International Journal of Electrical and Electronic Engineering & Telecommunications
IJEETC

ISSN 2319 – 2518 www.ijeetc.com
Vol. 3, No. 4, October 2014
© 2014 IJEETC. All Rights Reserved

Research Paper

AREA OPTIMIZED IMPLEMENTATION OF LOCAL NEIGHBORHOOD FUNCTION IN ASIP

Esterurani Jaladi[1]* and G S J Mani Kumar[2]

Corresponding Author: **Esterurani Jaladi ✉ jaladiestheru@gmail.com*

As per the analysis of the paper, presents a systematic approach to the design of application-specific instruction-set processors for high speed computation of local neighborhood functions and intrafield deinterlacing. The intended application is real-time processing of high definition video. The approach aims at an efficient utilization of the available memory bandwidth by fully exploiting the data parallelism inherent to the target algorithm class. An appropriate choice of custom instructions and application-specific registers is used together with a very long instruction word architecture in order to mimic a pipelined systolic array. This leads to a processing speed close to the limit imposed by memory bandwidth constraints. For three intrafield deinterlacing algorithms and 2-D convolution with four kernel sizes, the design approach yields speedup factors between 36 and 1330, Area-Time (AT) product improvements between 12 and 243, and energy consumption reduction factors between 13 and 262.

Keywords: Application-specific instruction-set processors (ASIPs), Deinterlacing, Local neighborhood functions, Video processing

INTRODUCTION

General-purpose instruction processors have dominated computing for a long time. However, they tend to lose performance when dealing with non-standard operations and non-standard data not supported by the instruction set format. The need for customizing instruction processors for specific applications is particularly acute in embedded systems, such as cell phones, medical appliances, digital cameras and printers. One way of supporting customisation is to augment an instruction processor with programmable logic for implementing custom instructions and another method is to implement them using existing FPGAs.

Digital embedded systems typically contain one or more programmable processors to run application-specific software. A processor can be general-purpose with extensive tool and operating system support, or special-purpose

[1] M.Tech. Student, Department of ECE, Chirala Engineering College, Chirala 523155, Prakasam Dt., AP.

[2] Associate Professor, Department of ECE, Chirala Engineering College, Chirala 523155, Prakasam Dt., AP.

with better performance and less versatility. Microcontrollers and embedded controllers are special kinds of programmable processors used to control the embedded system and monitor its environment, while digital signal processors (DSP processors) are ideal for high data rate computations.

In a hardware-software co-design methodology, the hardware and software components of an embedded system are designed jointly. Once the designer has determined which parts of the system functionality will be implemented in hardware, and which parts in software, then each of the hardware and software components are designed separately using appropriate tools (hardware synthesis, code generation and hardware-software co-simulation tools). Depending upon the results of simulations, the original design specification may be re-repartitioned and the process reiterated until the system requirements are satisfied.

Embedded systems, increasingly include ASIPs (Application Specific Instruction Set Processors). The key advantage of these programmable components is a tradeoff between efficiency and flexibility since software allows late change in the design cycle and reuse of existing hardware components.

MOORE's law (Mollick E, 2006) is both a blessing and a curse. On one hand, the continuing scaling of semiconductor processes allows for faster systems and opens the door to applications which would not have been possible or afford-able otherwise. For example, we have seen in recent years an explosion in offerings of high denition and portable video applications reaching the consumer market. However, this continuing scaling also brings forward the problem of the ever-increasing design complexity of digital systems. To mitigate this problem and make sure time-to-market objectives are not jeopardized by increasing design and verication efforts, new tools and design methodologies must be developed in parallel with the improvement of semiconductor processes. These tools and methodologies typically improve designers' productivity by leveraging design automation and higher levels of abstraction (Gerstlauer A and Gajski D D, 2002).

A possible solution is to implement complex digital functions using application-specic instruction-set processors (ASIPs) (Jain M K *et al*., 2001). This approach maintains some of the exibility, ease of design and productivity typically associated with general purpose processors and software solutions while approaching the efciency and performance of dedicated hardware.

In this paper, we propose a systematic approach to the design of ASIPs for high speed computation of local neighborhood functions and intra-eld deinterlacing. An appropriate choice of custom instructions and application-specic registers is used together with a very long instruction word (VLIW) architec-ture in order to mimic a pipelined systolic array. This leads to processing speeds close to the limits imposed by memory bandwidth constraints. Control dependencies are rst removed by transforming them into data dependencies. Then, data pro-cessing single instruction-multiple data (SIMD) instructions and custom registers are chosen so as to form the structure of a pipeline. Finally, an appropriate choice of custom load/store instructions makes it possible to nd a scheduling of the in-structions inside the inner loop of the algorithm where one load or store instruction is executed every single instruction cycle. Data processing occurs concurrently with load and store operations.

