing architectures and is often imposed by input or output considerations. Latency is important for real-time control applications where the output from an image processing algorithm is used to provide feedback. High latency, in these cases, may lead to unstable systems.

Memory bandwidth constraints are particularly relevant when dealing with external memory. While on-chip memory, in fact, can be less of a problem because it can be organized in a larger number of small blocks, external memory tends to be monolithic, and often pipelined. In some cases this leads to the need to transfer data from off-chip memory into multiple blocks of on-chip memory, where the constraints are more relaxed.

Resource constraints consist in two interrelated aspects: firstly, in a commercial application, to minimize production cost it is desirable to use the smallest fitting device. Therefore, is vital to make an efficient use of the available resources. On the other hand, a more circumscribed resource usage can lead to better timing scores: a smaller design can, in fact, result in an easier routing across the device, with resulting shorter routing delays. Conversely, when the size of the design approaches the resource availability of the FPGA, the design becomes harder to route efficiently, and routing delays may cause the clock frequency to decrease.

1.4 Design Tuning and Testing: Techniques and Strategies

Another challenge in the design and development of digital circuits for RTVP is connected with tuning and testing phases.

Most image processing algorithms have parameters (e.g: threshold levels, filtering parameters, etc.) that require optimization to achieve the best performance. When these parameters are hard wired into the implementation, at each parameter change the whole system needs to be re-implemented and re-mapped to hardware. As a consequence, since a completely run time configurable implementations lead to complicated and less optimized hardware definitions, it is necessary to have a proper trade off between run-time configurable and implementation-time configurable parameters.

In each case, though, performing the elaboration on whole images is eventually the only way to verify the effect of the parameters set on the algorithm implementation. The same principle applies to algorithm verification or testing, with the difference that while tuning usually requires an evaluation of the test case output on its own, testing typically requires some comparison with a reference or "golden" implementation.

However, generally speaking, it is not possible to perform an exhaustive test of the complete algorithms: in RTVP applications, every image that is processed will produce different behaviours in the algorithm and in its control flow. As a consequence, failures may not be the result of a wrong implementation, but also the effect of a not completely robust implementation. This problem is of course common to most design testing, and not only to image processing applications, but some specific problems and strategies apply to RTVP designs.

With reference to testing, the most common way to verify the proper functioning of an algorithm implementation is a comparison with a "golden implementation": this may be, for example, a reference implementation released by a standardisation committee ([19]), or simply a pre-existent hardware or software implementation. We could either have or not access to an internal view of the reference implementation (e.g: source code): in the first case we can speak of a "white box" reference, which can be of great help in testing of debugging single sub-parts of complex algorithms, while in the latter we can speak of a "black box" reference. In both cases we can choose between two strategies:

- Random selection of input test vectors: this is of particular use when debugging subsections of a complex algorithm, since in such a situation the space of possible input vectors reduces and it is easier to achieve a more representative sample of the input, even on a randomly generated test vectors set. Of course the approach can be extended to the test of the whole algorithm by choosing a set of random (real world or synthetic) images. In this case, proper care must be taken in choosing an adequate set.

- Standard dataset verification: this is typically connected with the testing of standard algorithms. In the testing framework an image set is defined for both specification adherence verification and performance comparison. These datasets are built to be well representative of the final use context of the system not only in the choice of images, but also in terms

of use conditions (i.e. noise, light conditions, etc.): as a consequence they tend to be very robust and, at the same time, very large.

In all cases, the verification on large test vector sets can be a very time consuming process, and more than one strategy is pursuable:

- Software verification: in most cases, mapping an algorithm to hardware requires changes in the actual implementation. Such change may be limited to a different control flow, a change in data representation (e.g: fixed point as opposed to floating point), or a substitution of some operations to better suit hardware capabilities and limitations. In these cases, a first verification can (and should) be performed in software before mapping. Software verification is, in fact, much easier both for the abundance of tools (MATLAB, step debuggers, etc.) and for the time required to perform changes in the implementation.

- Simulation: a further step from software verification. Taking for granted the availability of simulation models (included in the development kits released by FPGA manufacturers), simulation allows verification of the actual hardware behaviour before deploying it on the device and with the ability to inspect internal elaboration nodes very easily. On the other hand, since simulation consists in a serial processor (the host CPU) simulating concurrent hardware (the deployed circuit), the process can be quite time consuming, especially for large designs and for timing-accurate simulation. As a consequence, testing against large test image sets may be prohibitive.

- Co-simulation: this is an intermediate solution between the previous two, in which a part of the algorithm is simulated and a part is executed in software. This approach can be used to verify with greater reliability critical portions of the algorithm, while speeding up the total process by having the more straightforward parts executed in software, therefore reducing workload on the simulator. A scheme of this approach is shown in Figure 1.5a

- Hardware debug: in this scheme, the system is completely deployed on the FPGA. This gives the maximum throughput, provided that input and output data are fed to the device and retrieved from it without significant overhead. However, this environment presents two major disadvantages:

firstly, visibility of internal nodes can only be obtained by modifying the interface of the Device Under Test (DUT): in this case, the simplest approach is to route the desired internal signals to unused I/O ports, making them observable from outside the FPGA. A logical analyser or an oscilloscope can then be used to monitor the signals. In some other cases, if the resource occupation on the device is not too high, a logic analyser can be embedded within the FPGA, alongside with the DUT. However, each modification to the hardware or, beyond some extent, to the logic analyser configuration requires a new synthesis and routing of the device which may be very time consuming. Another issue related to modifying the circuit to include signal probes and/or logic analysers is that these alterations modify how the connections are routed on the FPGA. These can change timing characteristics of the design, potentially masking or generating timing problems ([20, 21]). This approach is depicted in Figure 1.5b

- Hardware in the Loop Verification: this approach, depicted in Figure 1.5c is quite similar to co-simulation, except that the hardware deployed part of the algorithm is executed on the device. Data is transferred from software to the hardware and results are transferred back to the software host. A "loop" is thus obtained, in which the FPGA is used to accelerate processing with respect to simulation, but with a lesser degree of inspectability, with similar problems and solutions as in hardware debug. In general terms, the DUT will be encapsulated in a wrapper which connects it to the HIL framework. Today, several tools for Hardware in the Loop verification exist, like MatLAB HDL Verifier ([22]).

1.5 An Example of Special Environments: FPGAs in Space Applications

A particular yet important environment in which FPGAs excel, especially when compared to other technologies, is related to space applications. The issues connected with the high levels of ionizing radiation pose a severe challenge to circuit designers: in space, electronic systems must endure extreme levels of radiation without risks to their reliability. While design techniques can mitigate the probability of failures, radiation hardened (Rad-Hard) compo-

nents offer a much higher level of dependability. Nowadays, rad-hard FPGAs offer capabilites which overcome the state-of-the-art of rad-hard ASICs, grant-

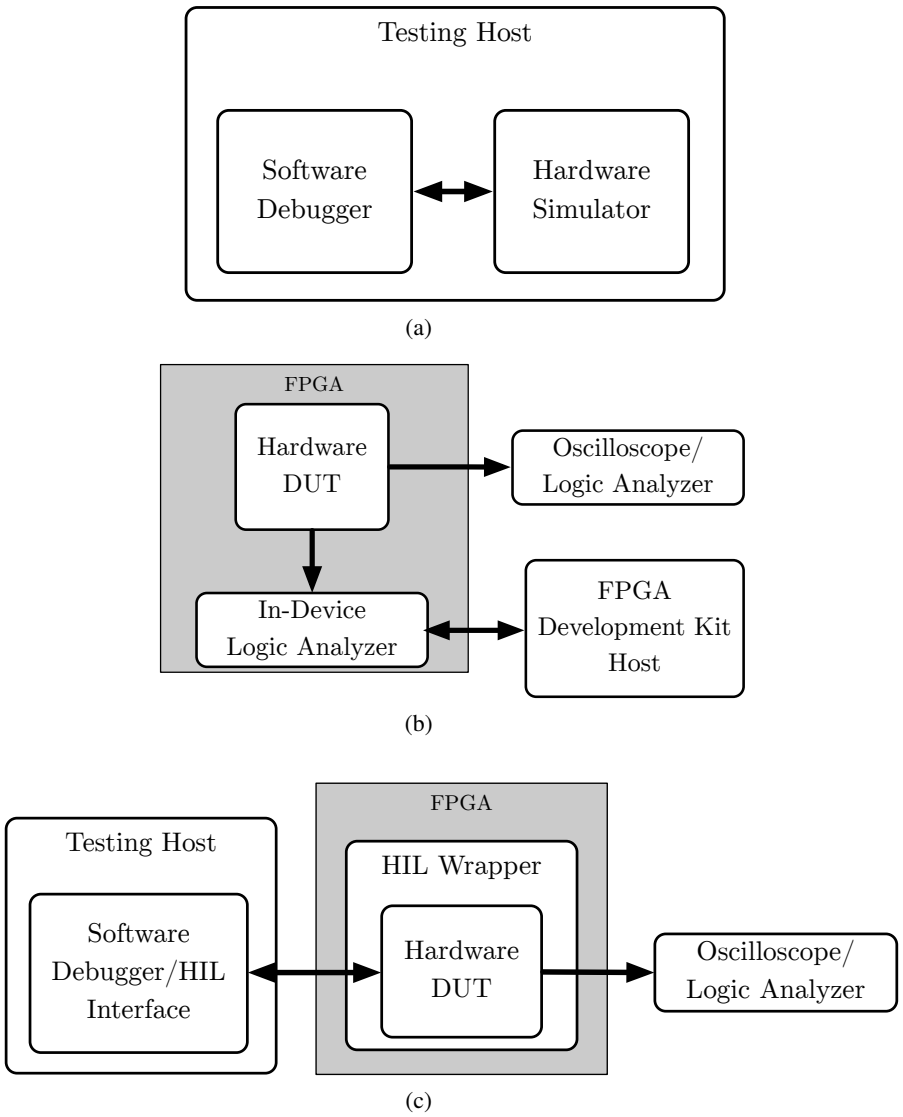


Figure 1.5: Some testing architectures: (a) Co-simulation; (b) Hardware Debug; (c) Hardware In the Loop Verification

ing much lower development costs and risks. As an example, NASA Mars rover missions Spirit and Opportunity used this kind of devices for the implementation of critical functions like landing and wheel movement control ([23]).

Rad-Hard FPGAs are divided into two categories: anti-fuse-based, One-Time-Programmable (OTP) devices and SRAM-based, reprogrammable devices. Anti fuse devices offer a slightly higher level of reliability, thanks to the intrinsically lower amount of programmable elements that can be upset by the ionizing radiation and to a pervasive use of redundant circuitry. The process for realization of SRAM-based devices offers an advantage of multiple process generations with respect to anti-fuse device process (and to Rad-Hard CMOS process for the realization of ASICs), offering greater capacity, better performance, and lower power consumption per gate. On the other hand, SRAM-based devices need to be configured at power-on, and radiation-tolerance can result in the requirement for error mitigation techniques such as ECCs: this error mitigation, though, can be applied selectively, optimizing the usage of resources. Of course, reprogrammability and boot-time configuration are an enormous source of flexibility: in a satellite system, bugs can be fixed or requirement changes can be accommodated even after launch.

The application example discussed in Chapter 3 is intended for air-borne and satellite-borne image sensing and makes use of a Xilinx Virtex 5QV Rad-Hard SRAM-based FPGA. In these devices, the standard 6-transistors configuration memory cell is replaced with a 12-transistors cell, which is about three orders of magnitude more resilient to changes in state than commercial SRAM cells, to mitigate the criticalities of boot-time configuration.

Chapter 2

An FPGA Based Architecture for the Implementation of the CDVS Standard for Visual Search

This chapter will cover the development of a standard algorithm for Visual Search known as CDVS (Compact Descriptors for Visual Search). Computer Vision is a more and more pervasive technology in modern image and video processing applications: examples include image driven search, stereoscopic image matching, panorama stitching and industrial automation tasks such as automatic control or robot navigation.

CDVS has been recently proposed as part of the MPEG-7 standard: it is based on a very well known algorithm in Computer Vision, namely SIFT (Scale Invariant Feature Transform). SIFT has been firstly formulated by D.G.Lowe in [24, 25] and, since then, it has been much discussed and appreciated by the scientific community thanks to its characteristics. Various other algorithms are based on variations of its rationale and elaboration pipeline ([26, 27, 28, 29]), but SIFT remains the most considered, to date. It has also been implemented in many forms and variants in literature ([30, 31, 32]): however, no implementation of the released CDVS standard (which differs from SIFT in more than one aspect) has to date been proposed.

This thesis work is part of a large-scale effort to produce a series of processors capable of implementing, with a full hardware approach, a real-time

implementation (with a throughput around 24 frames per second) the CDVS pipeline. The project has been carried out, in co-operation with ST Microelectronics, member of the MPEG committee, and with University of Salerno. The joint project gave the opportunity to divide the complex elaboration pipeline into sub-parts and apply a multitude of approaches and expertises to the same problems. We will now see an FPGA implementation of the majority of the CDVS algorithm steps that operates on the basic philosophy, in particular for what concerns the filtering operations, of block-wise, frequency domain operation.

2.1 MPEG 7: "the bits about the bits"

MPEG 7 is a standard for content description of multimedia formats, so it does not deal with the actual encoding of data streams like other MPEG standards (MPEG-2, MPEG-4, etc.): for this reason, its application scope is often loosely referred to as "the bits about the bits". The draft was standardized as ISO/IEC document number 15938, which is composed of several parts: part 13 of the document regards Compact Descriptors for Visual Search.

Traditionally, visual search algorithms require query images to be sent from the client device (typically a mobile device) to a remote server, where the actual visual search is performed over a reference image database. This implies a certain latency to transmit even a downsampled version of the image over a slow link. An alternative, to reduce latency, is based on extraction and compression of the query image (translated into a set of "features" that constitute a descriptor) directly on the client device: in this fashion only the descriptor is transmitted, with an improvement on performance, user experience, and power consumption, since less data is to be transmitted over wireless links. In addition, this approach has a positive impact on privacy concerns, since the features are anonymous, while pictures taken by a mobile device may contain user sensitive data.

2.1.1 CDVS standardisation effort and TMuC

CDVS standardisation has been conducted between the 92nd and 108th MPEG committee meetings: in particular, up to the 96th meeting, a group of experts has widely investigated applications and requirements of visual search. During the 97th meeting, an official call for proposals was formulated; at the 106th

meeting, CDVS entered the status of committee draft. Finally, during the 108th meeting, it entered the draft of international standards. In the document, the format of compact visual descriptors as well as the feature extraction process pipeline is addressed, in order to provide a common ground for interoperating applications. An evaluation framework (also referred to as Test Model under Consideration - TMuC or simply TM) has been constructed, encompassing two types of experiments: retrieval and pairwise matching. Retrieval experi-

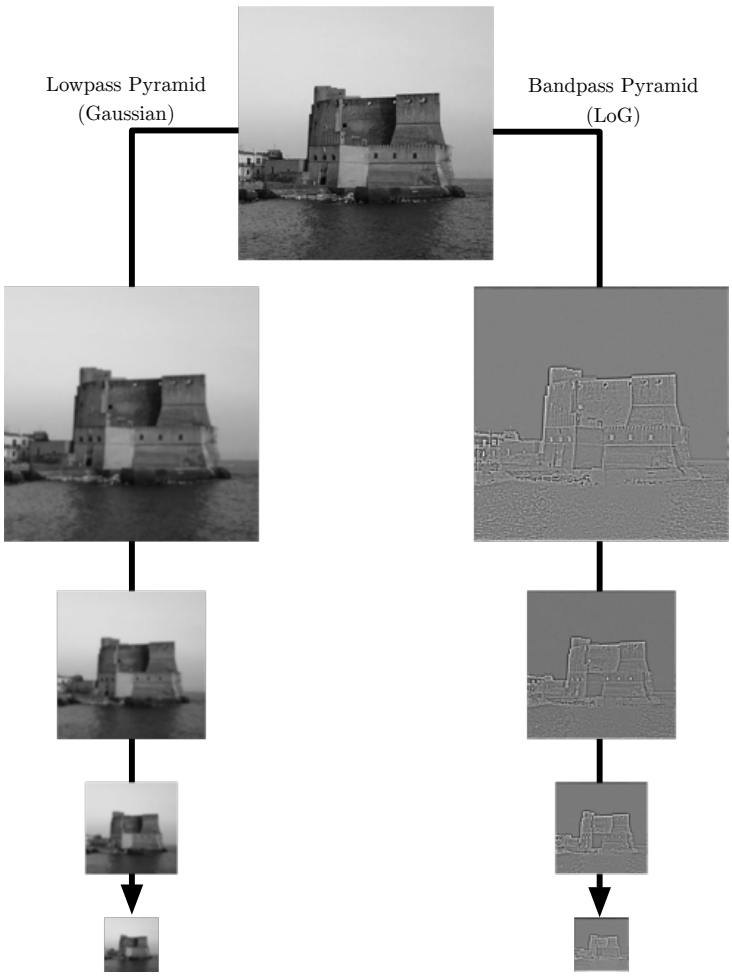


Figure 2.1: Construction of the band-pass and low-pass pyramids.

ments consist in the research, in a large dataset, of images containing the same objects or scene as the query, while pairwise matching is the direct confrontation of a query image with a reference image to determine whether they depict the same object/scene or not. Experiments are validated on an image dataset accompanied by a set of experiments which are used to evaluate the performance of the descriptors and of their implementation. This dataset, which is in turn divided into subsets, is composed of a large number of query and reference images. As an example, the largest subset contains 3,499 queries and 11,677 reference images of buildings. In addition, 3,805 matching pairs and 48,675 non matching pairs are defined. Being one of the most important areas of investigation of CDVS related to addressing the practical issue of descriptor rate scalability, the CDVS descriptors adapt to six operating points, each corresponding to a specific "bit budget". The proper configuration is then chosen on the basis of the specific use conditions.

2.2 The CDVS Algorithm

The CDVS Algorithm is based on scale-space theory, as formulated by Tony Lindeberg in [33]: the theory states that, being objects only meaningful at their characteristic size, when performing an automated visual search an image must be analysed at several different scales. This results in the elaboration of the concept of image pyramid. An image pyramid is a set of images obtained, by means of filterings and subsamplings, from a single original image. In CDVS, as Figure 2.1 shows, two parallel image pyramids are constructed: a low-pass pyramid, constituted by the results of filtering the original image with Gaussian kernels with different standard deviations σ, and a band-pass pyramid, constituted by filtering the same image with Laplacian of Gaussian (LoG) kernels. The first is used for describing the image at the various scales, while the latter is used to determine points of interests in the image.

The elaboration pipeline of the CDVS algorithm (or, in other terms, its application level algorithm) constitutes a very complex sequence of elaborations: for the sake of clarity we will divide it into five macro steps:

- Interest Point Detection: in this phase, the image is decomposed into the two pyramids and keypoints are individuated. Successively, orientations are assigned to the keypoints and the local descriptor for each keypoint is evaluated.

- Feature Selection: the compactness of the generated descriptor depends on the number of local features. In addition, a reason to perform a feature selection is that, removing less significant (e.g: noisy) features, the overall quality and discriminative power of the descriptor may improve.
- Local Descriptor Compression: another aspect that impacts the size of the descriptor is, of course, compression of the single local descriptors. For this purpose, CDVS encompasses a quantization scheme with a variable decimation that is dynamically adapted to the desired output bitrate. This step and the previous are jointly tuned to provide a good balance between the number of elements per descriptor and the number of descriptors that can be packed at a given descriptor length.
- Coordinate Coding: keeping track of the coordinates of keypoints is very important for geometric matching of objects and scenes, but may result in a significant memory occupation. Given a VGA (640x480) image, 19 bits are needed to encode the location of a keypoint without compression. If an image contains 500 keypoints, 1.16 KBs are needed simply to encode location of the keypoints: this information alone exceeds the lowest operating point (512 bytes) allowed by CDVS. As a consequence, to indicate keypoint coordinates, the image is divided into a grid and a histogram is built. Arithmetic coding is then employed to encode histogram values and the histogram map.
- Global Descriptor Aggregation: as a final step, a subset of the local descriptors is aggregated into a global descriptor with a technique known as Scalable Compressed Fischer Vector (SCFV) which, compared to other approaches in literature, combines high discriminative power with an unequalled scalability.

The circuits described in this chapter realize functions which relate to the first macro-step: as a consequence, we will examine this in major detail in the next sections.

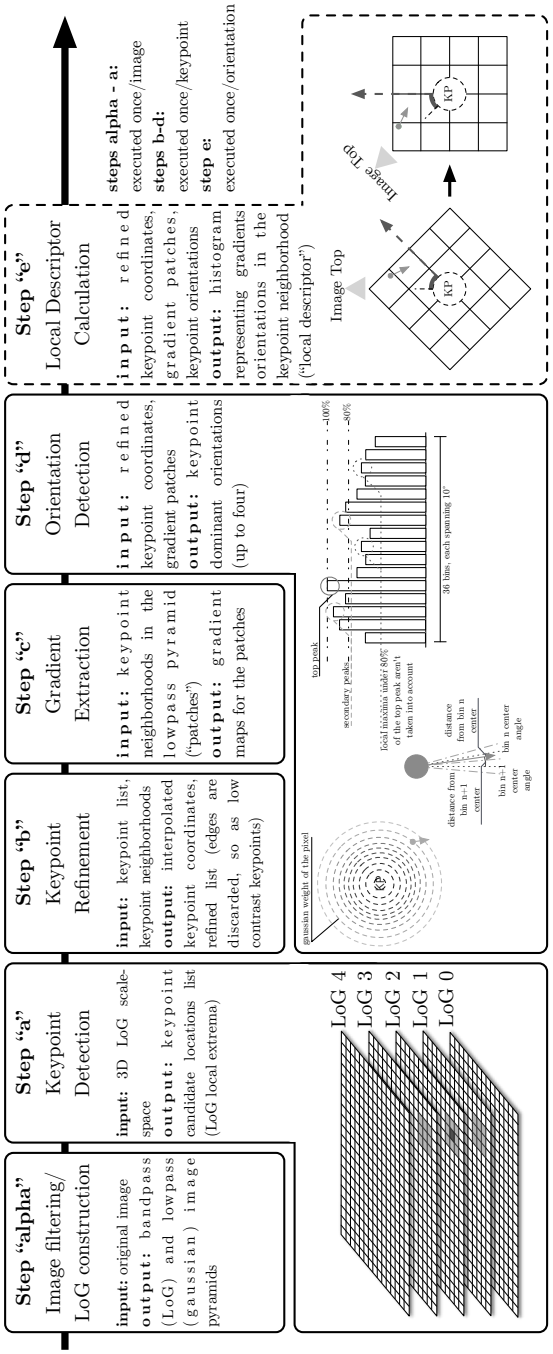


Figure 2.2: Elaboration pipeline of the CDVS Interest Point Detector.

2.2.1 Interest Point Detection Insights

Details of the application-level algorithm for the interest point detection phase of the CDVS pipeline are shown in Figure 2.2: in particular, a sequence of six steps is individuated.

The first step (step "alpha", in the figure) is the construction of the image pyramid. For what concerns this phase, the CDVS algorithm is characterized by an important difference with respect to SIFT: while the latter utilizes, as a detection filter, the Difference of Gaussians (DoG) kernels, which are an approximation of the Laplacian of Gaussians, CDVS utilizes the LoG themselves. As an additional difference, while SIFT makes use of space domain convolution (also because, as we will see, DoG filtering can be achieved by means of separable filters), CDVS involves a block partitioning and a frequency domain filtering of the original image. These characteristics, which result in the architecture known as "FBLoG" (Frequency Block-based LoG), are based on a proposal designed and developed in collaboration with ST Microelectronics during as part of this Ph.D thesis work and presented in the US patent [34]. During this algorithm step, in addition, the low-pass Gaussian pyramid is produced.

The step "a" shown in Figure 2.2 is the keypoint individuation phase. In this phase, as shown in Figure 2.3, the 3D space of the LoGs is scanned for local extrema. A local extremum, in this context, is a pixel in the LoG that has value either greater or lesser than all of its 26 neighbours (8 neighbouring pixels in the same LoG, 9 in the "above" LoG and 9 in the "below" LoG). All

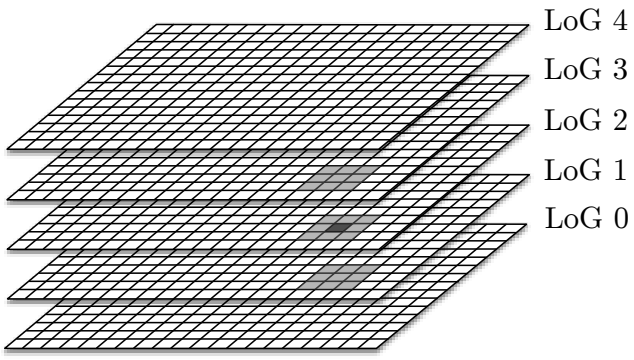


Figure 2.3: 3D matrix of the obtained LoG images.

Figure 2.4: Effects of keypoint refinement on a sample image: candidate keypoints before refinement.

of these pixels are marked as candidate keypoint locations and inserted into a queue for elaboration by the downstream steps.

Step “b” performs a refinement of the keypoints: this has the dual purpose of rejecting both low-contrast and poorly localized keypoints, which may be respectively caused by noise in the filtered images or by edge response. With particular reference to this latter case, edge response keypoints are prone to exhibit instability and, thus, give place to descriptors which have low reproducibility and reliability. The effect of this phase on a sample image is shown in Figure 2.4 and Figure 2.5 .

Step “c” consists in the extraction of luminosity gradients from the Gaussian filtered images produced in step alpha: on the basis of the gradients which lie in the proximity of the detected keypoints, in fact, the algorithm will build the final local descriptors.

Step “d” is the dominant orientations detection phase: this is needed to

Figure 2.5: Effects of keypoint refinement on a sample image: refined keypoints.

"normalize" the descriptors in terms of the given orientation in which the keypoints appear in the image and, thus, give robustness to the descriptor itself whenever the image (or the object in the image) is rotated. To accomplish to this, the gradients extracted in step "c" are weighted with their distance from the keypoint location and accumulated into a histogram accordingly with their phase. Up to four "local extrema" bins (reaching the 80% of the absolute peak) are taken as dominant orientations. Figure 2.6 summarizes the operation-level algorithm relative to this step.

Step "e" is the final step connected with the computation of local descriptors, and it represents the calculation of the descriptor itself. The descriptor represents a histogram obtained by accumulating gradient orientations in a way similar to the one discussed for step "d", but with some differences. Firstly, gradient phases are normalized with respect to the dominant orientation (when more than one orientation is detected for the keypoint, a different descriptor is

calculated for each of them). Secondly, gradients are spatially binned accordingly to their coordinates or, to be more precise, accordingly to their offset with respect to the keypoint location. This spatial binning divides the area around the keypoint in a grid of $4 \times 4 = 16$ bins and, again, is normalized to the orientation of the keypoint. As a last difference, while in step d the orientations histogram was composed by 36 bins each spanning $360°/36 = 10°$, the local descriptor is composed by 8 orientation bins per spatial bin. The total dimensionality of the descriptor is hence $4 \times 4 \times 8 = 128$. The normalization process and binning scheme is shown in Figure 2.7: the arrow represents the dominant

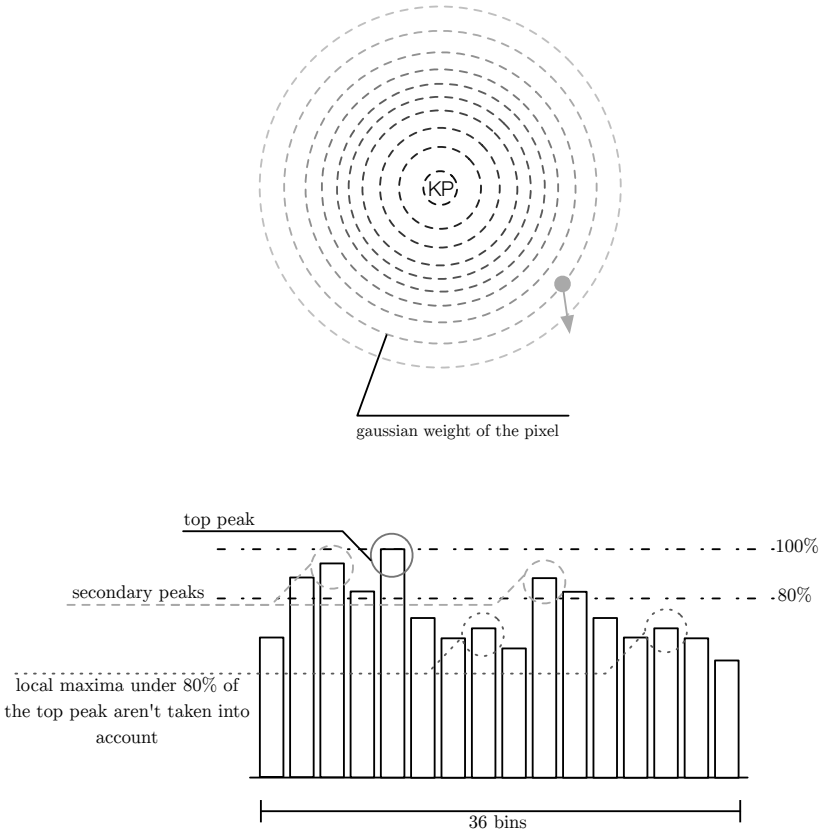


Figure 2.6: Orientation detection phase: Gaussian weights distribution and orientation histogram.

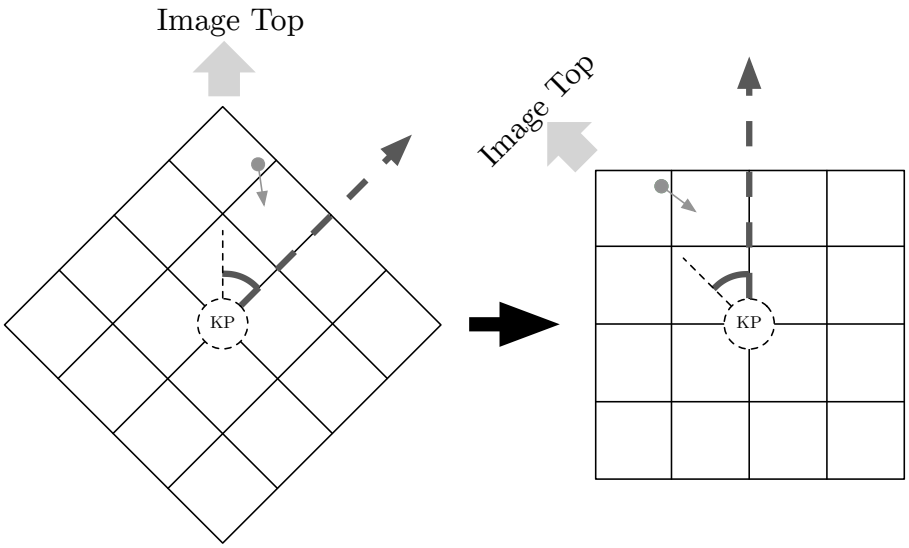


Figure 2.7: Descriptor construction phase: normalisation of the spatial binning grid to the keypoint orientation.

orientation of the keypoint, while the 4×4 grid represents the described spatial binning of the gradients. It is also to be noted that this is a "soft binning", in which each pixel gradient gives contribution to the four nearest spatial bins and to the two orientation bins which lie closer to the gradient phase.

As it has been stated earlier in this chapter, this thesis work is part of a joint development by research groups from University of Naples and University of Salerno. As a result, while some blocks (alpha to b) have been implemented independently and with different approaches by the two involved research groups, some other blocks have been assigned to the groups to be developed with a common interface in mind for later integration. In this perspective, steps c and d have been also designed and developed as part of this thesis work while step e, which for this reason is shown in Figure 2.2 with a dashed outline, has been developed by University of Salerno.

2.2.2 CDVS and SIFT: LoG and DoG

As we mentioned in §2.2.1, one of the most relevant differences between the CDVS algorithm and SIFT (as formulated in [24, 25]) lies in the adopted edge

detection filter. Despite a common rationale consisting of the Scale-Space theory formulated by Lindeberg in [33], which is based on the use of a Laplacian of Gaussian as the generating kernel for the band-pass pyramid, the algorithms feature a different, albeit tightly coupled, choice.

The SIFT algorithm employs, in fact, a well known edge detection filter known as Difference of Gaussian and presented for the first time in [35]. This filter consists, as the name suggests, in the difference between two Gaussian kernels: as proven in [36], selecting a proper ratio between the standard deviations σ_i and σ_e of the two involved Gaussian kernels, the resulting function approximates to a certain extent the profile of the Laplacian of Gaussian (though with some criticalities in the trade-off between bandwidth selectiveness and peak sensitivity of the filter). The advantage in such approach lies in the possibility of performing the filterings with the simple (and separable) Gaussian kernels while subtracting the results of the two filterings afterwards, obtaining the result of the DoG filtering. In this way, a computationally expensive 2D convolution, which would be necessary in case of a LoG filtering (or a direct DoG filtering), is avoided.

CDVS, being oriented to Frequency Domain filtering, does not incur in any computational overhead due to the use of Laplacian of Gaussian kernels. Furthermore, as proven in [37], LoG-based edge detectors produce the most stable features when compared to other typically used kernels.

Figure 2.8 shows the similarity between the two function profiles, and the criticality in the choice of the ratio between the two standard deviations σ_i and σ_e.

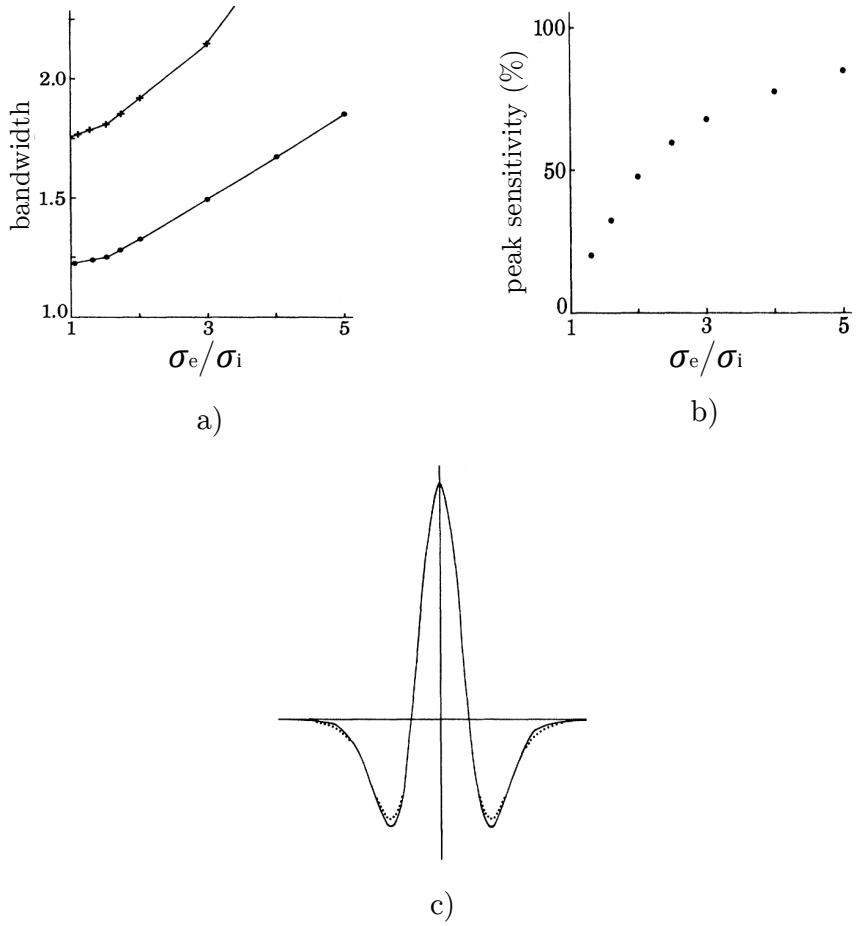
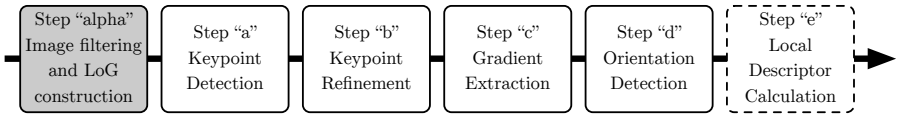


Figure 2.8: LoG approximation by means of DoG: a) In order to maintain selectiveness (i.e. low bandwidth), the ratio between σ_e and σ_i must be maintained low. The figure shows both the half-sensitivity bandwidth (+) and the half-power bandwidth (•). b) On the other hand, low σ_e and σ_i ratios limit the peak sensitivity of the filter: a trade-off is found in values around 1.6. c) The superimposition of a DoG (dotted) and a LoG kernel with appropriate σ. The two profiles are very similar.