This paper is organized as follows. Section II presents a short review of the related work. Section III presents the application for better understanding of subsequent sections. In this section, terminology and notation used in the remainder of this paper are introduced. Section IV highlights the characteristics of software code for the target class of algorithms and suggests some code transformation which aim at removing control dependencies. In SectionV, the steps involved in the design of the custom instructions are presented. Finally, results are presented and discussed in Section VI, followed by a conclusion summarizing our main results in Section VII.

RELATED WORK

Re-targetable code generation tools play an crucial role in the ASIP design process since many instruction set architectures have to be evaluated. The target architecture may be changing in order to minimise cost, speed, code size, and power consumption and/or to increase performance of the whole system. This is achieved by designing its architecture, generating code for it, and then evaluating the code for desired performance.

It has been shown that by using ASIPs, a significant reduction in power consumption in devices can be achieved. For instant, 50% power reduction for ISDN-hands free phone. Large instruction word allowed for low clock frequency (France Telecom). 50% power reduction in GSM application (Alcatel). Two ASIPs replaced 8 commercial DSPs 50 MIPs at 20Mhz instead of 116 Mhz (Italtel).

The best known local neighborhood function is the 2-D con-volution. Many fast and area-efcient architectures for imple-menting the 2-D convolution have been proposed in the liter-ature, with recent work focusing on implementation in FPGAs (Cardells-Tormo F and Molinet P L, 2006; Yadav D et al., xxxxx). Architectures based on a linear systolic array are com-monplace for the fast implementation of local neighborhood functions. Examples of this are the two-dimensional convolver described in (Bosi B *et al.*, 1999) and (Diamantaras K I and Kung S Y, 1997), which present the implementation of another class of local neighborhood functions: morphological functions. As an alternative to the classic linear systolic array, a novel cell-based architecture for fast local neighborhood image processing is proposed in (Porter R B *et al.*, 2006).

The use of ASIPs to implement complex functions in dig-ital systems is an active eld of research. Work has been pub-lished in recent years about the use of ASIPs in areas as di-verse as calculation of fast Fourier transforms (FFTs) (Guan X *et al.*, 2009), processing of global positioning data (Kappen G *et al.*, 2007), error correcting codes decoding Alles M *et al.*, 2009), (Muller O *et al.*, 2009), cryptography (O'Melia S and Elbirt A J, 2010; Groszschadl J *et al.*, 2006), video compression (Kim S D *et al.*, 2006), and video processing (Saponara S *et al.*, 2007). Also, a number of design automation solution vendors now offer EDA tool suites for ASIP design. These include Synopsys' Processor Designer (formerly CoWare), Tensilica's Xtensa Xplorer, and Target's IP Designer.

An alternative to ASIPs is provided by high-level synthesis tools, which compile software code into a synthesizable hard-ware description. This is a solution similar to ASIP ows in that the starting point is software code. The SPARK (Gupta S *et al*., 2004) framework improves code efciency using optimization passes similar to the ones found in compilers, but with a focus on leveraging instruction-level parallelism. Then, it schedules operations from the optimized software code onto a xed number of functional units chosen from a library. Finally, it generates VHDL code that contains the instantiation of the functional units, as well as a state machine and control logic that implement the schedule. Another high level synthesis tool which specically targets FPGAs is ROCCC (Guo Z *et al*., 2005). In addition to synthesizing the data path and control logic, ROCCC also analyses the pattern of access to data and improves the performance of local neighbor-hood function implementations through data reuse using smart buffers (Guo Z *et al*., 2004). High processing speeds may be obtained with this approach, especially when the optimization techniques used by the tool are tailored to the algorithm class (Buyukkurt B and Najj W A, 2008).

We previously reported on a high performance ASIP imple-mentation of the PBDI intra-eld deinterlacing algorithm (Aubertin P *et al*., 2009). In this paper, we extend the scope of this work by proposing a systematic ASIP design approach for real-time local neighbor-hood video processing applications. This approach leverages the ASIP design ow in order to improve design productivity.Com-pared to the high-level synthesis approach, which also focusses on design productivity, the ASIP ow and our design approach result in more exible designs, since the processor, in addition to its custom instructions tailored to the application, also retains its full generic instruction set.

LOCAL NEIGHBORHOOD FUNCTIONS

Local neighborhood functions form a class of algorithms of common use in image and video processing. Each pixel in the output image is calculated independently and is a function of a small subset of the input image. This subset—the window—contains the input pixel having the same coordinates as the output pixel, as well as some of its close neighbors. As pixels of differing coordinates are produced, the window “slides along”. Hence, local neighborhood functions are also called sliding window functions.