2.3 Filtering Unit



The first processor that will be presented is the filtering unit: as we have mentioned before, the novelty of the CDVS approach, with respect to prior, similar feature extraction algorithms, consists in the adoption of a Frequency-based Blockwise filtering for the production of LoG images and low-pass Gaussian images (FBLoG). This approach is based on the proposed filtering processor, which we will now see. The processor is based on a 1D, mixed-radix, Decimation in Frequency, fixed-point FFT Unit which is iteratively used to produce the 2D Fourier transform of the block. The overall architecture of the processor is shown in Figure 2.9: the block size and, thus, the size of the block buffers, has been determined by exploiting the technique described in [38]. This technique takes into account, as a first factor, the "duration" of the impulse response of the filters which will be used in the computation: this impulse response length, in fact, sets the need for an overlap between adjacent blocks which in turn results in overhead of a fixed amount, independent from block size. The second factor to be considered is the total number of operations that will be performed on the single block. The optimization, thus, is carried out by minimizing the product of this amount of operations by the total number of blocks which depends on the blocks overlap and, obviously, on image size: if we denote the image width and height respectively by W and H , the block size by N and the maximum filter "tail" by L, the total number of blocks is easily found as:

$$\left\lceil \frac{W}{N-L+1} \right\rceil \cdot \left\lceil \frac{H}{N-L+1} \right\rceil . \tag{2.1}$$

The next point is determining the computational load for the elaboration of a single block. To accomplish for this, we must analyse the algorithm flow: for each block we need to perform

- N forward FFTs in the row direction
- N forward FFTs in the column direction

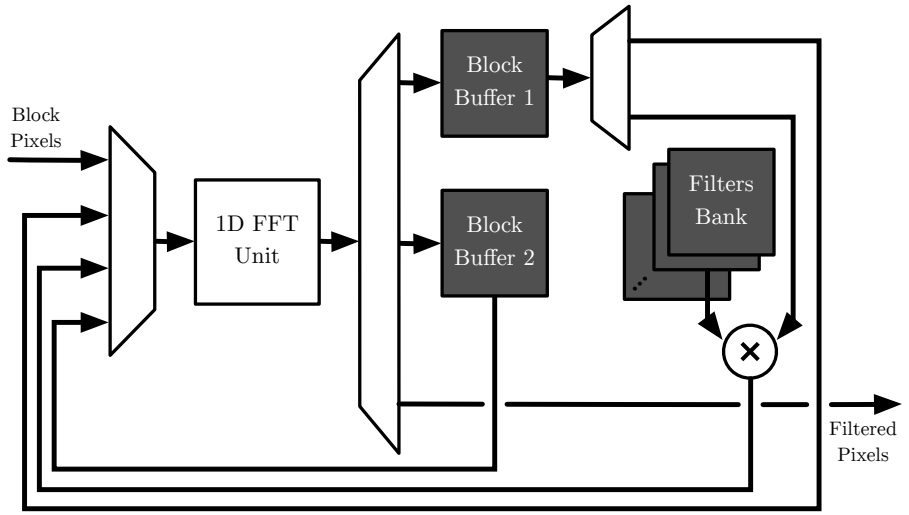


Figure 2.9: Architecture of the filtering unit.

and, for each filter in the filter bank,

- N^2 multiplications
- N inverse-FFTs in the row direction
- N inverse-FFTs in the column direction

Since for each FFT we can assume a complexity of $N \log_2(N)$, supposing the filter bank to be composed by 8 filters coherently with the pyramid construction phase as formulated in CDVS, we obtain:

$$9 \cdot (2N \cdot (N \log_2(N))) + 8N^2. \tag{2.2}$$

Limiting our attention to block sizes that are integer powers of two, we obtain the computational loads in Table 2.1, which refer to a case in which the image resolution is set to be standard VGA size and maximum kernel size "L" is 33, according to CDVS formulation.

As it can be seen from the table, the computational load does not follow an unimodal curve. This is due to the presence of the "ceiling" function in (2.1).

Another optimisation that has been exploited during the development of the filtering processor is connected to the filter response storage: descending from

Table 2.1: Computational loads for various block size which are integer powers of two (VGA image resolution, kernel size=33).

Block size (pixels)	Total computational load (operations)
64	1.43×10^7
128	7.68×10^6
256	8.97×10^6
512	8.91×10^6
1024(†)	1.97×10^7

(†) *indicates that, for this size, the processing is done within a single block. Please note that, for this filter size,* 64 *is the minimum meaningful block size.*

the fact that the Fourier transform of a real and even signal is itself real and even, we can see that it is not necessary to store the totality of the filter samples. Furthermore, being actually our filters radially symmetric, their transform will only depend on radial frequency: as a consequence we can store just a quadrant of its real part (since its imaginary part will be identically zero) and save $7/8$ of the memory that would be needed to store the whole transform.

2.3.1 Filtering Processor Datapath Optimisation

The definition of the processing datapath width has been a crucial step in the definition of the processor architecture. When performing a Fourier transform (FT) in fixed point, the reorganisation of signal energy among the samples can lead to a practical increase in the dynamic range of the manipulated data. As an example, we can consider the DC component of the signal, which corresponds to the sum of its samples. Being our image defined in the $[0, S_{max}]$ range, the DC component of the Fourier transform of a signal composed by 128 samples can be as large as $128 * S_{max} = 2^7 * S_{max}$. This leads to an expansion of 7 bits in the dynamic range of the signal for each FFT step. In addition, the least significant bit (lsb) of the output of the FT will have the same weight as the lsb of the input. As a consequence, to have greater detail in the FT samples (especially in the samples which have low power or, in other terms, smaller absolute values), it is appropriate to add "zeros" as lsbs of the input to artificially expand its dynamic range. Normalisation of the FT output is also a factor that can be considered: at the output of each FT step, a right shift of a

certain amount of bits can be performed to reduce the dynamic range into the datapath width. This normalisation, in a two-steps 2D transform (i.e. obtained from two successive 1D transforms), can be obviously performed after each transform step.

To conclude, this leads to a number of inter-connected parameters in the definition of the datapath:

- Dynamic range of the input signal, in bits (D)
- Datapath width, in bits (W)
- Weight of the lsb, expressed as a power of two (L)
- First FT step normalisation, in terms of right shifts (N1)
- Second FT step normalisation, in terms of right shifts (N2)

where, to have the maximum exploitation of the datapath width, the following condition must apply:

$$D - L - N1 - N2 + 7 + 7 = W. \tag{2.3}$$

This results in a set of configurations all of which produce a certain PSNR when compared to a "golden" reference, which can be represented by floating-point calculation.The PSNR plot of Figure 2.3.1 is the result of a thorough examination of these combinations: the sequence of optimal configurations, one for each datapath width, produces the depicted Pareto curve. Table 2.2 shows the parameters combination for some of the optimal configurations.

As the table shows, the best resource allocation strategy consists in reducing the weight of the lsb in the input signal and performing normalisation after each FT step: in this way, during the data manipulations which are performed inside of each FT, the maximum amount of information is preserved. When the datapath width approaches 22 bits, the optimal configurations become almost equivalent and further increments do not provide significant gain in terms of computational precision.

Table 2.2: Parameters configuration for some of the optimal filtering unit configurations (D=8).

Datapath size (pixels)	L (2's power)	N1 (right shifts)	N2 (right shifts)	PSNR dB
14	-5	7	7	29.02
15	-6	7	7	33.82
16	-7	7	7	38.31
17	-8	7	7	41.98
18	-9	7	7	44.24
19	-9	6	7	45.24
20	-10	6	7	45.58
21	-10	6	6	45.68
22	-11	6	6	45.71
23	-12	6	6	45.72
24	-12	6	5	45.72
25	-13	6	5	45.72
26	-15	6	6	45.72

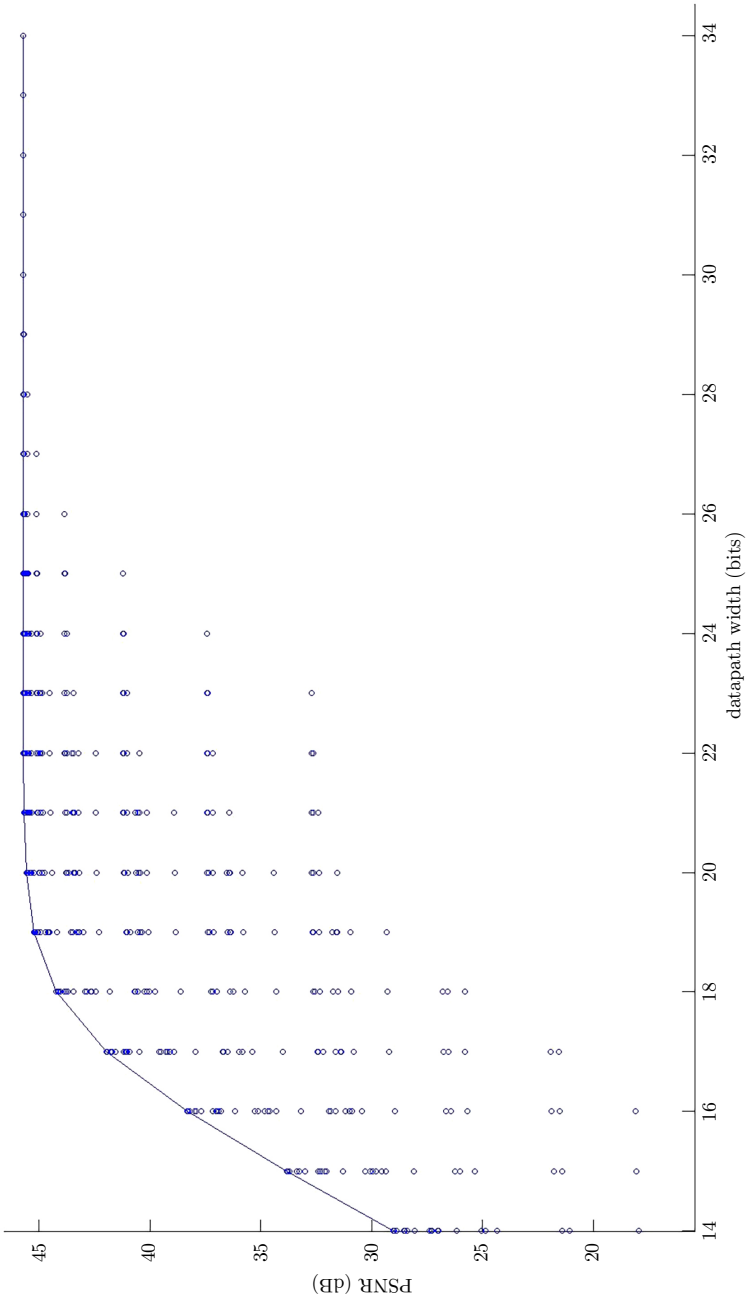


Figure 2.10: PSNR values for different configurations of the filtering unit. The dots indicate possible configurations that satisfy condition (2.3). The Pareto curve traces the sequence of optimal configurations for each datapath width.

Table 2.3: Implementation results for the filtering processor on an Altera Stratix IV device.

Logic Elements (amount)	Block RAM (kBs)	DSP Elements (amount)	f_{max} (MHz)
7486	304.8	96	93.23

2.3.2 Implementation Results

Table 2.3 shows the implementation results for the 22 bit datapath version of the filtering processor on an ALTERA Stratix IV device. To obtain the image throughput of the processor we must take into account the following:

1. The filtering processor outputs two pixels per clock cycle
2. A VGA image decomposes into 35 blocks
3. Each block is processed in 147k clock cycles.

As a consequence, the throughput, in terms of frames per second, can be obtained as:

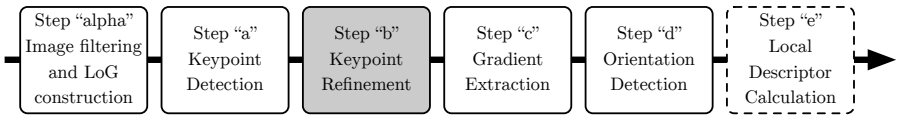
$$\frac{9.32e7}{35 * 147k} \simeq 18fps. \tag{2.4}$$

This throughput can be considered sufficient for real-time applications in which typical frame rates are around 24 fps: a simple 2-to-1 subsampling in time domain would lead to a frame rate of 12 fps that could be accommodated by the unit. In addition, the proposed architecture adapts to many possible optimisation: as an example, by adding a second FFT unit, the inverse FFTs could be parallelised: in this way, each block would be processed in $\sim$82k clock cycles, leading to a throughput of 32.5fps.

Figure 2.11 shows a comparison between keypoints detected by the realized processor and keypoints extracted by the CDVS TMuC. As the image shows, the majority of the most significant keypoints (the ones with greater contrast and localization) is similarly distributed.

Figure 2.11: Comparison between keypoints extracted by the proposed processor (red) and keypoints extracted by the CDVS TMuC (blue). Perfectly matching keypoints are shown in green.

2.4 Keypoints Refinement Unit



The Keypoint Refinement phase has the following two functions:

- Individuating local extrema at a sub-pixel scale by a first order Taylor expansion of the 3D-fitting function of the Laplacian of Gaussians.
- Discarding unstable keypoints, such as edge responses which are poorly localized (unlike angle keypoints) or low contrast keypoints.

The algorithm firstly approximates first and second order derivatives by means of finite differences, and then obtains the offsets $\{\delta x, \delta y, \delta s\}$ by solving the following linear equations system:

$$\begin{bmatrix} Dxx & Dxy & Dxs \\ Dyx & Dyy & Dys \\ Dsx & Dsy & Dss \end{bmatrix} * \begin{bmatrix} \delta x \\ \delta y \\ \delta s \end{bmatrix} = \begin{bmatrix} -Dx \\ -Dy \\ -Ds \end{bmatrix},$$

where, if $P(x, y, s)$ represents the pixel at (x, y, s) in the scale-space, then for example Dx represents the quantity $[P(x+1, y, s) - P(x-1, y, s)]/2$ and Dxy represents the quantity $[P(x+1, y, s) - P(x-1, y, s) + P(x, y+1, s) - P(x, y-1, s)]/4$. As a consequence, $Dxy = Dyx$ and the matrix is symmetric.

If any of the obtained spatial offsets $\{\delta x, \delta y\}$ has absolute value greater than 0.6 the corresponding coordinate (and thus the keypoint candidate) is shifted accordingly and the algorithm is iterated. After five iterations (or less if, at any point, the offsets are under 0.6 in absolute value) the algorithm is stopped: in case any of the offsets at the last iteration is greater in absolute value than a threshold of 1.5 pixels, the candidate is discarded; otherwise, the offsets constitute the final, sub-pixel coordinates.

After this stage, the quantity $P(x + \delta x, y + \delta y, s + \delta s)$ is calculated on the basis of the first order Taylor expansion. This interpolated peak value is then compared with a threshold equal to $0.03 * Pmax$, where $Pmax$ is the

maximum possible pixel value. If the peak absolute value is lower than the threshold, the keypoint is considered of low contrast and, thus, discarded.

At this point, given the Hessian matrix

$$H = \begin{bmatrix} Dxx & Dxy \\ Dxy & Dyy \end{bmatrix},$$

its determinant $\det(H) = Dxx * Dyy - Dxy * Dxy$ and its trace $\mathrm{tr}(H) = Dxx + Dyy$ are calculated: in case the determinant is negative, the point is a "saddle" and it is thus discarded.

The matrix trace and its determinant are bound to the eigenvalues α and β as follows:

$$\mathrm{tr}(H) = \alpha + \beta,\ \det(H) = \alpha\beta.$$

Let now be $\alpha = r\beta$:

$$\frac{\mathrm{tr}(H)^2}{\det(H)} = \frac{(\alpha+\beta)^2}{\alpha\beta} = \frac{(r\beta+\beta)^2}{r\beta^2} = \frac{(r+1)^2}{r}.$$

This ratio is at its minimum when the two eigenvalues are equal and it increases with r. Now, being the eigenvalues proportional to the principal curvatures of the curve around the point in which the derivatives are calculated (as proven in [39]), we compare the ratio with a proper threshold r, which is, in [25], proposed as 10. Above this value, we consider the keypoint to be representative of an edge and, thus, poorly localized and prone to be unstable.

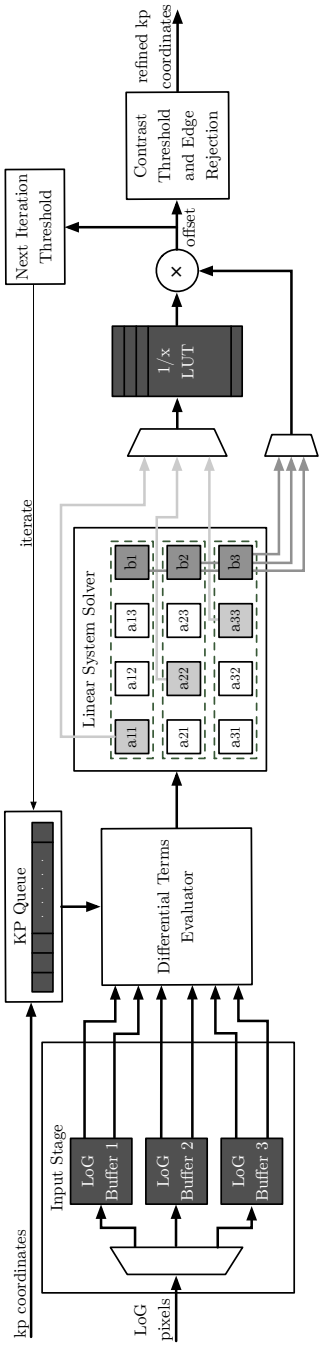


Figure 2.12: Architecture of the Keypoint Refinement unit.

2.4.1 Proposed Processor Architecture

The proposed processor has been designed to receive the input image LoGs on a block-by-block basis from the filtering unit. An intermediate buffer between the upstream steps and the keypoint refinement processor is used to discard the outmost pixels of each block so that only 96×96 pixels are processed by the unit.

Processor architecture is shown in Figure 2.12: the 96×96 pixels blocks are received streamingly in interleaved order (LoG #0 of the first block, LoG #1 of the first block, ..., LoG #4 of the first block, LoG #0 of the second block, etc.) and, to optimize buffering, they are stored in round-robin into three block buffers.

Every time the extrema detector identifies a keypoint in the block, the coordinates are put into a KP queue and sent to the processor, which fetches the needed pixels from the buffers and calculates the differential approximations. The fetch order has been engineered to reduce the number of memory accesses to the minimum and parallelise them as much as possible: the standard CDVS algorithm performs a total of 27 memory accesses, while our processor only carries out 19 accesses. By holding values that are used more than once in registers and exploiting parallel memory access on the FPGA dual-port memories, only 5 clock cycles are actually sufficient to read all the needed pixel values, with a notable reduction on access times and an optimised exploitation of the memory bottleneck. Table 2.4 shows the order of memory accesses on the BRAM ports: the triad (x, y, s) represents the offset of the accessed neighbouring pixel with respect to the candidate $(0, 0, 0)$.

Table 2.4: Differentials calculation memory fetch order on dual port LoG buffers: pixel coordinates are expressed as (x, y, s).

Buffer/	Cycles				
Port	1st	2nd	3rd	4th	5th
1A	(0,0,-1)	(-1,0,-1)	(0,-1,-1)	(+1,0,-1)	(0,+1,-1)
2A	(0,0,0)	(-1,0,0)	(0,-1,0)	(+1,0,0)	(0,+1,0)
2B	(+1,+1,0)	(+1,-1,0)	(-1,+1,0)	(-1,-1,0)	none
3A	(0,0,+1)	(-1,0,+1)	(0,-1,+1)	(+1,0,+1)	(0,+1,+1)

After this phase, the linear equations system is populated and solved with a division-free algorithm, derived from the classical Gauss Jordan: this algo-

rithm avoids division during the transformations of the linear system matrix, only requiring divisions for the final calculation of the solutions as the ratios between the constant terms b_i and the pivot elements A_{ii}. However, by aid of a LUT which computes the $1/x$ function, this calculation of the final divisions is also avoided. If any of the obtained spatial offsets is greater than the threshold, the keypoint candidate is shifted accordingly and the procedure is iterated; otherwise, a low-contrast and edge rejection block checks if the keypoint is stable enough to be further considered by the CDVS algorithm (by checking the conditions discussed in the previous section) and, if so, outputs the refined coordinates.

2.4.2 Algorithm optimization for fixed-point hardware design

Being the CDVS algorithm software-oriented, the released C/C++ implementation requires complex libraries and high processing power, hardly obtaining real-time performance. Our processor is aimed at a completely different scenario, namely mobile and low-power devices such as smartphones or sensor networks. As a consequence, a fixed point precision rework of the algorithm has been conducted to jointly adapt to hardware capabilities and limitations and meet real-time requirements.

Division Free Linear Equation System Resolution

The linear system resolution has been performed with a division-free variant of the Gauss method known as the "one step division-free Gaussian elimination" [40, 41]: the chosen method transforms the A matrix of the linear system into a diagonal matrix, on the basis of the circuit presented in [42]. As a consequence, divisions are only needed for the final step of dividing the constant terms by the coefficients on the main diagonal and obtain the unknown variables vector. Table 2.5 shows the number of multiplications and divisions for both the one-step method, the traditional Gauss method (which transforms the matrix into a triangular matrix) and the Gauss-Jordan method (which, as the one-step method, transforms the matrix into a diagonal matrix) as presented in [43]: the final divisions are implemented using a Lookup Table (LUT) for calculation of $1/x$.

Table 2.5: Computational complexity (in terms of multiplications and division) of Linear Equation Systems resolution algorithms.

Operation	One Step	Gauss	Gauss-Jordan
Divisions	3	6	9
Multiplications	45	18	27

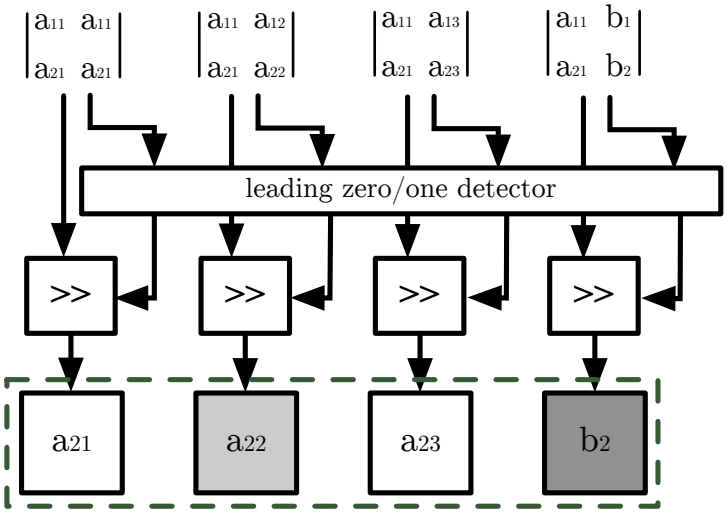


Figure 2.13: Dynamic rescaling of the Linear System Equations.

Linear equations system dynamic rescaling

One of the optimizations carried out to better exploit fixed-point precision is the dynamical rescaling of equations by use of a leading zero/one detector during the linear system resolution: this is based on the well-known property for which, if one of the equations of a linear system is multiplied by a coefficient, the solution is unchanged.

Being the general term of the system a_{ij}, at each step, given by the determinant of a 2×2 matrix composed by four terms of the system at the previous iteration, the dynamic range of the terms would linearly increase with the iteration: as a consequence, being the initial terms 16 bit wide, at the end of the third iteration the term should theoretically be 128-bit wide, in order to prevent

any overflow. To reduce register size a rescaling could be performed at each step, but with a detrimental effect on output precision: to minimize this, the rescaling is dynamically adapted in order to perform only the strictly necessary shifts to obtain overflow avoidance. As Figure 2.13 shows, in fact, a leading zero/one detector is utilized to detect how many shift operations are needed to avoid overflow and the shifters are accordingly operated.

Table 2.6 summarizes the advantage introduced by this technique over a classical, static rescaling approach: the error metric is expressed in terms of a "failure rate" or, in other terms, the percentage of cases in which the absolute error on the offsets is over 0.5 pixels, which can lead to a failure of the refinement for the considered keypoint. As the numbers show, to obtain comparable precision it is necessary to implement a considerably larger circuit, ideally the full 128-bit wide circuit with no rescale, with a much higher usage of DSP blocks and a much lower maximum clock frequency.

Table 2.6: Effect on precision, timing performance and resource occupation of the dynamic rescaling linear system solver.

Case	Failure Rate(†)	Occupied Logic		f_{max}
f (registers size/rescale)	(%)	(Slices)	(DSP)	(MHz)
18-bit/Dynamic(*)	0.12	3244	24	183,79
18-bit/Static	95.2	1210	22	183,99
90-bit/Static	0.11	11204	704	73,02
128-bit/No Rescale	0	3765	200	90,6

(*)*this thesis*

(†)*The refinement is considered to fail over a given system when the absolute error in the offset calculation is over* 0.5

Non-Uniformly Spaced $1/x$ LUT with Mantissa and Exponent

The $1/x$ function has a notable output range over the considered input interval ($]0, 2^{17} - 1]$). To obtain accurate yet small sized LUT versions of the function several approaches exist in literature, like range reduction (see [44]). Given the architectural characteristics and resource availability of FPGAs, however, our implementation has been targeted towards a simpler approach which, levering on some of the basic principles of range reduction such as lead zero detec-

tion while utilizing a non-uniformly sampled range with piecewise constant approximation, obtains good precision and area utilization. Figure 2.14 shows such circuital implementation.

The input range is divided in two segments. As a consequence, the input value x is converted into an unsigned value and then the 10 MSBs and the 10 LSBs are extracted: these two sections of x are utilized to address two LUTS.

If the bitwise OR of the leading 7 bits of the MSBs is "0" the output from LUT C is selected: this LUT holds the values for the $]0, 2^{10} - 1]$ interval, in which the first derivative of the function has greater absolute values and high precision is crucial. In this interval, thus, the function is sampled on a point-per-point basis. If the 7 MSBs are not all zeros, otherwise, the output is taken from LUT I: this holds the values for the $]2^{10}, 2^{17} - 1]$ interval. Being the derivative smoother in this interval, the function is sampled with a step equal to 2^7 and interpolated using zero order interpolation.

In each case, the output value is represented by a mantissa and by an exponent which represents the LSB weight. Conversion to fixed point is eventually performed by shifting the mantissa or its complement (if the bit sign is equal to 1).

Table 2.7 compares precision and LUT size of the optimized segmented LUTs with mantissa and exponent (M,E) architecture to non segmented (completely tabulated) LUTs with mantissa and exponent and standard LUTs.

Table 2.7: Comparison of precision and LUT size of the proposed segmented LUT technique with mantissa and exponent (M,E) with standard implementations.

Case	Error (RMS)	LUT Size (KBs)
11 bit (M,E), segmented* (this thesis)	1.89e-6	3.98
10 bit (M,E), completely tabulated	1.04e-6	240
11 bit (M,E), completely tabulated	4.35e-7	256
18 bit (M,E), completely tabulated	3.97e-9	368
18 bit Standard, completely tabulated	1.08e-6	288
27 bit Standard, completely tabulated	4.30e-9	432

**LUT is completely specified up to* 2^{10}*. Afterwards, one value is specified each* 2^7 *points.*

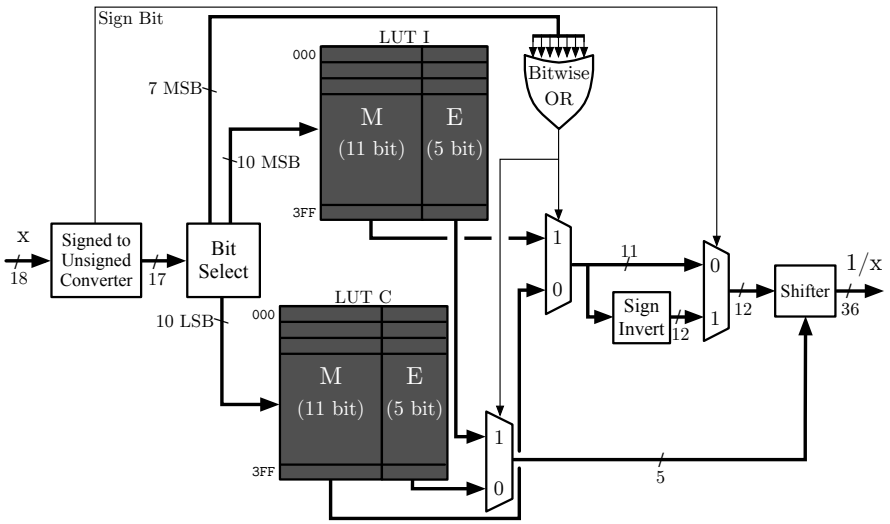


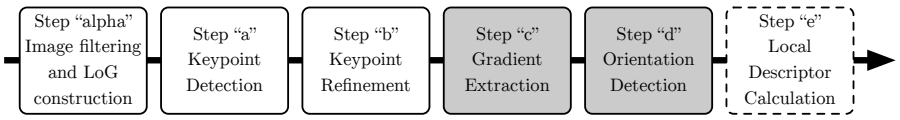
Figure 2.14: Architecture of the non-uniformly spaced $1/x$ LUT with Mantissa and Exponent.

2.4.3 Implementation Results

The presented processor has been implemented and deployed on an ALTERA Stratix IV EP4SGX230KF40C2 FPGA. The footprint of the processor is negligible on such a mid-sized device, with the Utilized Logic Slices resulting in 5080 (3%), the BRAM usage equal to 540 KB (4%), and the DSP blocks usage equal to 30 (2%).

For what concerns performance analysis, if we consider that each iteration of the algorithm on the keypoint requires 30 clock cycles and that the maximum number of iterations is equal to 5, the obtained maximum working frequency of 100 MHz allows a throughput of 6.67×10^5 KPs/sec, which in turn results in a maximum throughput per frame (at a theoretical frame rate of 30fps) of 2.22×10^4 KPs per frame. Simulation results confirm a refinement failure rate (as defined in §2.4.2) under 1%, with a mean absolute error of 3.6×10^{-2} pixels on successfully refined keypoints.

2.5 Gradients Extractor and Orientations Detector



One of the characteristics of the CDVS algorithm is to produce "features" or, in other terms, descriptors which exhibit robustness with respect to several image transformations such as rescalings, rotations, change in illumination and point of view, or perspective distortions. This robustness is obtained as a result of several aspects of the algorithm: as an example, the concept itself of image pyramid and multi-scale analysis leads to a rescaling invariance of the descriptors. To obtain robustness to rotation, conversely, descriptors are "normalised" with respect to one or more "dominant orientations" which characterise the keypoint.

In CDVS, the dominant orientations are calculated by examining the image luminosity gradients in the areas surrounding the keypoints and accumulating their phases in a histogram. As a first step, thus, the luminosity gradients need to be extracted in terms of their magnitude and phase from the appropriate image in the low-pass pyramid. Being the image denoted with $I(x, y)$, the gradients are extracted as

$$m(x,y) = \sqrt{[I(x+1,y) - I(x-1,y)]^2 + [I(x,y+1) - I(x,y-1)]^2}, \tag{2.5}$$

and

$$\theta(x,y) = \tan^{-1}\left(\frac{I(x,y+1) - I(x,y-1)}{I(x+1,y) - I(x-1,y)}\right). \tag{2.6}$$

Gradients are evaluated inside of a circular patch of diameter $4.5 \cdot \sigma_{kp}$, where σ_{kp} corresponds to the interpolated scale coordinate of the keypoint, as calculated in step "b". As a consequence, keypoints detected in higher scales will have a larger neighboorhood of gradients analysed for orientation detection.

The histogram is composed by 36 bins, with the first centred at 5°, and each bin spanning 10° for a total of 360°. Gradients contributions, as Figure 2.15

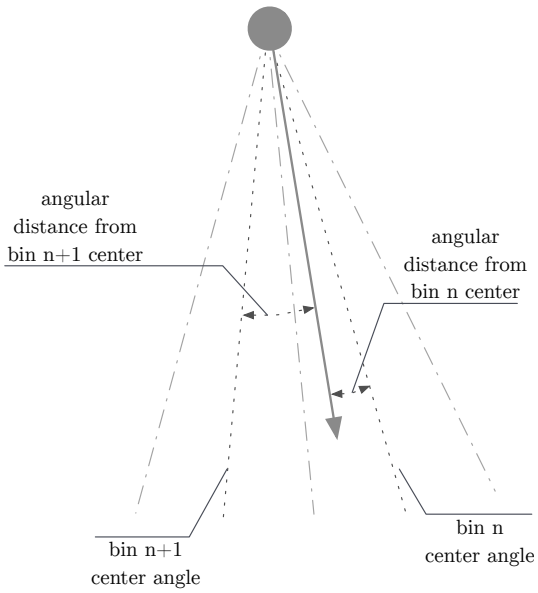


Figure 2.15: Soft binning of the pixel gradients: the weight of the contribution to bin n is inversely proportional to the distance between the gradient phase and the central angle of the bin itself.

shows, are calculated with a "soft binning" technique: while with traditional binning, for example, a gradient with phase equal to $14°$ would just give contribution to the bin centred at $15°$, in this scheme it would give a contribution of 0.9 to the bin centred at $15°$, plus a contribution of 0.1 to the one centred at $5°$.

In general terms, the contributions to the upper and lower bin are weighted with the factors

$$w_u = 1 - \Delta_{\theta u}/10, \quad w_l = 1 - \Delta_{\theta l}/10, \tag{2.7}$$

where $\Delta_{\theta u}$ and $\Delta_{\theta l}$ represent the differences between the angle of the gradient phase and, respectively, the centre of the upper and lower bin as expressed in sexagesimal degrees: as a consequence, the following conditions apply.

$$\Delta_{\theta l}, \Delta_{\theta u} \in [0, 10] \tag{2.8}$$

$$w_u, w_l \in [0, 1] \tag{2.9}$$

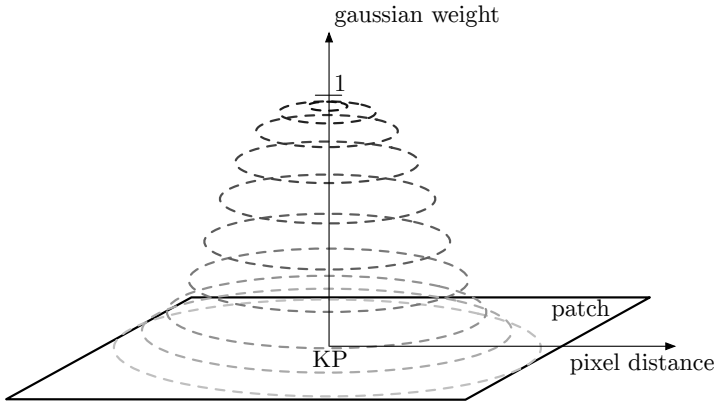


Figure 2.16: Representation of the Gaussian window: each pixel in the patch gives a contribution which is inversely proportional to its distance from the Keypoint location.

$$w_u + w_l = 1 \tag{2.10}$$

The contribution is also weighted with the distance between the keypoint location and the pixel coordinates following a Gaussian profile, as shown in Figure 2.16. This "Gaussian weight" is equal to 1 at the keypoint sub-pixel coordinates (as generated at step "b"), descending towards zero as the distance from the keypoint increases.

The final formulation of the contributions to the upper and lower bins, c_u and c_d, is:

$$c_u(x, y) = m(x, y) \cdot G(d) \cdot w_u(\theta(x, y)) \tag{2.11}$$

$$c_d(x, y) = m(x, y) \cdot G(d) \cdot w_d(\theta(x, y)) \tag{2.12}$$

where $m(x, y)$ is the gradient magnitude in (x, y) and $G(d)$ is the Gaussian weight of the pixel at distance d from the keypoint location.

When the histogram is completely built, a rolling average smoothing is operated. To reduce window size to three, the operation is repeated for a total of six iterations: the equivalent smoothing kernel is shown in Figure 2.5. After this phase, the histogram is scanned for up to four local maxima: local maxima with value under the 80% of the global maximum of the histogram will not be considered. The local maxima are further interpolated with a parabolic

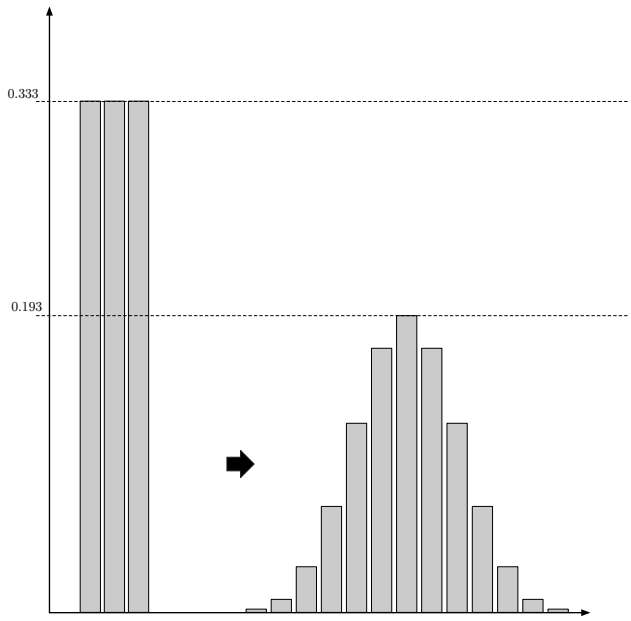


Figure 2.17: Rolling average smoothing: 3 taps window and equivalent smoothing kernel after the imposed six iterations.

interpolation as Figure 2.5 shows: in this way the keypoint orientation can be calculated with maximum precision.

2.5.1 Processor architecture

Figure 2.19 shows the architecture of the orientation detection processor: the circuit takes as input the gradients calculated at step "c" (expressed in magnitude and phase) as well as information regarding pixel distance and keypoint scale (σ). The two units inside of the histogram builder indicated as "Gaussian weights generator" and "Angle binner" are the most crucial units for the execution of the algorithm: we will analyse them in major detail in this section. As the figure shows, the histogram builder makes use of the squared distance between the keypoint location and the involved pixel, to simplify the calculation and avoiding the computation of the square root.

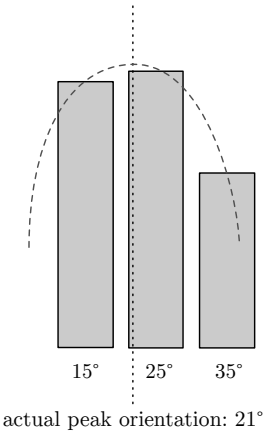


Figure 2.18: Histogram peak parabolic interpolation.

Gaussian weight generator

The Gaussian weight generator calculates the weight to be assigned to the generic pixel of coordinates (x, y), situated at a distance d from the coordinates of the keypoint (x_{kp}, y_{kp}). The value of the output is not only dependent

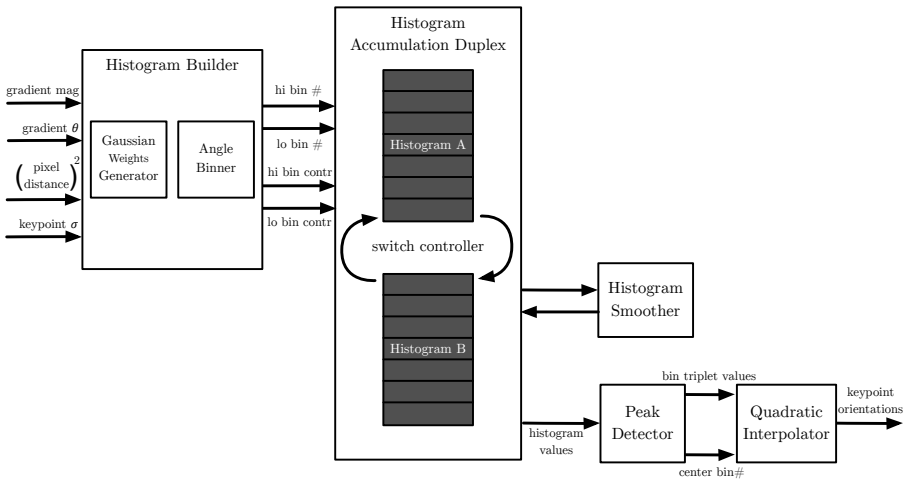


Figure 2.19: Architecture of the Orientation Detection processor.

on d, but also on the keypoint scale coordinate σ_{kp}. The Gaussian function has, in fact, formulation:

$$G(d, \sigma_{kp}) = e^{-\frac{d^2}{2(1.5 \cdot \sigma_{kp})}}. \tag{2.13}$$

The 1.5 coefficient which contributes in determining the standard deviation of the Gaussian curve is the window magnification factor, as defined by the CDVS algorithm formulation.