An image of width W and height H may be represented by W x H the matrix

$$P = \begin{pmatrix} p_{1,1} & p_{1,2} & \cdots & p_{1,W} \\ p_{2,1} & p_{2,2} & \cdots & p_{2,W} \\ \vdots & \vdots & \ddots & \vdots \\ p_{H,1} & p_{H,2} & \cdots & p_{H,W} \end{pmatrix} \qquad \text{...(1)}$$

where each element $p_{i,j}$ is a pixel. The representation of each such pixel is application-dependent. It is usually either a scalar representing a luminance in the case of grayscale images, or a vector representing the color of the pixel in some color space in

the case of color images. The output image, represented by Q matrix , is produced by F applying the local neighborhood function , as follows:

$$q_{i,j} = F(X(i, j)) \quad ...(2)$$

where $X(i, j)$, the window, is the h x w matrix

$$X(i,j) = \begin{pmatrix} x_{1,1} & \cdots & x_{1,w} \\ \vdots & \ddots & \vdots \\ x_{h,1} & \cdots & x_{h,w} \end{pmatrix} \quad ...(3)$$

where

$$x_{k,1} = p_{i+k-1.\,j+l-1} \forall_i \in [1, H-h+1], j \in [1, W-w+1] \quad ...(4)$$

meaning x1,1 is pi,j , and other elements of are the neighboring pixels below and/or right of pi,j. Formally, it is implied by (4) that the output image is smaller than the input image by h-1 lines and w-1 columns. This is because these are the only coordinates for which all the pixels in the window are defined. For example, in X(i,W), x1,2 is pixel pi,W+1 , which is outside the image. In practical implementations, the size of the output image is often extended to match that of the input image. Various means of doing this exist. One option consists in generating boundary pixels by using a separate function which depends on a smaller window.

Another option is to conceptually enlarge the input image by copying existing pixels or by giving the non-existent pixels outside the input image a fixed value. Whole images are scanned following a regular path. This leads to a relationship where some pixels in the window used to produce one pixel are also present in the window used to generate the next pixel. Without loss of generality, let us assume that each image line is scanned from left to right. In that case, if the Nth pixel produced is qi,j, the N+1th pixel produced is qi,j+1, assuming pi,j is not a boundary pixel. This means that, when producing the Nth pixel,x1,2 is pi,j+1 , and that same pixel becomes x1,1when producing the N+1th pixel. More generally, as the window slides from one pixel to the next,xk,l for the Nth pixel becomes xk,i-1for the N+1th pixel for k>1 . This property of the local neighborhood functions offers opportunities for data reuse. The most common local neighborhood function is the two dimensional convolution. It consists in individually weighting each pixel of the window and summing the weighted values, which can be expressed as follows:

$$q_{i,j} = \sum_{k=1}^{h} \sum_{l=1}^{w} a_{k,l} . x_{k,l}$$

where the ak.l are the weights. The 2-D convolution is useful to apply various types of filters to an image in order to sharpen it, blur it, enhance features such as edges, etc. Other simple local neighborhood functions include dilation and erosion. A notable class of complex local neighborhood functions are the intra field deinterlacing algorithms. Deinterlacing is the process of converting an interlaced video sequence into a progressive one. Various classes of methods for performing this conversion exist. Intra-field methods work by generating each complete image from the pixel data of a single field. Missing pixels are generated by interpolating available neighboring pixels. Thus, these methods may be classified as local neighborhood functions. Simple intra field deinterlacing methods consist in generating missing pixels by copying the available pixel just above or below (a method commonly called BOB) or by linear interpolation between the pixel right above and the one right below. These methods often yield poor quality results. Edge-based methods aim at improving quality by tackling a specific artifact: jagged edges. They work by detecting edges in video fields using various methods, and then interpolating in a direction parallel to these edges. The vanilla edge-based method is Edge-Based Line Averaging (ELA). Other, more complex methods aim at further improving image quality through higher angular resolution and more reliable edge detection. They include enhanced ELA (EELA), modified ELA (MELA), pattern-based directional interpolation (PBDI), and others.An application-specific instruction-set processor (ASIP) is a component used in system-on-a-chip design. The instruction set of an ASIP is tailored to benefit a specific application. This specialization of the core provides a tradeoff between the flexibility of a general purpose CPU and the performance of an ASIC.

Some ASIPs have a configurable instruction set. Usually, these cores are divided into two parts: static logic which defines a minimum ISA (instruction-set architecture) and configurable logic which can be used to design new instructions. The configurable logic can be programmed either in the field in a similar fashion to an FPGA or during the chip synthesis.

Developers of complex electronic systems are continuously faced with the challenge of designing more integrated products that offer higher levels of flexibility to address evolving market needs. These devices have a broad range of data processing functions which require programmability and power-efficiency. To meet highly specialized processing requirements in their SoCs, designers often turn to application-specific instruction-set processors (ASIPs), which offer more architectural specialization as well as instruction and data–level parallelism compared to general-purpose processors.

Synopsys' Processor Designer and IP Designer (including IP Programmer and MP Designer, formerly products of Target Compiler Technologies) are proven tools for automating and accelerating the design of highly-efficient ASIPs.

ASIPs possess an instruction set which is tailored to benefit a specific application. Such specialization allows ASIPs to serve as an intermediate between two dominant processor design styles- ASICs which has high processing abilities at the cost of limited programmability and Programmable solutions such as FPGAs that provide programming exibility at the cost of less energy efficiency. In this dissertation the goal is to design ASIP, keeping in mind a temperature sensor system. The platform used for processor design is LISA 2.0 description language and processor designing environment from Co-Ware. Co-ware processor de-signer allows processor architecture to be defined at an abstract level and automatic generation of chain of software tools like assembler, linker and simulator for functional verification followed by RTL level description. RTL level description is used to generate synthesized report of the design using RTL compiler and finally the layout is created using Cadence encounter.