Due to the non-linearity of the described function, the calculation has been implemented in tabulated form, as the cascade of two units: the first unit calculates the reciprocal value for $(1.5 \cdot \sigma_{kp})$, while the second one calculates the actual exponential

$$\exp\left(d^2 \cdot rec(4.5 \cdot \sigma_{kp})\right) \tag{2.14}$$

The tabulation of the exponential and reciprocal function is performed with a linearly interpolated approach, to obtain a trade off between simplicity and accuracy.

With such an approach, the function value $f(x)$ is approximated as:

$$\begin{aligned} f(x) \simeq \phi(x) = f(x_i) + \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i} \cdot (x - x_i) = \\ = f(x_i) + s(x_i) \cdot (x - x_i), \end{aligned} \tag{2.15}$$

where $s(x_i)$ represents the slope of the interpolating segment between x_i and x_{i+1}.

As a consequence, to avoid computing slopes "on-line", it is necessary to adopt an architecture based on [45] which makes use of two Look Up Tables (LUT): the first for storing the samples $f(x_i)$, and the second for the storage of the slopes $s(x_i)$.

The choice of the sampled values for the LUTs is made so that the most significant bits of the x values are directly used to access the stored values: in other terms, supposing that the interval of interest be $x \in [0, 2^N - 1]$, to be sampled in $N = 2^m$ points, the function is sampled in the set S defined as

$$S = \{n \cdot 2^{(N-m)-1}\}, \tag{2.16}$$

where n is a positive integer in the range $[1, 2^m]$.

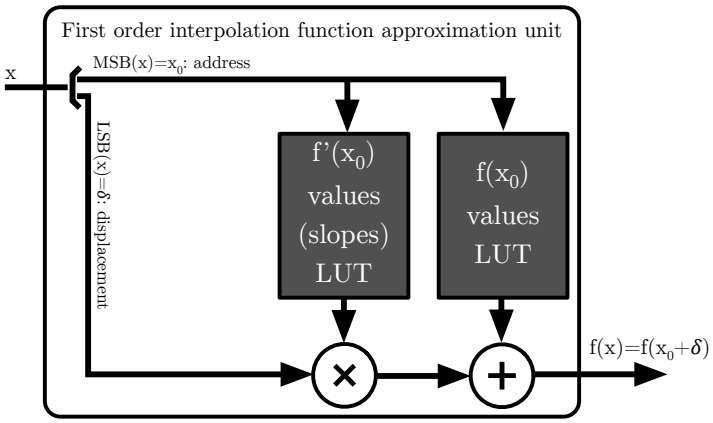


Figure 2.20: Block diagram of the first order approximation unit used in the Gaussian Weight Generator.

In this way, as it can be easily seen, the m most significant bits of x can be used to access the LUTs. The residual $N-m$ least significant bits, conversely, will be used to determine the offsets with respects to the stored values and calculate the linear interpolation.

The diagram for the described scheme of first order approximation unit is shown in 2.20.

At this point it is crucial, to optimise the tables, to define the input ranges for the inputs of the LUTs, d and σ_{kp}. In the CDVS algorithm, orientation histograms are built from a neighbourhood of the keypoint which can be represented as a circle contained in a square "image patch" with a fixed maximum size (in our implementation, 24 pixels). In some critical cases (when the keypoint is near the image borders) the patch can also be rectangular. As a consequence, the distance d is limited by the size of this square patch to:

$$d^2_{MAX} = 12^2 + 12^2 = 288 \Rightarrow d^2 \in [0, 288]. \tag{2.17}$$

On the other hand, the considered neighbourhood has a radius which is equal to:

$$r = \lfloor (4.5 \cdot \sigma_{kp}) \rfloor. \tag{2.18}$$

By analysing experimentally the outputs of the CDVS dataset, it has been noted that typical standard deviation values $\sigma_{kp}$are in the interval [0.36,1.15].

For keypoints having larger scale, thus, the Gaussian tails of the distribution of weights will be truncated: of course, these value would anyway be less impacting on the final histogram. It has to be noted, furthermore, that the minimum value for σ_{kp} is not zero: this can be exploited to reduce LUT size by subtracting a constant value from the x to obtain the data address in the Look Up Table.

Let us now consider the LUT for the calculation of the exponential function: as a first consideration, we may notice that the minimum significant output for the exponential generator is

$$G_{min} = 2^{LSB_G} = e^{-z_{MAX}}. \tag{2.19}$$

As a consequence, the maximum z value to produce a non-zero output will be:

$$z_{MAX} = -ln(2^{LSB_G}) = -LSB_G * ln(2). \tag{2.20}$$

For any z greter than this threshold the output will be identically zero. To gain maximum precision, thus, we will scale the z input with a scale factor α which is easily determined as:

$$\alpha = -\frac{2^k - 1}{LSB_G * ln(2)}. \tag{2.21}$$

We will then normalise z by compensating this scale factor inside of the LUT values, so that the actually tabulated function will be:

$$G(d^2, \sigma_{kp}) = e^{d^2/2\cdot(4.5\sigma_{kp})^2} = e^{-z} = e^{-z^*/\alpha}, \tag{2.22}$$

where $z^* = z \cdot \alpha$.

So, summing it up, the tabulated functions will be:

$$rec(\sigma_{kp}) = \frac{\alpha}{2 \cdot (4.5\sigma_{kp})^2}, \tag{2.23}$$

and

$$\exp(z^*) = e^{-z^*/\alpha}. \tag{2.24}$$

The final architecture of the Gaussian weight generator is shown in Figure 2.21.

The obtained circuit provides a number of parameters to be defined, even if we consider the representation of the inputs d^2 and σ_{kp} as being fixed: in

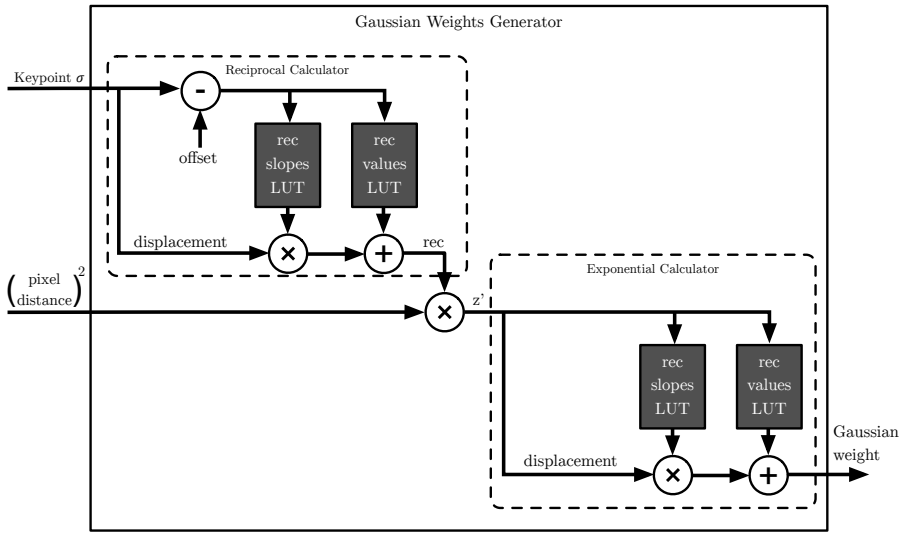


Figure 2.21: Architecture of the Gaussian Weight Generator.

particular, for each of the LUT the number of entries and the representation for each of them must be determined. In § 2.5.2 we will see the effects of tuning these parameters to control the produced approximation error in the detected orientations.

Angle Binner

The second block which constitutes the orientation detection processor is the angle binner, which determines the two bins the pixel gradient will give contribution to, also calculating the weight of the contribution to each of them on the basis of the angular distance between the bin centre and the actual gradient phase (a scheme that is already been described with the name of soft binning).

The only input to the angle binner is the gradient phase in the considered pixel, which is normalised to π and expressed on 15 bits, so that:

$$\theta(x, y) \in [0, 2[\tag{2.25}$$

and the weights of the most and least significant bits of the signal are, respectively, 0 and -14.

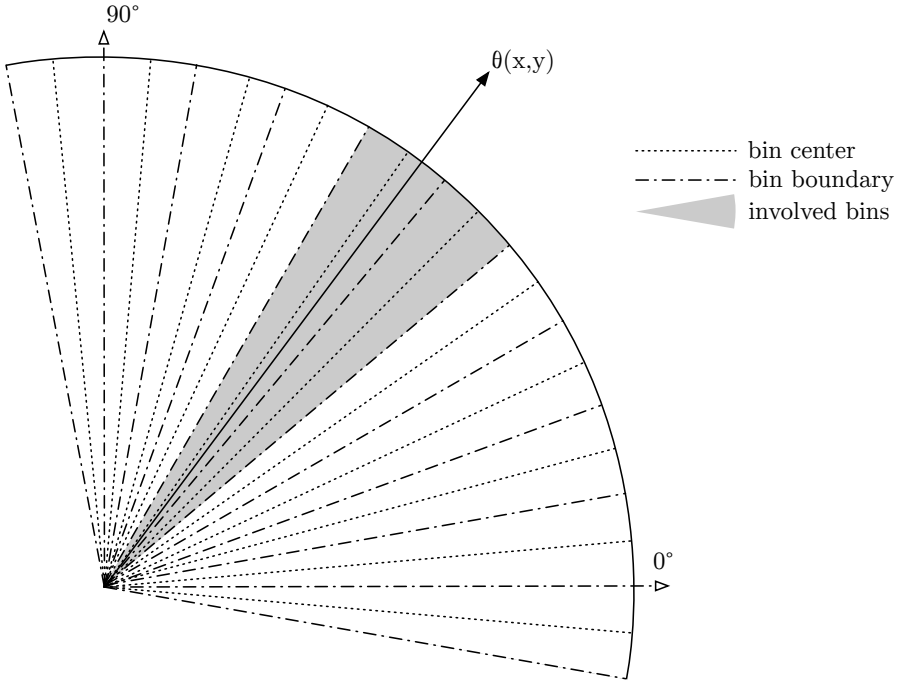


Figure 2.22: Division of the first quadrant into 11 bins.

The histogram divides the range of the phase into 36 bins: being this number not a power of two, it is not possible to adopt an approach similar to the one used for the LUTs, i.e. using the most significant bits of the input to identify the bin. On the other hand, since $36 = 2^2 * 9$, this technique can be exploited, by examining the 2 most significant bits of the phase, to identify the quadrant in which the gradient lies in. The other bits of the phase can, then, be used to discriminate amongst the 9 bins in each quadrant, with an overall reduction in comparison complexity: the quadrant can be thus utilised as an "offset".

With this approach, only 11 subtracters are needed to identify the pair of bins the gradient gives contribution to: 9 of those are needed to discriminate amongst the 9 bins of the quadrant; the other two are needed to compare the gradient phase with the first bin of the second quadrant and the last of the fourth quadrant, as Figure 2.22 shows.

At this point, the sign of the subtractions output will easily give the two bins to which the gradient shall give contribution: in fact, one of the two will

exhibit a positive difference between its centre angle and the gradient phase, while the other one will exhibit a negative difference.

The other bits of the two difference values which individuate the pair of affected bins will be used to determine the contribution weight: to accomplish for this, a normalisation consisting in a subtraction with a constant term is performed. The final architecture for the angle binner is in Figure 2.23.

2.5.2 Implementation results

The processor architecture implies a typical feed forward cascade of block, which adapts naturally to a development in pipeline to improve maximum frequency and, thus, throughput. The insertion of 11 pipeline levels, as proven in [46], reduces overall power consumption of the processor, while maximizing clock frequency.

To obtain processor throughput, we can consider the worst case scenario in which each keypoint requires the largest possible patch, composed by 24×24 pixels. The approximate, worst-case number of clock cycles in which the orientation detector will process a patch is, thus, $24 * 24 = 576$.

The circuit has been deployed on ALTERA Stratix IV FPGA, setting the pipeline for the target clock frequency of 200 MHz, obtaining the results in Table 2.8. At this clock speed, the throughput of the processor is:

$$\frac{212.04 \ \ MHz}{576 \ \ clock \ \ cycles/KP} = 368.125 \ \ KPs/s, \tag{2.26}$$

which, at a theoretical frame rate of 30fps, would in turn result in a maximum elaboration capacity of

$$\frac{368.125 \ \ KPs/s}{30 \ \ fps} = 12.271 \frac{KPs}{frame}, \tag{2.27}$$

which is certainly over the requirements of a typical VGA elaboration, and makes the processor suitable also for high-definition processing. The obtained architecture is, in fact, completely independent of image size.

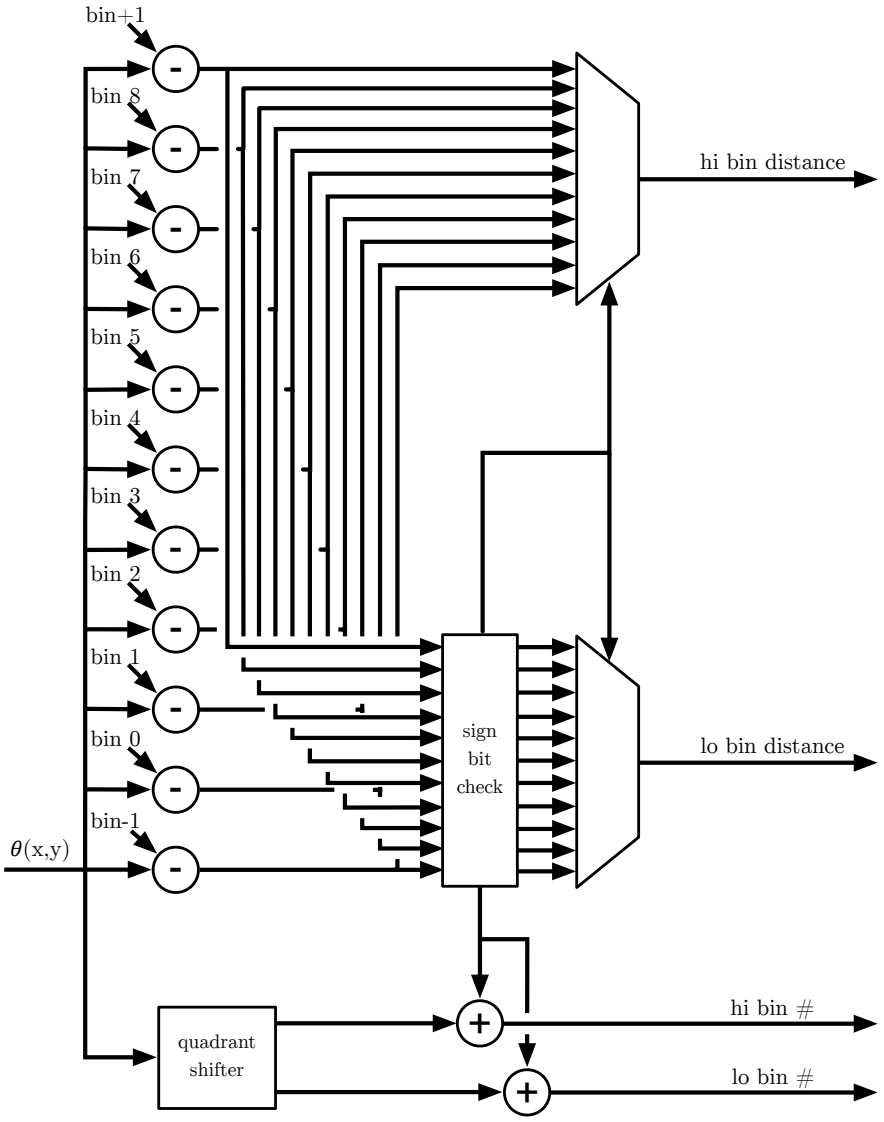


Figure 2.23: Architecture of the Angle Binner.

Table 2.8: Implementation results for the Orientation Detector on ALTERA Stratix IV FPGAs.

Logic Elements (amount)	Block RAM (kBs)	DSP Elements (amount)	f_{max} (MHz)
4111	0	4	212.04

Chapter 3

An Algorithm for Fast Lossless Compression of Hyperspectral Images

Multispectral and Hyperspectral Images are widespread in Remote Sensing and Space Imaging Applications. They are three dimensional arrays of pixels and their raw content may include thousands of bands for a total of over a billion pixels. For adequate storing and transmission over satellite communication links a proper compression scheme is required. To this purpose the Consultative Committee for Space Data Systems (CCSDS) standardized, on May 2012, the 123.0-B-1 standard algorithm for lossless compression of hyperspectral images.

In this chapter, we will see a modified algorithm for such a lossless compression that is bit-true to the standard algorithm while allowing higher throughputs in hardware implementations. A relatively inexpensive FPGA implementation of the algorithm is also presented. The proposed circuit is, according to the CCSDS standard, user configurable in various aspects.

The proposed architecture is composed by a space-class FPGA device (Virtex 5QV SX50T), in which the proposed algorithm is implemented, and an external DRAM. The realized circuit overcomes the state-of-the-art in terms of image pixel throughput (reaching over 100 MSamples/s) and has a negligible footprint in terms of resource allocation (13%) on the considered, small sized FPGA. The high throughput of this architecture, furthermore, enables it to be compliant to the data rate requirement of highly demanding remote sensing

missions such as NASA HyspIRI.

3.1 Introduction: hyperspectral images and the CCSDS-123.0-B-1 algorithm

Hyperspectral images (Figure 3.1) are three-dimensional arrays of pixel data in which each layer is a matrix representing a single image band. Typically, all three dimensions of the matrix can be in the order of magnitude of hundreds, approaching thousands in some cases. Hyperspectral image sensors, thus, continuously capture large amounts of data that are to be steadily transmitted to ground, given the small memory capacity the host systems typically provide. On the other hand, communication links between the sensors themselves and ground stations are quite limited in bandwidth, making an efficient compres-

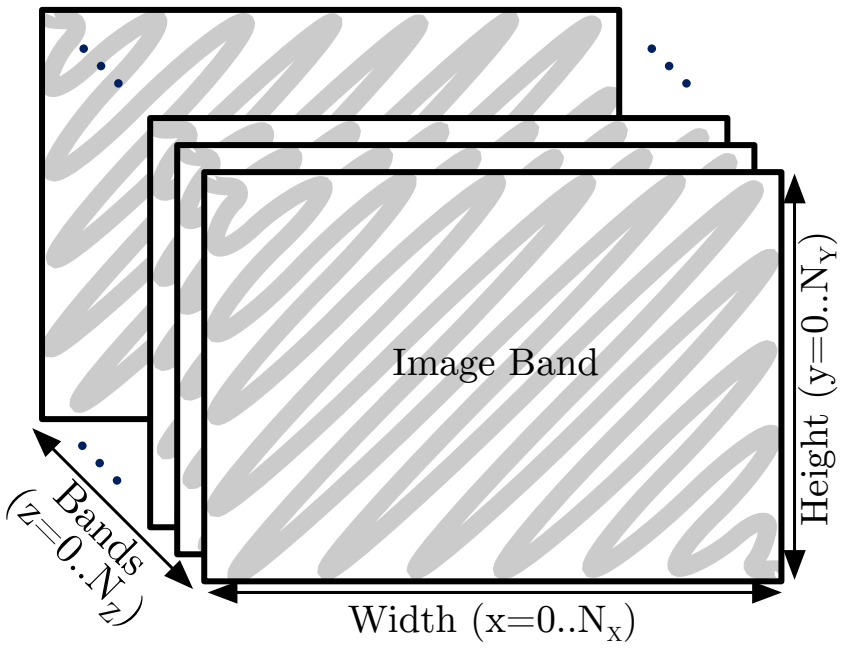


Figure 3.1: A hyperspectral image in Band Interleaved by Line (BIL) pixel ordering. In BIL ordering, all bands of each line are stored in contiguous space, so that both spatial and spectral information is accessed fairly easily.

sion algorithm a crucial element for these systems. Such an algorithm, in addition, must also be tailored to the limited computational capability of these environments of operation.

The Consultative Committee for Space Data Systems (CCSDS) is constituted by all major space agencies (NASA, ESA, JAXA, etc.) which cooperate towards the development of data handling standards for space applications. Several recommended standards have been developed by this committee over the years, with CCSDS 123.0-B-1 being only the latest for what regards lossless hyperspectral imagery compression. CCSDS-released algorithms and protocols are a relevant area in space-related image and data processing literature, with many examples of implementations (see [47], [48], [49]).

The CCSDS-123.0-B-1 standard compression algorithm has been recently released (May 2012) by the CCSDS with the purpose to provide a well documented and universally accepted approach to lossless compression of multispectral and hyperspectral images. The algorithm is a variation of the Fast Lossless algorithm [50], a 3D compression technique which is well known in current literature for its overall better results when compared to 2D approaches like LOCO-I [51] and 2D-CALIC [52] or other 3D algorithms such as LCL-3D [53] or SLSQ [54].

The CCSDS standard does not define any explicit implementation or throughput requirement. However, the algorithm is intended for on-line (real-time) streaming use and, as a consequence, for modern demanding missions, maximum data-rates and throughput may be a critical point. NASA HyspIRI mission, as an example, has a maximum data-rate of around 70 MSamples/sec (see [55]). None of the current hardware implementations of the Fast Lossless algorithm is capable of reaching such a high-throughput, with a huge gap in performance yet to be filled. To the best of our knowledge, moreover, no hardware implementation of the standard CCSDS algorithm exists in current literature. Software implementations, besides, are typically much slower than circuital implementations, generating the demand for a completely different approach.

In this chapter, we will see an algorithm variation which reduces performance criticalities of Fast Lossless and CCSDS-123.0 while preserving bit-trueness to the original CCSDS Standard, and that will be thus referred to, in the following, as “Fast CCSDS-123”. An implementation of this algorithm, based on a space-class Virtex5QV FPGA and an external DRAM, is also presented.

As we have mentioned in §1.5, FPGAs are available in Space-Grade packages, representing a viable alternative to Rad-Hard ASICs implementation and at a much lower cost. Furthermore, the performance benefit that is usually connected with ASIC implementations is much reduced when dealing with radiation hardened process, leading to FPGAs being a more and more established technology in this field of application (see [23]).

The implemented circuit is characterized by the architecture shown in Figure 3.2a. Following a scheme that is quite typical in compression algorithms, the main elements are a prediction stage, which produces the residuals that are to be actually coded, and the entropy coder. The two units are connected to an external memory for convenient storage of some coding statistics, as we will see in greater detail in the following sections. On the decompresser side, an entropy decoder obtains the prediction residuals that are fed to an inverse predictor, giving place to the original samples.

The proposed circuit is configurable by the user and can be adapted to the dimension of the hyperspectral image (up to 4096 columns, 4096 rows, and 4096 bands). It also implements both full and reduced prediction modes, allows the user to select the amount of preceding bands used for prediction (details in [56]), and can be configured in other details of the compression scheme. In terms of performance, the proposed circuit overcomes the state-of-the-art with respect to pixel throughput reaching 100.4 MSamples/s, while keeping a negligible footprint in terms of logic resource occupation (also when compared with current literature proposals).

The next section will sketch the workflow of the CCSDS-123.0-B-1 standardized algorithm. we will then see the standard-compliant implemented entropy coding scheme, focusing then on the modified algorithm which is at the core of this proposal. Afterwards we will analyse the architecture of the implemented compressor processor and some further architectural improvements implemented in the proposed circuit. To conclude we will see FPGA implementation results with respect to Xilinx Virtex 5 QV family Space Grade FPGAs.

3.2 CCSDS-123.0-B-1 prediction algorithm overview

In this section the prediction algorithm defined by the CCSDS standard and indicated with the 'CCSDS predictor' block in Figure 3.2a will be described.

The CCSDS-123.0-B-1 is a lossless compression algorithm. As a general

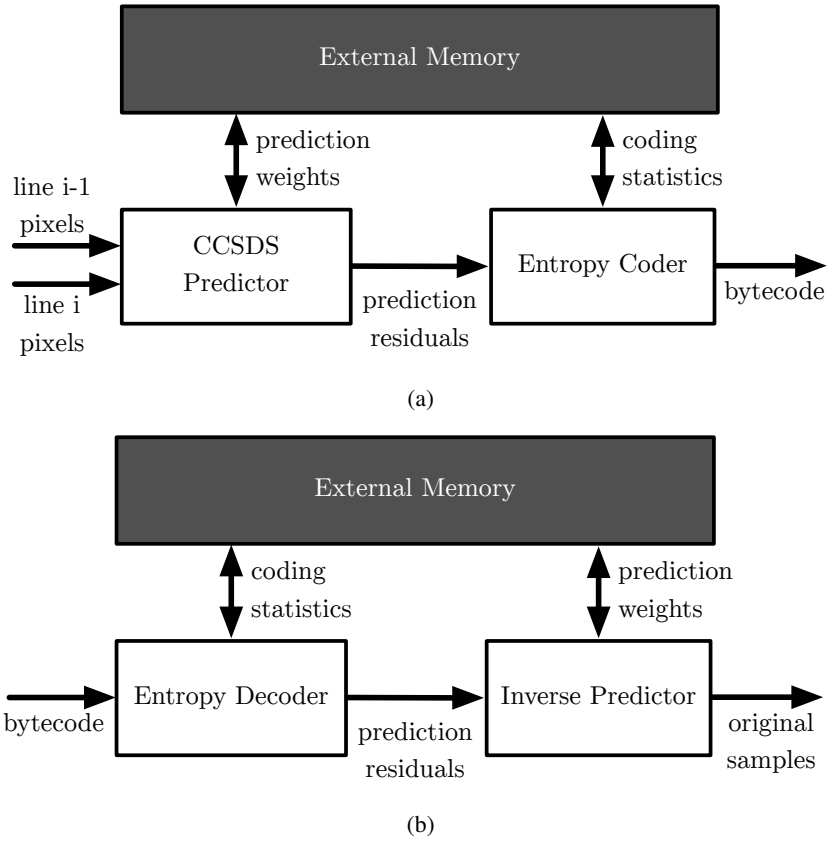


Figure 3.2: Overview of the CCSDS 123.0 compression architecture: a) Compressor chain. b) Decompresser chain.

indication, in satellite and airborne image processing, a lossless compression is preferred over lossy schemes because of both the high economical cost connected to the collection of this information and the need for post-processing at high fidelity and resolution. This is particularly true for satellite-borne sensing for target recognition or classification. The CCSDS-123.0-B-1 algorithm is a "causal lossless" algorithm, meaning it only makes use of the causal part of the image information (i.e. pixels of the image that have already been processed), based on the Fast Lossless compression technique.

The CCSDS algorithm is a low complexity and low memory impacting algorithm which provides good performance on both calibrated and uncalibrated

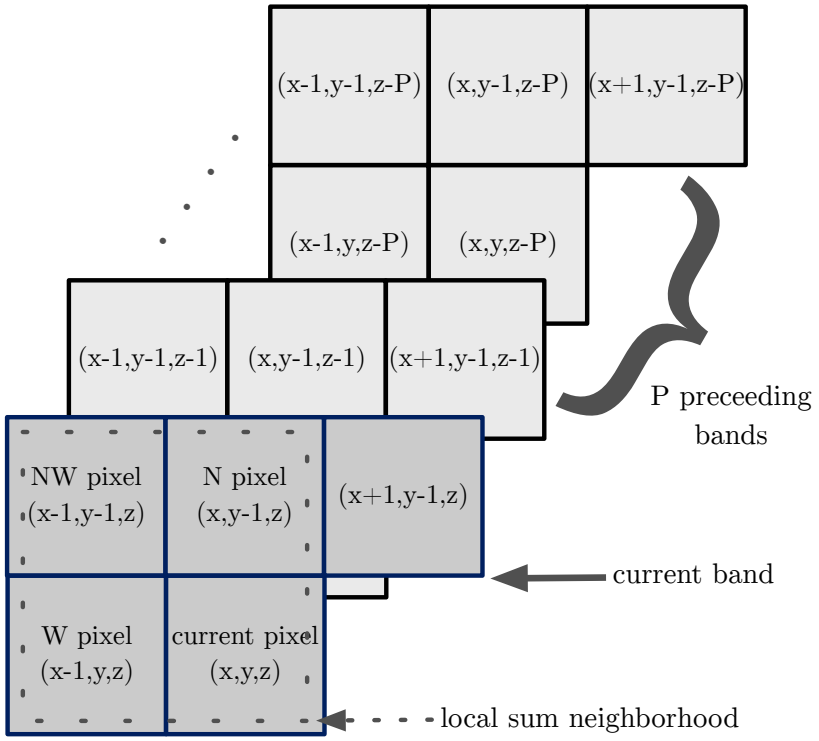


Figure 3.3: Prediction neighbourhood: in the presented implementation, P is a parameter which can be set in a range from 0 to 5.

images. We will discuss it in the remainder of this section, for brevity, without analysing all the variations supported by the reference standard but focusing on the working modes that apply to our implementation. Also for brevity, boundary and/or special conditions, which have been implemented faithfully to the recommended standard, will not be covered. For further information about the standard the reader is referred to [56].

The CCSDS algorithm predicts the value of each pixel by examining a three dimensional neighbourhood of the pixel itself (see Figure 3.3). The prediction is linear, which makes it less requiring in terms of computational power, and it is adaptive (i.e. the set of prediction weights is updated, at each pixel, on the basis of the prediction error for the preceding one). The prediction involves P preceding image bands, with a typical P value of 3 or 5. In our

implementation, *P* is configurable at runtime in a range from 0 to 5.

Let $s_{x,y,z}$ be a pixel value at spatial coordinates (x, y) and band z. We will refer to this as the current pixel or current sample (for the sake of simplicity, from now on every variable which is relative to the current sample will be indicated without subscripts, e.g: $s_{x,y,z}$ will be indicated simply by s). The CCSDS-123.0 algorithm steps are described in the following:

1. A "neighbour oriented local sum" is calculated from nearby pixels according to the following formula:

$$\sigma_{x,y,z} = (s_{x-1,y,z} + s_{x-1,y-1,z} + s_{x,y-1,z} + s_{x+1,y-1,z}) \tag{3.1}$$

   This local sum represents a first estimate of the pixel value in (x, y, z).

2. A set of "directional local differences" is obtained by subtracting the local sum from four times the sample values in the current ($s_{x,y,z}$) pixel and in the N ($s_{x,y-1,z}$), NW ($s_{x-1,y-1,z}$), and W ($s_{x-1,y,z}$) neighbours (see Figure 3.3). In addition, if the predictor operates in full prediction mode, "central" local differences are also calculated for pixel in spatial coordinates (x, y) for the *P* preceding bands. If $P = 5$, in other words, pixels C1 ($s_{x,y,z-1}$) to C5 ($s_{x,y,z-5}$) are also taken into account to form the local differences vector:

$$\boldsymbol{U} = \begin{pmatrix} u_N & u_{NW} & u_W & u_1 & u_2 & u_3 & u_4 & u_5 \end{pmatrix}.$$

   Otherwise, in reduced prediction mode, only local differences in current band are accounted, and the local differences vector reduces to:

$$\boldsymbol{U} = \begin{pmatrix} u_N & u_{NW} & u_W \end{pmatrix}.$$

   Typically, full prediction mode yields better results for calibrated imagery (see [57]).

3. The local differences vector $\boldsymbol{U}$ is multiplied with a weights vector $\boldsymbol{W}$, producing the singleton (scalar) value:

$$d = \boldsymbol{U} \cdot \boldsymbol{W} \tag{3.2}$$

The weights vector can be initialized either at a default or at a user-specified value set.

The obtained value, d, is an estimate of the difference between the current pixel value and the average value of its neighbours.

4. The "scaled predicted sample value" is calculated as:

$$\tilde{s} = clip\left(\left\lfloor \frac{mod_R^*[d + 2^{\Omega}(\sigma - 4s_{mid})]}{2^{\Omega+1}} \right\rfloor + 2s_{mid} + 1\right). \qquad (3.3)$$

In the equation, *clip* indicates the saturation between the input image dynamic range extrema and $\lfloor x \rfloor$ is the largest integer that is not greater than x. The symbol $mod_R^*[x]$ represents the two's complement integer that is congruent to x modulo 2^R (where R is a tunable "register size" which the standard mentions as needing to be set for overflow avoidance but without endorsing a specific recommendation); Ω is a parameter which controls the resolution of prediction weights; σ is the local sum; s_{mid} is the mid-range sample value representable according to the input image dynamic range. In other words, $\tilde{s}$ makes use of σ and d to predict the value of the current pixel.

5. Weights are updated on the basis of the scaled prediction error and of the weight update scaling exponent ρ, which determines at each pixel the entity of the weights update or, in other terms, the speed of the algorithm adaptation. At the initial stage the value of ρ is $\nu_{min} + D - \Omega$; at regular intervals, determined by the parameter t_{inc}, ρ is increased by one up to the maximum value $\nu_{max} + D - \Omega$. Also ν_{min} and ν_{max} are user controllable parameters. At each pixel, and for each component i of the weights vector, the updated weight $W_{i,next}$ is thus calculated as:

$$W_{i,next} = W_i + \left\lfloor 0.5(sgn[e] * 2^{-\rho} * U_i + 1) \right\rfloor \qquad (3.4)$$

where U_i is the corresponding element in the local differences vector.

6. The final step in prediction computation is the generation of the mapped prediction residual: being $\Delta = s - \lfloor \tilde{s}/2 \rfloor$ and $\theta = min\{\lfloor \tilde{s}/2 \rfloor, 2^{R-1} - \lfloor \tilde{s}/2 \rfloor\}$, the mapped prediction residual is defined as:

$$\delta = \begin{cases} |\Delta|+\theta & \text{if } |\Delta| > \theta \\ 2|\Delta| & \text{if } 0 \leq (-1)^{\tilde{s}}\Delta \leq \theta \\ 2|\Delta|-1 & \text{otherwise} \end{cases} \tag{3.5}$$

3.3 Entropy coding scheme

In this section the coding algorithm defined by the CCSDS standard and indicated with the 'Entropy Coder' block in Figure 3.2a is described.

The CCSDS-123.0 standard contemplates two possible entropy coding schemes, respectively defined as Block-Adaptive Entropy Coding and Sample-Adaptive Entropy Coding. The first is a GPO-2 based coding scheme also formalized by the CCSDS committee in the CCSDS-121.0 standard. This option is included to provide backwards compatibility with pre-existent space-grade implementations of the same algorithm. The Sample-Adaptive Entropy Coding scheme, which proves to be slightly better performing (see [57]), is implemented in the presented circuit. We now briefly introduce the coding algorithm, referring the reader to [56] for further information.

At each pixel the expected value of the mapped prediction residual is estimated. On the basis of this estimation, a variable length coding is assigned to the pixel itself.

The algorithm is thus composed by the following steps:

1. A counter named "Γ_z" is incremented at each pixel of band z and halved each time a certain configurable threshold is reached, producing a sawtooth profile.

2. An accumulator named "Σ_z" is calculated by repeated sums of the mapped residuals of band z and halved together with "Γ_z". The ratio of Σ_z and Γ_z represents the estimation of the mapped prediction residual at each pixel for band z.

3. The block labelled k_z calculates the largest positive integer value for which the condition

$$\Gamma_z(t)2^{k_z(t)} \leq \Sigma_z(t) + \left\lfloor \frac{49}{2^7}\Gamma_z(t) \right\rfloor \tag{3.6}$$

   is verified.

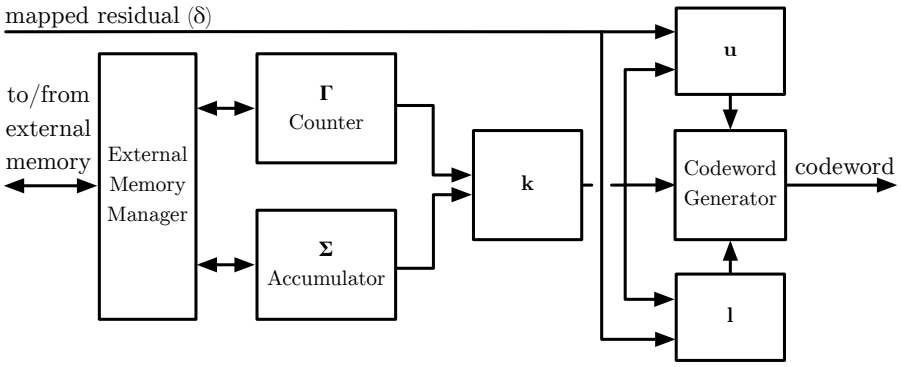


Figure 3.4: Architecture of the implemented entropy coder

4. The mapped residual δ is split into its most significant part "u", obtained by k right shifts, and its least significant part "l", which corresponds to the k least significant bits of the residual itself.

5. If $u < U_{max}$, where U_{max} is a tunable parameter, the codeword for δ is composed by u zeros, followed by a one, and by the binary representation of l. Otherwise, the codeword is composed by U_{max} zeros, followed by the binary representation of δ.

Figure 3.4 shows the architecture of the realized entropy coder. The blocks labeled as "Γ_z" and "Σ_z" calculate the band compression statistics and are connected to an external memory manager. The sample adaptive encoding algorithm, in fact, maintains separate coding statistics for each band which, as a consequence of the BIL encoding chosen for our implementation, results in the need for a certain memory amount, as we will se in Section 3.5. These coding statistics, similarly to the prediction weights, can be conveniently stored in an external DRAM to reduce leverage on in-device memory.

In §3.7 we will compare the output of the proposed circuit with the CCSDS standard implementation showing the bit-trueness of the proposed circuit with respect to the standard coder.