ASIP is Application Specific Instruction-set

Processor dedicated designed for an application domain. ASIP instruction set is specifically designed to accelerate heavy and most used functions. ASIP architecture is designed to implement the assembly instruction set with minimum hardware cost. ASIP DSP is an application specific digital signal processor for iterative data manipulation and transformation extensive applications. General-purpose processor designers think of both the maximum performance and maximum flexibility. The instruction set must be general enough to support general applications. The compiler should offer compilation for all programs and to adapt all programmers' coding behaviors. ASIP designers have to think about applications and cost first. Usually the biggest challenges for ASIP designers are the silicon cost and power consumption. Based on the carefully specified function coverage, the goal of an ASIP design is to reach the highest performance over silicon, over power consumption, as well over the design cost. The requirement on flexibility should be sufficient instead of ultimate. The performance is application specific instead of the highest one. In this tutorial, ASIP, application specific instruction set processor, will be introduced and discussed for audience who want to know ASIP yet not want to design it. The introduction includes ASIP design flow, source code profiling, architecture exploration, assembly instruction set design, design of assembly language programming tool chain, firmware design, benchmarking, and micro architecture design. Two examples, design for instruction set level acceleration of radio baseband, and design for instruction set level acceleration of image and video signal processing, will be introduced.

Deinterlacing

Deinterlacing is the process of converting interlaced video, such as common analog television signals or 1080i format HDTV signals, into a non-interlaced form. Interlaced video frame consists of two sub-fields taken in sequence, each sequentially scanned at odd then even lines of the image sensor; analog television employed this technique because it allowed for less transmission bandwidth and further eliminated the perceived flicker that a similar frame rate would give using progressive scan. CRT based displays were able to display interlaced video correctly due to its complete analogue nature. All of the newer displays are inherently digital in that the display comprises discrete pixels. Consequently the two fields need to be combined into a single frame, which leads to various visual defects which the deinterlacing process should try to minimise. Deinterlacing has been researched for decades and employs complex processing algorithms; however, consistent results have been very hard to achieve.