3.4 Proposed prediction algorithm modification

Figure 3.5 shows the computational flow of the CCSDS-123.0 standard algorithm. The dashed arrow highlights the critical part of the algorithm for circuital performance: the weight update feedback creates a strict data dependency which, as known in literature, prevents the insertion of pipelining reg-

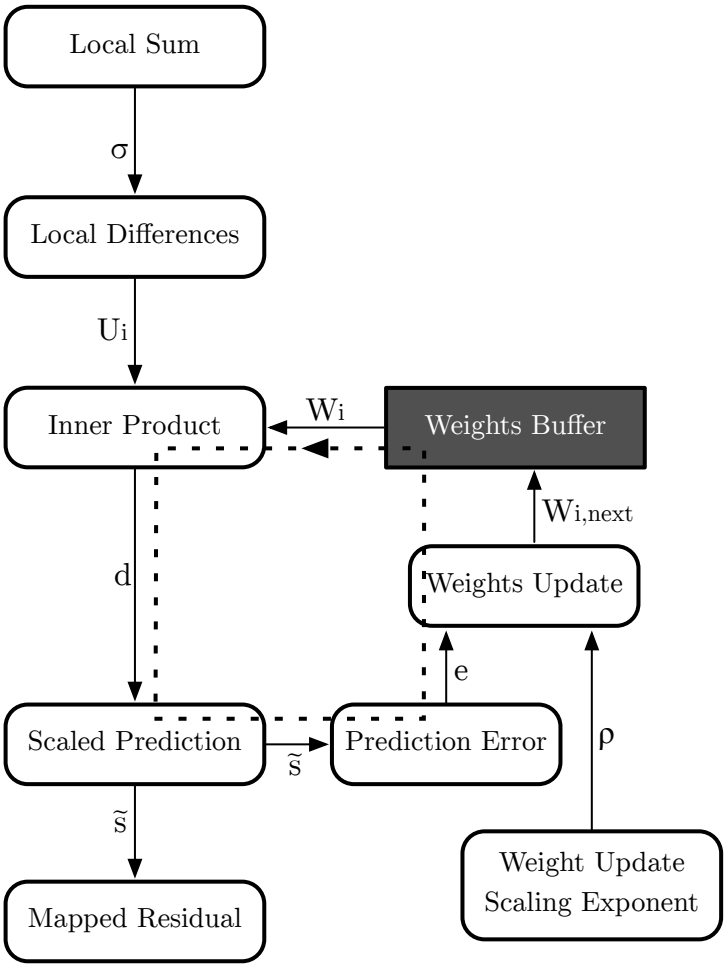


Figure 3.5: Standard CCSDS-123.0 Algorithm Flowchart

.

isters thus creating a combinatorial loop which involves high-delay circuitry. The loop is in fact composed by the following algorithm phases:

- Inner Product, which requires 8 products and a sum tree between the same, calculating (3.2).
- Scaled Prediction Calculation, which involves the computation of (3.3).
- Prediction Error Evaluation, which requires a subtraction.
- Weights Update, which corresponds to (3.4).

Any circuital implementation of this scheme would incur in such high delay and, thus, would not be able to perform at high-throughput.

The main contribution of this work to the state-of-the-art consists in an algorithmic modification of the standard CCSDS-123.0 algorithm which we named the "Fast CCSDS-123" algorithm. The flow-chart diagram of Figure 3.6 shows the main differences between the two:

- The Weight Update phase is replaced with a redundant computation of "Lookahead Weights" as the result of the following analysis of (3.4). The dependency of $W_{i,next} = f(W_i, e, \rho, U)$ on data at the current iteration i is, in fact, broken down as follows:

  the equation
  $$W_{i,next} = f(W_i, e, \rho, U) \tag{3.7}$$
  can be rewritten as
  $$W_{i,next} = f(e, \boldsymbol{\Psi}), \tag{3.8}$$
  where
  $$\boldsymbol{\Psi} = \boldsymbol{\Psi}(W_i, \rho, U))) \tag{3.9}$$
  is the "Lookahead Weight" set.

  In other terms, the dependency on e is isolated from the calculation. Furthermore, this dependency is of reduced complexity, being the updated weights actually only dependent on $sgn[e]$. As a consequence, it is possible to evaluate the two possible values of $W_{i,next}$ for both the values of $sgn[e]$, making the set of the "Look Ahead Weights" actually composed of just two elements, Ψ^+ and Ψ^-.

- To further benefit from this improvement, the Inner Product phase is also duplicated in a similar lookahead scheme. This gives place to two possible inner product values (denoted as d^+ and d^- in the diagram) among which the correct one must be selected on the basis of the prediction error, as the "Select" block shows.

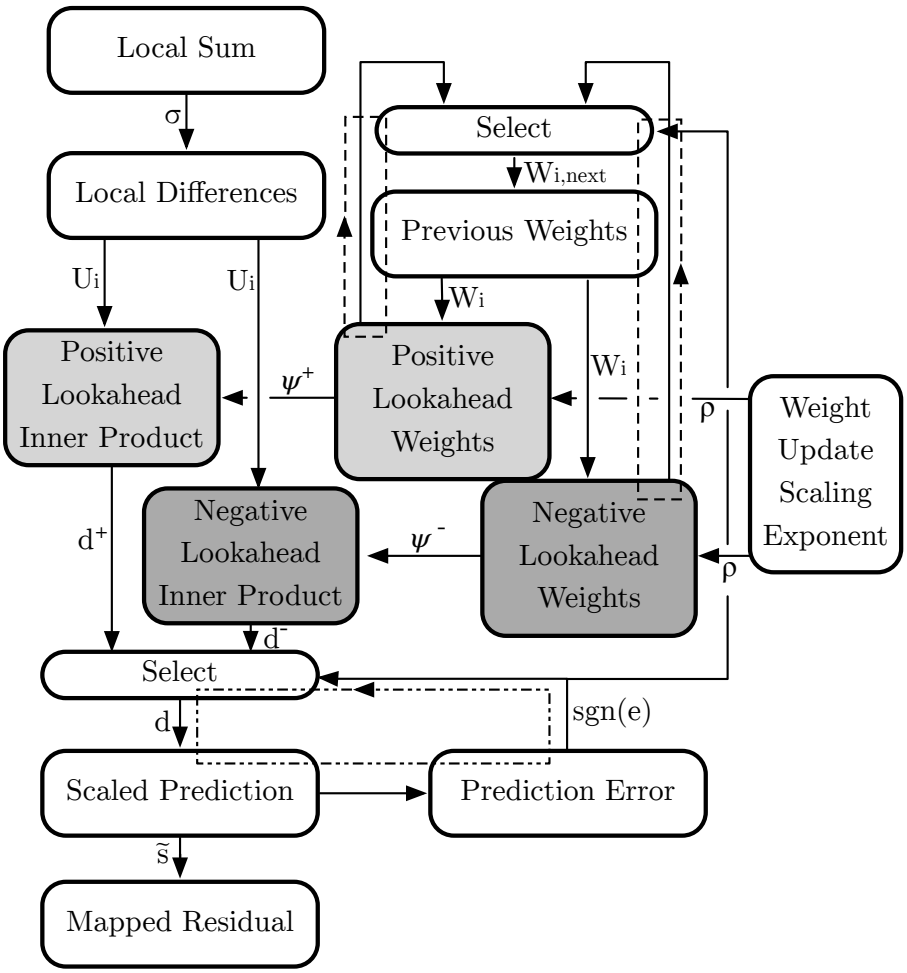


Figure 3.6: Fast CCSDS-123 Lossless Compression Algorithm Flowchart.

This different algorithm, as the figure shows, reduces dramatically the length of the feedback loop, actually splitting it into two shorter paths. The first path is the one that involves the computation of Scaled Prediction and Prediction Error (dashed and dotted, in the figure), which clearly implies a shorter delay path with respect to the one in the standard algorithm; the second path is the one that involves the computation of the lookahead weights and their selection (dashed, in the figure). In particular, the computation of the lookahead weights, being free from the part of the calculation that is connected with the evaluation of $sgn[e]$, is also less impacting than the original Weight Update phase of the standard algorithm.

3.5 Realized processor architecture

As mentioned in Section 3.1 and shown in Figure 3.2a, the compression processor is divided in two main units: the predictor, which implements the modified algorithm discussed in Section 3.4, and the entropy coder, which realizes the algorithm exposed in Section 3.3. Both elements are connected to an external memory for the storage, respectively, of the prediction weights **W** and of the coding statistics $\{\Gamma, \Sigma\}$. The total external memory occupation, calculated on the worst case scenario of an image with 4096 bands and P=5, which is the maximum allowed by the proposed architecture, is obtained as follows:

$$\begin{aligned}\text{Weights: } &(max\, number\, of\, bands) * \\ &* ((max\, P) + 3) * (weights\, resolution) = \\ &= 4096 * (5 + 3) * 17 = 544 \text{ Kb} \end{aligned} \quad (3.10)$$

$$\begin{aligned}\text{Coding statistics: } &(max\, number\, of\, bands) * \\ &* (counter\, size + accumulator\, size) = \\ &= 4096 * (20 + 36) = 224 \text{ Kb} \end{aligned} \quad (3.11)$$

The memory needed to store the local differences for the *P* preceding bands is stored, conversely, in the FPGA Block RAMs to improve circuital perfor-

mance. This memory accounts up to:

$$(max\, number\, of\, bands) * (max\, P) * (local\ differences\, resolution) = \\ = 4096 * 5 * 19 = 380 \text{ Kb} \quad (3.12)$$

On the other hand, the external memory allocation of the prediction weights and coding statistics does not impact performance thanks to the adoption of a Ping-Pong scheme: during the processing of each band "i", a "Shadow Register" pre-fetches weights for the next band "i+1". When the processing of band "i+1" begins, the registers are switched, so that the now-become shadow register can store to the external memory the updated weights vector for band "i": the same scheme repeats for the compression statistics. This actually poses a lower bound on the image "Nx" size (the number of columns), which cannot be less than the number of clock cycles necessary to complete the store/fetch operation by the memory manager. This limit has been set, accounting for an appropriate safety margin, to 100 pixels.

As Figure 3.2a shows, the predictor receives input from two subsequent lines in Band-Interleaved-by-Line (BIL) format. It then produces the mapped prediction residuals which are sent to the entropy coder. The entropy coder produces the compressed bytecode which is prepared to be sent over a serial connection or to be byte-packed.

The overall architecture of the predictor has been pipelined except for the two combinatorial "loops" seen in §3.4. As a consequence, the critical path resides in the longest of these loops (namely the one involving the computation of the lookahead weights) limiting the performance to what will be shown in §3.7.

3.6 Further architectural improvements

The CCSDS 123.0-B-1 algorithm, defining in real arithmetic the compression process, is sometimes muddled in its formulation of the various steps. As a consequence, during circuital implementation, some passages can be completely reformulated, giving place to conditions and formulas that better suit the programmable hardware architecture while improving, at the same time, performance and logic resource occupation. In this section we will, thus, consider some further reformulations we have carried out during the development of the processor.

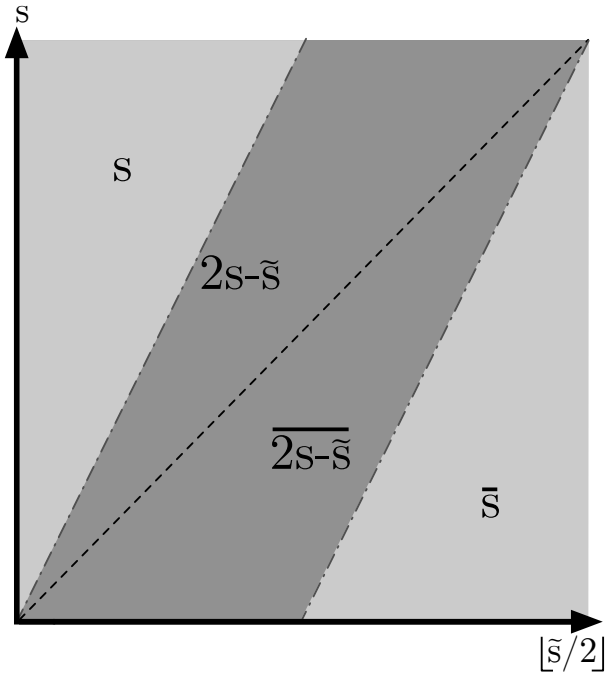


Figure 3.7: Geometrical representation of the optimized mapped residual formulation. The values reported on the plane represent the mapped residual values obtained as a function of s and $\lfloor \tilde{s}/2 \rfloor$.

- Firstly, the formulation of the Scaled Prediction calculation given in (3.3) can be rewritten exploiting the features of register arithmetic. By choosing a register size ("R") large enough to avoid overflows, since in our case s_{mid} is an integer value, the equation can be simplified to:

$$\tilde{s} = clip(\lfloor (d/2^{\Omega} + \sigma + 2)/2 \rfloor). \tag{3.13}$$

  Furthermore, the floor operation is carried out by the register logic itself and the clipping is simply a saturation to the image dynamic range. The scaled prediction, in this way, is implemented by using: a shifter, an adder, and a saturator (implemented with a multiplexer and a comparator).

- The calculation of the mapped residual (3.5) has also been reformulated.

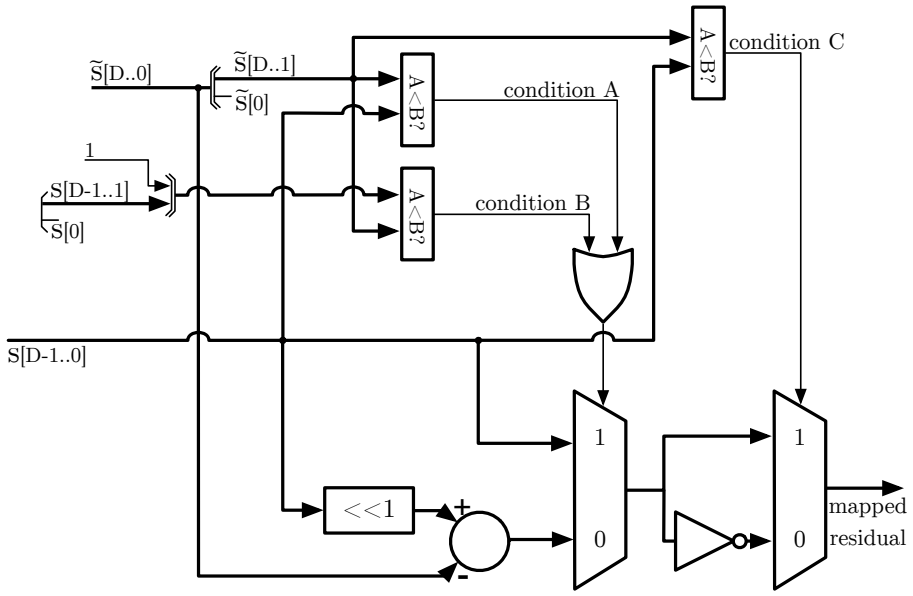


Figure 3.8: Implementation of the optimized mapped residual calculator

The starting point of this reformulation is the fact that, in two's complement arithmetic, a negative number is represented by the bitwise NOT of the unsigned representation of its opposite augmented by 1. As a consequence, the second and third cases in (3.5) can be combined and rewritten as follows:

$$r = \begin{cases} s & \text{if } \lfloor \tilde{s}/2 \rfloor < s/2 \text{ or} \\ & \lfloor \tilde{s}/2 \rfloor > \{1, \lfloor s/2 \rfloor\} \\ 2s - \tilde{s} & \text{otherwise} \end{cases} \tag{3.14}$$

$$\delta = \begin{cases} \text{r} & \text{if } \lfloor \tilde{s}/2 \rfloor < s \\ \text{not r} & \text{otherwise} \end{cases} \tag{3.15}$$

where $\{A, B\}$ indicates the concatenation of the bit sequence A with B.

This formula can be represented by the 2D map of Figure 3.7 and implemented by the circuit shown in Figure 3.8. The shown circuit has a

Table 3.1: Proposed Circuit (Fast CCSDS) Resource Allocation and Performance on a Space Grade Xilinx Virtex 5Q SX50T FPGAs.

	Slices (amount)	BRAM (kB)	Max Frequency (MHz)
Occupied	2800	414	100.4
Available	20480	10728	–
Percentage	13%	3.86%	–

combinatorial delay path composed by a comparator, an OR gate, two 2-to-1 multiplexers, and a NOT gate. Such rewriting, thus, yields better performance when compared to the delay path that would result from the original standard formulation, which would include at least a subtracter and two comparators, as (3.5) shows.

- The inner product unit has also been optimized by implementing the sum of the 8 products with a fully balanced tree in order to reduce summing stages to $\log_2(8) = 3$. In addition, register retiming has also been carried out inside the tree, dividing the 4 levels (one multiplication level and the three summing stages) and reducing total combinatorial delay.

3.7 Implementation results and comparison with state-of-the-art

Table 3.1 shows implementation statistics for the deployment of the realized circuit on the Virtex5QV SX50T FPGA, which is a small sized FPGA manufactured by Xilinx. The Virtex5QV is a space grade, radiation hardened family of SRAM-based FPGAs which exhibit very high levels of radiation tolerance (SEU latch-up immunity > 100 Mev-cm^2/mg - see also [58]). Resource occupation percentages show a very small footprint for major statistics even on such a small device. The maximum clock frequency obtained is of 100.4 MHz, allowing for a maximum throughput of 100.4 Msamples per second.

While no hardware implementation of the standard CCSDS-123.0-B-1 algorithm is available in current literature for performance comparison, Table 3.2 shows a throughput comparison with some implementations of the original Fast Lossless compression algorithm, which is in fact very similar to the CCSDS-standardized algorithm and, thus, to our algorithm. The considered

Table 3.2: Performance comparison with current literature implementations: proposed algorithm is shown in bold and, together with comparison implementations realized during this thesis work, is denoted by the acronym "t.t." (this thesis)

Proposal	Year	Throughput (MSamp/s)	Implementation (Technology)
Fast CCSDS-123	*(t.t.)*	100.4	Xilinx Virtex5QV
CCSDS-123.0	*(t.t.)*	55.4	Xilinx Virtex5QV
Fast CCSDS-123	*(t.t.)*	62.9	Xilinx Virtex4†
Keymeulen, al.[61]	2014	44.9	nVidia GTX 580 GPU
Schmidt, al.[60]	2013	29.1	Xilinx Virtex4 (SoC)‡
Schmidt, al.[59]	2012	1.32	Xilinx Virtex4 (SoC)‡

(†) *Virtex 4 FX60*

(‡) *System-on-Chip deployed on a Virtex 4 FX60*

published works include [59, 60], which enforce a System-on-Chip (SoC) approach to implement Radiation Hardening by software, and [61], which represents a GPU programming approach to the algorithm implementation. The table shows throughput results for the implementations of both the original CCSDS-123.0 and the proposed Fast CCSDS-123 algorithms.

As Table 3.2 shows, the presented work overcomes all these implementations resulting in being the only implementation capable of exceeding, and with a substantial margin, 50 MSamples/sec; our implementation of the original algorithm (which makes use of the circuit-oriented optimizations introduced in Section 3.6) also exceeds 50 MSamples, but the performance gain connected with the adoption of the modified algorithm is of a factor around 2:1.

With respect to resource allocation, a comparison with FPGA-based literature proposals is shown in Table 3.3: the table shows how our proposal is also less impacting in terms of device occupation with respect to the current state-of-the-art.

The resulting architecture has been cross-tested with the CCSDS-released software compressing tool on a set of hyperspectral images from spectrometers such as AVIRIS, MODIS and CRISM showing 100% bit-trueness with the standard algorithm, as table 3.4 clarifies.