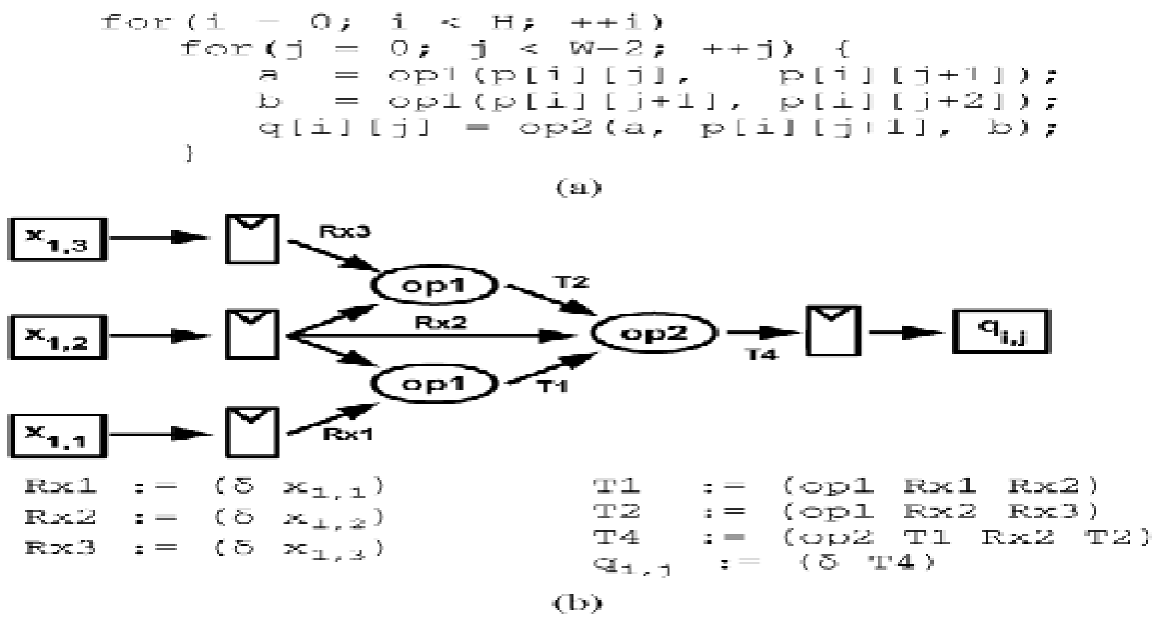
Intra-Field Deinterlacing

Three ASIP implementations of intra-field deinterlacing algorithms were created: one of the ELA algorithm, one of Enhanced ELA and one of the PBDI algorithm. These are three edge-based algorithms of differing complexity. Each implementation is intended to process 24-bit color pixel data (eight bits per color component). Sixteen pixels are generated in nine instruction cycles, and the data path has sufficient width to process four pixels in parallel. The architectures of the three implementations are very similar. Simulations were performed with a data set of four images each having a resolution of 352 288 pixels. Also, a data cache size of 64 kB, with 64 bytes per cache line, was used for the simulations in this section and the next. The clock frequency and processor area are estimates given by the Tensilica tools and assume a 130 nm low voltage process. The area estimates given by the tools do not include the size of the data cache. The slight decrease in clock frequency of the ASIPs when compared to the reference configuration is due to the presence of the 128-bit load/store unit. The custom instructions were designed and pipelined to leave the clock frequency unchanged. The three algorithms possess very different computational complexities. The execution time of the software implementation of the most complex algorithm, PBDI, is 14 times that of the simplest algorithm, ELA. However, the results show that the processing speed is roughly the same for all three ASIP designs.

DESIGN OF CUSTOM INSTRUCTIONS

In this section, we present the steps involved in the choice of custom instructions and application-specific registers. These steps are as follows: first, SIMD instructions which produce multiple output pixels in parallel are created. These instructions make use of a specific intra-line data reuse scheme. Then, each SIMD instruction is split into multiple instructions throughpipelining in order to increase throughput. Some hardware simplifications may be possible at that point. Finally, the load and store instructions are created. Figure 2(a) shows the C language code of a very simple local neighborhood function. This example will be used to illustrate the various steps involved in the design of

Figure 2: Simple Local Neighborhood Function Example with (a) C Language Code



```
for(i = 0; i < H; ++i)
    for(j = 0; j < W-2; ++j) {
        a   = op1(p[i][j],     p[i][j+1]);
        b   = op1(p[i][j+1],  p[i][j+2]);
        q[i][j] = op2(a, p[i][j+1], b);
    }
```

(a)

```
Rx1  := (δ x1,1)                T1    := (op1 Rx1 Rx2)
Rx2  := (δ x1,2)                T2    := (op1 Rx2 Rx3)
Rx3  := (δ x1,3)                T4    := (op2 T1 Rx2 T2)
                                qi,j   := (δ T4)
```

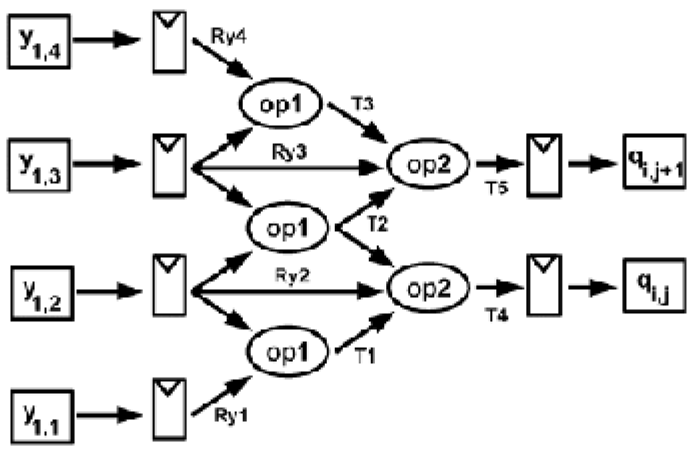
(b)

custom instructions and was contrived to exhibit all the proposed simplifications for illustrative purposes. Each given practical algorithm will likely allow for some of the possible simplifications but not all them. In the code of Figure 2(a), and are arbitrary pure functions. Figure 2(b) shows a representation of a single custom instruction which could implement the local neighborhood function of Figure 2(a). In Figure 2(b), labelled squares indicate input and output pixels. These pixels are typically stored in memory locations. Thus, the squares are indicative of load and store operations. Labelled ellipses represent processing operations, and edges represent intermediate results. These edges are indicative of data dependencies. Finally, the vertical rectangles represent registers. In order to simplify data reuse and instruction scheduling, a load-store architecture is used. Consequently, there are registers for the input pixels between the load operations and the processing operations, and also for the output pixels between the processing operations and the store operations. These registers are noted in the textual representation by the time delay operator, with having the semantic" delayed by one execution cycle".

A. Custom SIMD Instructions

The creation of SIMD instructions,which calculate more than one pixel in parallel, is discussed in this section. Calculating many pixels in parallel leads to an increase in silicon area requirements, and only leads to a performance improvement if the datapath is not starved by an inability to feed data and retrieve results at a sufficient rate. For this reason, the number of pixels calculated in parallel must be balanced with the size of the

Figure 3: Single SIMD Custom Instruction



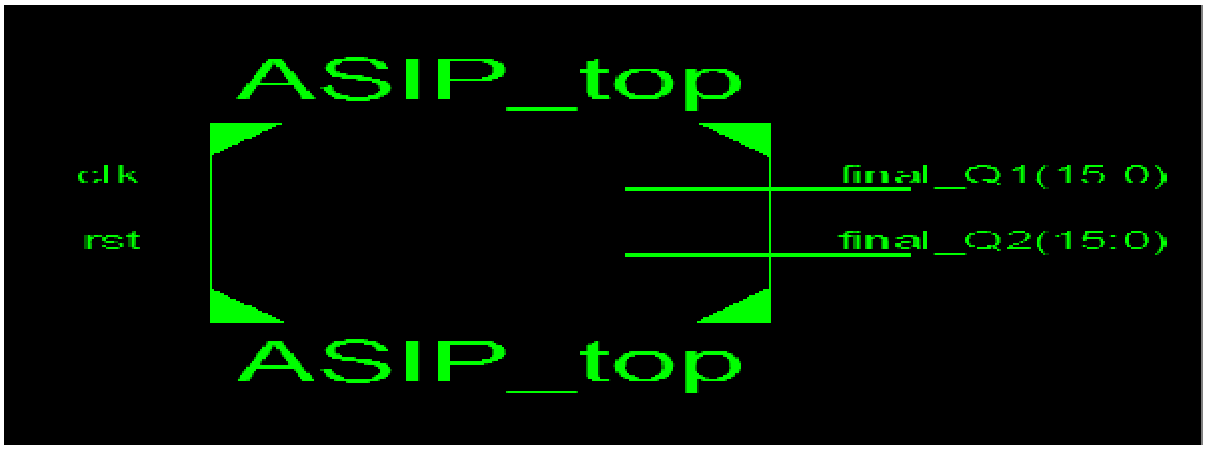
```
Ry1  := (δ $y_{1,1}$)          T3      := (op1 Ry3 Ry4)
Ry2  := (δ $y_{1,2}$)          T4      := (op2 T1 Ry2 T2)
Ry3  := (δ $y_{1,3}$)          T5      := (op2 T2 Ry3 T3)
Ry4  := (δ $y_{1,4}$)          $q_{i,j}$     := (δ T4)
T1   := (op1 Ry1 Ry2)          $q_{i,j+1}$   := (δ T5)
T2   := (op1 Ry2 Ry3)
```

bus used to access the data in memory. The link between and the bus size is addressed in Section V-F.

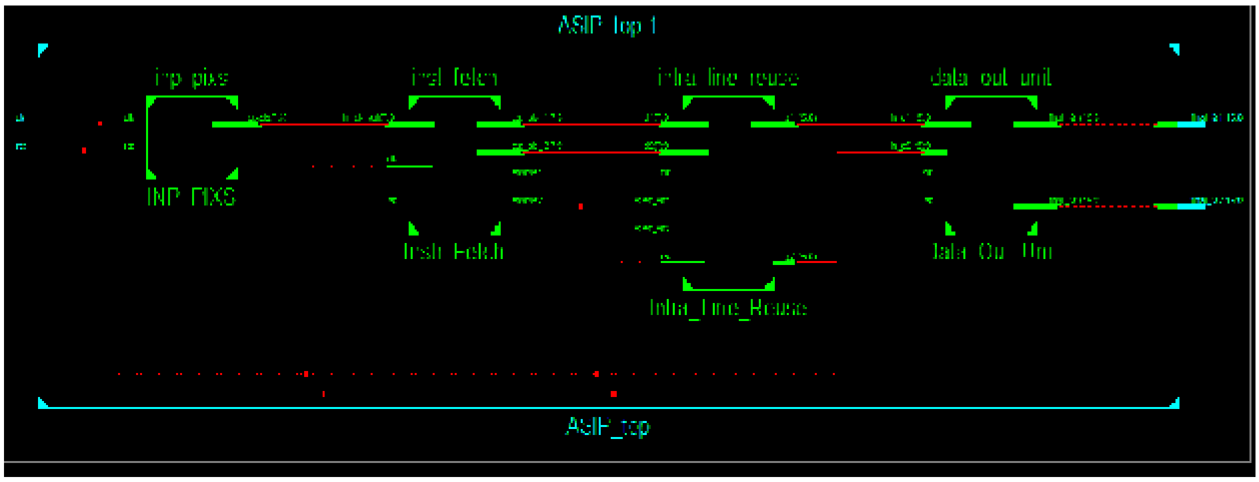
RESULTS

This section presents the results obtained for the implementation of various local neighborhood functions using the approach described in previous sections. Three intra-field deinterlacing algorithms were implemented, as well as 2-D convolution with four different kernel sizes. Performance results for one of the deinterlacing algorithms were previously published in [21]. All implementations are based on the Xtensa LX2 configurable and extensible processor [29], [30]. In the configuration which is used for the ASIPs, it has a 128-bit memory bus and load-store unit which can be used by custom load-store instructions. The results presented here were obtained from cycle-accurate simulations performed using the Tensilica tools. For