Table 3.3: Comparison of Resource Occupation on a Xilinx Virtex4 FX60 FPGA with state-of-the-art literature FPGA implementations of Fast Lossless

Proposal	Slices FFs Amount(%)	4 Input LUTs Amount(%)
This Thesis (Fast CCSDS)	1959 (3%)	7116 (14%)
Schmidt, al.[59] (Fast Lossless)	8545 (17%)	7530 (15%)
Schmidt, al.[60] (Fast Lossless)	15736 (31%)	20969 (42%)

Table 3.4: CCSDS toolkit cross-test results: the error metric indicates how many pixels, in the images processed by the presented implementation, differ from the ones processed by the standard CCSDS toolkit. The presented image sets are part of AVIRIS, MODIS and CRISM databases

Image set	Pixel count (amount)	Compression Rate (%)	Errors (pixels)
AVIRIS	77987840	61.1	0
MODIS	54972400	52.4	0
CRISM	12787200	75.0	0

Table 3.5, to conclude, shows the difference in power dissipation between a state-of-the-art implementation of the standard CCSDS-123 algorithm realized during this same thesis work and the presented, faster implementation when run at the same clock frequency of 50 MHz (which is near the maximum allowed frequency of the CCSDS-123 standard algorithm implementa-

Table 3.5: Power Dissipation of the realized circuits implementing both the standard CCSDS 123 algorithm and the proposed Fast CCSDS 123 algorithm. Results are shown for an implementation on a Xilinx Virtex5QV SX50T FPGA

Implemented Algorithm	Static Power (mW)	Dynamic Power (mW/MHz)
Fast CCSDS-123	837	3.92
Standard CCSDS-123.0	836	3.36
Keymeulen, al.[61]	n/a	n/a
Schmidt, al.[60]	n/a	n/a
Schmidt, al.[59]	n/a	n/a

tion). The results show an increase in dynamic power dissipation due to the intrinsic redundancy of the lookahead phase.

Conclusions

In this thesis, we have examined the role of digital Field Programmable Gate Arrays circuits in Real Time Video Processing (RTVP) applications. Starting from Chapter 1 we have seen how this technology is rapidly emerging as one of the best candidates for carrying out this task with a good balance between cost, performance level, and flexibility. In particular, we have explored typical intrinsic features of RTVP algorithms and matched them with the structure and capabilities of these devices. We also have examined typical problems and challenges connected with the use of this technology in the solution of RTVP problems, like design testing, and reviewed the most important strategies to overcome these issues.

We have then discussed two practical applications of this topic: in particular, Chapter 2 has presented a topical literature problem, namely computer vision algorithms, and one of its most notable solutions, consisting in the Compact Descriptors for Visual Search (CDVS) proposed standard algorithm. The implementation in either hardware or software of this algorithm is a very challenging and discussed topic, and this dissertation has show an ample set of design solutions. In particular, some of these solutions have been considered by the CDVS committee for inclusion in the standard draft of the algorithm, marking an important contribution to the state-of-the-art. All of the circuits proposed in this chapter allow for the accomplishment of the primary task of the CDVS algorithm: enforcing real-time feature extraction on mobile, low power devices.

As a concluding remark, it has to be noted that, being CDVS still a "work in progress" standard draft, the implemented circuits are themselves open to changes: notably, constant values (e.g: thresholds, characteristics of the Gaussian filters, etc.) as well as other implementation details have been left as "parameters" in the circuital implementation to let the implementation able to adapt easily to minor changes in the final definition of the CDVS algorithm.

In Chapter 3, to conclude, we have analysed an example of space applcaions, a special application field with stricts technology requirements in which FPGAs excel, being nowadays the best option in terms of speed, flexibility and reliability. We have examined an algorithm for the compression of hyperspectral images, developed during this Ph.D. course, that is bit-true with a well known standard algorithm by the Consultative Committee for Space Data Systems (CCSDS). Our proposal, circumventing an inherent bottleneck of the CCSDS proposed algorithm, allows higher throughput, overcoming any state-of-the-art implementations of the original algorithm and making it suitable for modern, demanding space missions with strict throughput requirements.

Publications

- G.Lopez, N.Petra, D.De Caro, E.Napoli, "An FFT-based Coprocessor for Real-time Video Processing ", *Proceedings of GE 2012*.
- G.Lopez, E.Napoli, A.G.M.Strollo, "Towards a Frequency Domain Processor for Real-Time SIFT-based Filtering", *Applications Pervading Industry, Environment and Society, Lecture Notes in Electrical Engineering*, (in press).
- E.Napoli, G.Lopez, A.G.M.Strollo, "Noise with Colored Power Spectrum Derived from a Single bit White Noise Input", *20th IMEKO TC4 International Symposium and 18th International Workshop on ADC Modelling and Testing*, 2014.
- G.Lopez, E.Napoli, A.G.M.Strollo, "A Frequency Domain Processor for Real-Time CDVS Keypoints Extraction", *SITIS 2014 - The 10th International Conference on Signal Image Technology and Internet Based Systems*, 2014.
- G.Lopez, E.Napoli, A.G.M.Strollo, "FPGA Implementation of the CCSDS-123.0-B-1 Lossless Hyperspectral Image Compression Algorithm Prediction Stage", *LASCAS 2015 - 6th IEEE Latin American Symposium on Circuits and Systems*, 2015.
- G.Lopez, E.Napoli, A.G.M.Strollo, "An FPGA Processor for Real-Time, Fixed-Point Refinement of CDVS Keypoints", ISCAS 2015 - International Symposium on Circuits and Systems, 2015 (accepted).

- G.Lopez, E.Napoli, A.G.M.Strollo, D.De Caro, N.Petra, “Definition and Hardware Implementation of a High Throughput Lossless Image Compression Algorithm Bit-true Compatible with CCSDS-123.0”, IEEE Transactions on Circuits and Systems for Video Technology (submitted for publication)-

Acknowledgements

At the end of my Ph.D. course, I'd like to thank sincerely all the personnel of the Electronics Engineering research group: in particular my thoughts go to the greatly skilled young researchers who literally fight daily, hoping for better days for Research, University, Education. Like them, I strongly believe in these values, and hope that they will return to be the pillars of our society, like they were in each of the Golden Ages of mankind.

I would like to thank Davide de Caro, Nicola Petra, Michele Riccio and Luca Maresca who helped me more than once in finding my way; a very special thanks goes to Michele de Martino who has been a great lab (and laughs) mate; and then, last but not least, I would like to thank my tutor Ettore Napoli, who gave me the chance to live this great self-improving experience.
I hope our roads will cross again.

Bibliography

[1] D. G. Bailey, *Design for embedded image processing on FPGAs*. John Wiley and Sons (Asia) Pte Ltd, 2011.

[2] H. S. Neoh and A. Hazanchuk, "Adaptive edge detection for real-time video processing using FPGAs," *Global Signal Processing*, vol. 7, no. 3, pp. 2–3, 2004.

[3] P. M. Athanas and A. L. Abbott, "Real-time image processing on a custom computing platform," *Computer*, vol. 28, no. 2, pp. 16–25, 1995.

[4] S. Jin, J. Cho, X. Dai Pham, K. M. Lee, S.-K. Park, M. Kim, and J. W. Jeon, "FPGA design and implementation of a real-time stereo vision system," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 20, no. 1, pp. 15–26, 2010.

[5] J. Woodfill and B. Von Herzen, "Real-time stereo vision on the parts reconfigurable computer," in *Field-Programmable Custom Computing Machines, 1997. Proceedings., The 5th Annual IEEE Symposium on*. IEEE, 1997, pp. 201–210.

[6] I. S. Uzun, A. Amira, and A. Bouridane, "FPGA implementations of Fast Fourier Transforms for real-time signal and image processing," in *Vision, Image and Signal Processing, IEE Proceedings-*, vol. 152, no. 3. IET, 2005, pp. 283–296.

[7] M. C. Herbordt, T. VanCourt, Y. Gu, B. Sukhwani, A. Conti, J. Model, and D. DiSabello, "Achieving high performance with FPGA-based computing," *Computer*, vol. 40, no. 3, p. 50, 2007.

[8] P. Ngan, "The development of a visual language for image processing applications," Ph.D. dissertation, Massey

University, New Zealand, 1992. [Online]. Available: http://mro.massey.ac.nz/bitstream/handle/10179/3007/02_whole.pdf

[9] K. Compton and S. Hauck, "Reconfigurable computing: a survey of systems and software," *ACM Computing Surveys (csuR)*, vol. 34, no. 2, pp. 171–210, 2002.

[10] T. J. Todman, G. A. Constantinides, S. J. Wilton, O. Mencer, W. Luk, and P. Y. Cheung, "Reconfigurable computing: architectures and design methods," *IEE Proceedings-Computers and Digital Techniques*, vol. 152, no. 2, pp. 193–207, 2005.

[11] "Zynq-7000 all programmable SoC" - Xilinx, Inc. [Online]. Available: http://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html

[12] "Stratix 10 FPGA overview" - Altera Corporation USA. [Online]. Available: https://www.altera.com/products/fpga/stratix-series/stratix-10/overview.html

[13] "Cyclone V SoCs overview" - Altera Corporation USA. [Online]. Available: https://www.altera.com/products/soc/portfolio/cyclone-v-soc/overview.html

[14] R. B. Foist, C. S. Grecu, A. Ivanov, and R. Turner, "An FPGA design project: creating a PowerPC subsystem plus user logic," *Education, IEEE Transactions on*, vol. 51, no. 3, pp. 312–318, 2008.

[15] N. Ohba and K. Takano, "An SoC design methodology using FPGAs and embedded microprocessors," in *Proceedings of the 41st annual Design Automation Conference*. ACM, 2004, pp. 747–752.

[16] D. Etiemble, S. Bouaziz, and L. Lacassagne, "Customizing 16-bit floating point instructions on a Nios II processor for FPGA image and media processing," *Embedded Systems for Real-Time Multimedia*, pp. 61–66, 2005.

[17] J. Cho, H. Chang, and W. Sung, "An FPGA based SIMD processor with a vector memory unit," in *Circuits and Systems, 2006. ISCAS 2006. Proceedings. 2006 IEEE International Symposium on*. IEEE, 2006, pp. 4–pp.

[18] N. Kehtarnavaz and M. Gamadia, "Real-time image and video processing: from research to reality," *Synthesis Lectures on Image, Video & Multimedia Processing*, vol. 2, no. 1, pp. 1–108, 2006.

[19] "MPEG-4 reference software implementation" - the Moving Picture Experts Group. [Online]. Available: http://mpeg.chiariglione.org/standards/mpeg-4/reference-software

[20] P. S. Graham, "Logical hardware debuggers for FPGA-based systems," Ph.D. dissertation, Brigham Young University, 2001.

[21] J. Tombs, M. A. Echanóve, F. Munoz, V. Baena, A. Torralba, A. Fernandez-León, and F. Tortosa, "The implementation of a FPGA hardware debugger system with minimal system overhead," in *Field Programmable Logic and Application*. Springer, 2004, pp. 1062–1066.

[22] "HDL Verifier" - the MathWorks, Inc. [Online]. Available: http://it.mathworks.com/products/hdl-verifier/

[23] "Curiosity and NASA's mission to Mars: a case for small business" - National Aeronautics and Space Administration. [Online]. Available: http://osbp.nasa.gov/docs/MARS OSB_CS55_FINAL_LO=TAGGED.pdf

[24] D. Lowe, "Object recognition from local scale-invariant features," *International Conference on Computer Vision, 1999*, pp. 1150–1157, 1999.

[25] D. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, vol. 60, no. 2, pp. 91–110, 2004.

[26] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "Speeded-up robust features (surf)," *Computer vision and image understanding*, vol. 110, no. 3, pp. 346–359, 2008.

[27] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 1. IEEE, 2005, pp. 886–893.

[28] Y. Ke and R. Sukthankar, "PCA-SIFT: a more distinctive representation for local image descriptors," in *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, vol. 2. IEEE, 2004, pp. II–506.

[29] V. Chandrasekhar, G. Takacs, D. Chen, S. Tsai, R. Grzeszczuk, and B. Girod, "CHoG: Compressed Histogram of Gradients a low bit-rate feature descriptor," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE, 2009, pp. 2504–2511.

[30] L. Yao, H. Feng, Y. Zhu, Z. Jiang, D. Zhao, and W. Feng, "An architecture of optimised SIFT feature detection for an FPGA implementation of an image matcher," in *Field-Programmable Technology, 2009. FPT 2009. International Conference on*. IEEE, 2009, pp. 30–37.

[31] J. Qiu, T. Huang, and T. Ikenaga, "A FPGA-based dual-pixel processing pipelined hardware accelerator for feature point detection part in SIFT," in *INC, IMS and IDC, 2009. NCM'09. Fifth International Joint Conference on*. IEEE, 2009, pp. 1668–1674.

[32] F.-C. Huang, S.-Y. Huang, J.-W. Ker, and Y.-C. Chen, "High-performance SIFT hardware accelerator for real-time image feature extraction," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 22, no. 3, pp. 340–351, 2012.

[33] T. Lindeberg, "Scale-space theory: A basic tool for analyzing structures at different scales," *Journal of applied statistics*, vol. 21, no. 1-2, pp. 225–270, 1994.

[34] D. Pau, A. Bruna, E. Napoli, and G. Lopez. (2014, Jul. 17) "Method and apparatus for computing image pyramids and related computer program product". US Patent App. 14/156,327. [Online]. Available: http://www.google.com/patents/US20140198995

[35] H. R. Wilson and S. C. Giese, "Threshold visibility of frequency gradient patterns," *Vision Research*, vol. 17, no. 10, pp. 1177 – 1190, 1977. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0042698977901523

[36] D. Marr and E. Hildreth, "Theory of edge detection," *Proceedings of the Royal Society of London. Series B. Biological Sciences*, vol. 207, no. 1167, pp. 187–217, 1980.

[37] K. Mikolajczyk and C. Schmid, "A performance evaluation of local descriptors," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 27, no. 10, pp. 1615–1630, 2005.

[38] B. Hunt, "Minimizing the computation time for using the technique of sectioning for digital filtering of pictures," *Computers, IEEE Transactions on*, vol. C-21, no. 11, pp. 1219–1222, Nov 1972.

[39] H. Guggenheimer, *Differential Geometry: McGraw-Hill [1963].*, ser. McGraw-Hill series in higher mathematics. University Microfilms, 1963.

[40] E. H. Bareiss, "Sylvester's identity and multistep integer-preserving Gaussian elimination," *Mathematics of Computation*, vol. 22, no. 103, pp. 565–578, Jul. 1968.

[41] S. Peng and S. Sedukhin, "Array processors design for division-free linear system solving." *Comput. J.*, vol. 39, no. 8, pp. 713–722, 1996.

[42] R. Martinez-Alonso, K. Mino, and D. Torres-Lucio, "Array processors designed with VHDL for solution of Linear Equation Systems implemented in a FPGA," in *Electronics, Robotics and Automotive Mechanics Conference (CERMA), 2010*, Sept 2010, pp. 731–736.

[43] S. Peng and S. Sedukhin, "Parallel algorithm and architectures for two-step division-free gaussian elimination," in *Algorithms and Architectures for Parallel Processing, 1997. ICAPP 97., 1997 3rd International Conference on*, Dec 1997, pp. 489–502.

[44] A. G. Strollo, D. De Caro, and N. Petra, "Elementary functions hardware implementation using constrained piecewise-polynomial approximations," *Computers, IEEE Transactions on*, vol. 60, no. 3, pp. 418–432, 2011.

[45] D. De Caro, N. Petra, and A. G. Strollo, "High-performance special function unit for programmable 3-d graphics processors," *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. 56, no. 9, pp. 1968–1978, 2009.

[46] R. Martinez-Alonso, K. Mino, and D. Torres-Lucio, "Array processors designed with VHDL for solution of linear equation systems implemented in a FPGA," in *Electronics, Robotics and Automotive Mechanics Conference (CERMA), 2010*, Sept 2010, pp. 731–736.

[47] H. Jiang and X. Zhou, "A VLSI implementation of CCSDS for meteorology image lossless compression," in *Advances in the Astronautical Sciences*, vol. 117, 2004, pp. 183–188.

[48] L. Li, G. Zhou, B. Fiethe, H. Michalik, and B. Osterloh, "Efficient implementation of the CCSDS 122.0-B-1 compression standard on a space-qualified Field Programmable Gate Array," *Journal of Applied Remote Sensing*, vol. 7, no. 1, 2013.

[49] E. Della Sala, E. Sciagura, D. De Caro, A. Caravella, P. Longobardi, P. Zicari, V. Garofalo, G. Capuano, P. Corsonello, E. Napoli, and G. Strollo, "High rate data down link," in *European Space Agency, (Special Publication)*, no. 647 SP, 2007, pp. 217–222.

[50] M. A. Klimesh, "Low-complexity lossless compression of hyperspectral imagery via adaptive filtering," *The Interplanetary Network Progress Report*, vol. 42-163, pp. 1–10, 2005.

[51] M. Weinberger, G. Seroussi, and G. Sapiro, "The LOCO-I lossless image compression algorithm: principles and standardization into JPEG-LS," *IEEE Transactions on Image Processing*, vol. 9, no. 8, pp. 1309–1324, Aug 2000.

[52] X. Wu and N. Memon, "Context-based, adaptive, lossless image coding," *IEEE Transactions on Communications*, vol. 45, no. 4, pp. 437–444, Apr 1997.

[53] S. Hunt and L. Rodriguez, "Fast piecewise linear predictors for lossless compression of hyperspectral imagery," in *Proceedings of IGARSS '04, Geoscience and Remote Sensing Symposium*, vol. 1, Sept 2004, p. 312.

[54] F. Rizzo, B. Carpentieri, G. Motta, and J. Storer, "Low-complexity lossless compression of hyperspectral imagery via linear prediction," *Signal Processing Letters, IEEE*, vol. 12, no. 2, pp. 138–141, Feb 2005.

[55] C. Hartzell, L. Graham, T. Tao, H. Goldberg, J. Carpena-Nunez, D. Racek, C. Taylor, and C. Norton, "Data system design for a hyperspectral imaging mission concept," in *Aerospace conference, 2009 IEEE*, March 2009, pp. 1–21.

[56] "CCSDS recommended standard for lossless multispectral & hyperspectral image compression". [Online]. Available: http://public.ccsds.org/publications/archive/123x0b1ec1.pdf

[57] J. Sanchez, E. Auge, J. Santalo, I. Blanes, J. Serra-Sagrista, and A. Kiely, "Review and implementation of the emerging CCSDS recommended standard for multispectral and hyperspectral lossless image coding," in *First International Conference on Data Compression, Communications and Processing (CCP)*, June 2011, pp. 222–228.

[58] G. R. Allen, L. Edmonds, C. W. Tseng, G. Swift, and C. Carmichael, "Single-event upset (SEU) results of embedded error detect and correct enabled block random access memory (block ram) within the xilinx xqr5vfx130," *IEEE Transactions on Nuclear Science*, vol. 57, no. 6, p. 3426, 2010.

[59] A. Schmidt, J. Walters, K. Zick, M. French, D. Keymeulen, N. Aranki, M. Klimesh, and A. Kiely, "Applying radiation hardening by software to Fast Lossless compression prediction on FPGAs," in *2012 IEEE Aerospace Conference*, March 2012, pp. 1–10.

[60] A. Schmidt and M. French, "Fast lossless image compression with radiation hardening by hardware/software co-design on platform FPGAs," in *2013 IEEE International Conference on Application-Specific Systems, Architectures and Processors (ASAP)*, June 2013, pp. 103–106.

[61] D. Keymeulen, N. Aranki, A. Bakhshi, H. Luong, C. Sarture, and D. Dolman, "Airborne demonstration of FPGA implementation of fast lossless hyperspectral data compression system," in *2014 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, July 2014, pp. 278–284.