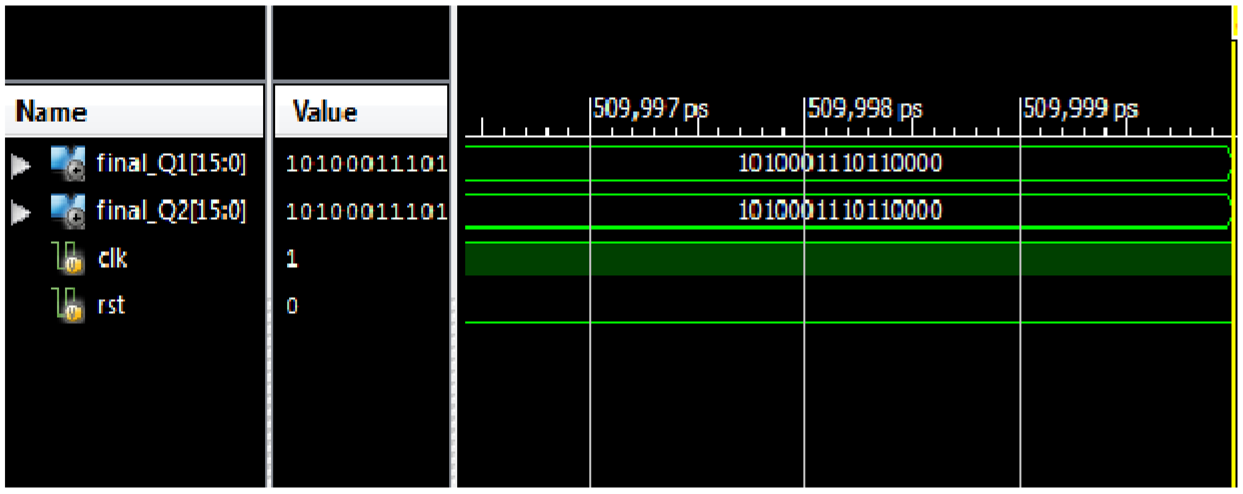
RTL Schematic



Internal View of a Design



Waveform



comparison purposes, in each case, simulation results are presented for the ASIP, but also for a pure software implementation running on a bare Xtensa LX2 without instruction-set extension. In this reference configuration, the Xtensa LX2 is similar to other common general-purpose 32-bit RISC processors.

While digital signal processors aremore likely to be used in practical real-time video processing systems, they also differ widely in architecture and configuration, which makes it difficult to establish a useful point of comparison. By contrast, a 32-bit general-purpose RISC processor is a general basis against which DSPs and ASIPs alike can be compared. The reference configuration uses a 32-bit memory bus. In fact, increasing the bus width to 128 bits leads to an increase of 29% in processor area but does not improve execution speed.

This is because the marginal improvement in terms of the reduced number of clock cycles is counterbalanced by a decrease in clock frequency. The clock frequency decrease is due to the presence of the 128-bit load/store unit.

A. INTRA-FIELD DEINTERLACING

Three ASIP implementations of intra-field deinterlacing algorithms were created: one of the ELA algorithm [24], one of Enhanced ELA [25] and one of the PBDI algorithm [27]. These are three edge-based algorithms of differing complexity. Each implementation is intended to process 24-bit color pixel data (eight bits per color component). Sixteen pixels are generated in nine instruction cycles, and the data path has sufficient width to process four pixels in parallel. The architectures of the three implementations are very similar. The architecture of the PBDI implementation is described in detail in [21]. Simulations were performed with a data set of four images each having a resolution of 352 288 pixels. Also, a data cache size of 64 kB, with 64 bytes per cache line, was used for the simulations in this section and the next. A summary of the profiling results is given in Table I. The clock frequency and processor area are estimates given by the Tensilica tools and assume a 130 nm low voltage process. The area estimates given by the tools do not include the size of the data cache. The slight decrease in clock frequency of the ASIPs when compared to the reference configuration is due to the presence of the 128-bit load/store unit. The custom instructions were designed and pipelined to leave the clock frequency unchanged. The three algorithms possess very different computational complexities. The execution time of the software implementation of the most complex algorithm, PBDI, is 14 times that of the simplest algorithm, ELA. However, the results show that the processing speed is roughly the same for all three ASIP designs. This is to be expected since the pattern of access to data is the same: in all three cases, the kernel has a height of two pixels, which means that two blocks of input pixels are needed to produce one output block.

CONCLUSION

A systematic approach to the design of ASIPs for high speed computation of local neighborhood functions and intra-field deinterlacing was proposed. This approach aims at an efficient utilization of the available memory bandwidth by fully exploiting the data parallelism inherent to the target algorithm class. An appropriate choice of custom instructions and application-specific registers is used together with a VLIW architecture in order to mimic a pipelined systolic array. Results for the implementation of three intra-field deinterlacing algorithms and four kernel sizes of the 2-D convolution function were presented. These results show that a significant improvement of the processing speed, the Area-Time (AT) product and the energy consumption is possible using this approach. Indeed, speedup factors between 36 and 1330 were obtained when compared to a pure software implementation. Also, again compared to a pure software implementation, the AT product was improved by factors between 12 and 243. We also report significant improvements in terms of energy consumption, with improvement factors varying between 13 and 262 among the seven algorithms considered. The results seem to indicate that the factor of improvement of the AT product is a good predictor of the improvement in terms of energy consumption.

REFERENCES

1. Alles M, Vogt T and Wehn N (2008), “Flexichap: A reconfigurable ASIP for convolutional, turbo, and LDPC code decoding”, in *Proc. 5th Int. Symp. Turbo Codes Related Topics*, pp. 84–89.
2. Aubertin P, Mohammadi H M, Savaria Y and Langlois J M P (2009), “High performance ASIP implementation of PBDI—A new intra-field deinterlacing method”, in *Proc. Joint IEEE NEWCAS-TAISA Conf.*, pp. 372–375.
3. Bosi B, Savaria Y, and Bois G (1999), “Reconfigurable pipelined 2-D convolvers for fast digital signal processing,” *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, Vol. 7, No. 3, pp. 299–308.
4. Buyukkurt B and Najj W A (2008), “Compiler generated systolic arrays for

wavefront algorithm acceleration on FPGAs”, in *Proc. Int. Conf. Field Program. Logic Appl*., pp. 655–658.

5. Cardells-Tormo F and Molinet P L (2006), “Area-efficient 2-D shift-variant convolvers for FPGA-based digital image processing,” *IEEE Trans. Circuits Syst. II, Expr. Briefs*, Vol. 53, No. 2, pp. 105–109.

6. Diamantaras K I and Kung S Y (1997), “A linear systolic array for real-time morphological image processing,” *J. VLSI Signal Process.*, Vol. 17, No.1, pp. 43–55.

7. Gerstlauer A and Gajski D D (2002), “System-level abstraction semantics,” in *Proc. 15th Int. Symp. Syst. Synth.*, pp. 231–236.

8. Groszschadl J, Ienne P, Pozzi L, Tillich S and Verna A K (2006), “Combining algorithm exploration with instruction set design: A case study in elliptic curve cryptography”, in *Proc. Conf. Design, Autom. Test Euro*, pp. 218–223.

9. Guan X, Lin H, and Fei Y (2009), “Design of an application-specific instruction set processor for high-throughput and scalable FFT,” in *Proc. Conf. Design, Autom., Test Euro. (DATE)*, pp. 1302–1307.

10. Guo Z, Buyukkurt B and Najjar W (2004), “Input data reuse in compiling window operations onto reconfigurable hardware”, *SIGPLAN Notices*, Vol. 39, No. 7, pp. 249–256.

11. Guo Z, Buyukkurt B, Najjar W and Vissers K (2005), “Optimized generation of data-path from c codes for FPGAs”, in *Proc. Conf. Design, Autom., Test Euro*, pp. 112–117.

12. Gupta S, Gupta R, Dutt N and Nicolau A (2004), SPARK: A Parallelizing Approach to the High-Level Synthesis of Digital Circuits, NewYork, Springer-Verlag.

13. Jain M K, Balakrishnan M, and Kumar A (2001), “ASIP design methodologies: Survey and issues,” in *Proc. 14th Int. Conf. VLSI Design*, pp. 76–81.

14. Kappen G, Bahri S, Priebe O, and Noll T (2007), “Evaluation of a tightly coupled ASIP/co-processor architecture used in GNSS receivers,” in *Proc. Int. Conf. Appl.-Specific Syst., Arch. Process. (ASAP)*, pp. 1302-1307.

15. Kim S D, Lee J H, Hyun C J and Sunwoo M H (2006), “ASIP approach for implementation of H.264/AVC”, in *Proc. Conf. Asia South Pacific Design Autom*., pp. 758–764.

16. Mollick E (2006), “Establishing Moore’s law,” *IEEE Annals History Comput.*, Vol. 28, No. 3, pp. 62–75.

17. Muller O, Baghdadi A and Jézéquel M (2009), “From parallelism levels to a multi-ASIP architecture for turbo decoding”, *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, Vol. 17, No. 1, pp. 92–102.

18. O’Melia S and Elbirt A J (2010), “Enhancing the performance of symmetric- key cryptography via instruction set extensions”, *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, Vol. 18, No. 11, pp. 1505–1518.

19. Porter R B, Frigo J R, Gokhale M, Wolinski C, Charot F, and Wagner C (2006), "A run-time re-configurable parametric architecture for local neighborhood image processing," in *Proc. IEEE Comput. Soc. 9th EUROMICRO Conf. Digit. Syst. Design (DSD)*, pp. 107–115.

20. Saponara S, Fanucci L, Marsi S, Ramponi G, Kammler D and Witte E (2007), "Application-specific instruction-set processor for retinex-like image and video processing", *IEEE Trans. Circuits Syst. II, Expr. Briefs*, Vol. 54, No. 7, pp. 596–600.

21. Yadav D, Gupta A, and Mishra A, "A fast and area efficient 2-D convolver for real time image processing," in *Proc. IEEE Region 10 Conf. TENCON*, pp. 1–6.