VLSI Design

VLSI Circuits, Systems, and Architectures for Advanced Image and Video Compression Standards

Guest Editors: Maurizio Martina, Muhammad Shafique, and Andrey Norkin

VLSI Circuits, Systems, and Architectures for Advanced Image and Video Compression Standards

VLSI Circuits, Systems, and Architectures for Advanced Image and Video Compression Standards

Guest Editors: Maurizio Martina, Muhammad Shafique, and Andrey Norkin

Editorial Board

Contents

Hindawi Publishing Corporation
VLSI Design
Volume 2012, Article ID 102585, 3 pages
doi:10.1155/2012/102585

Editorial

VLSI Circuits, Systems, and Architectures for Advanced Image and Video Compression Standards

Maurizio Martina,[1] Muhammad Shafique,[2] and Andrey Norkin[3]

[1] *VLSI Lab, Department of Electronics and Telecommunications, Polytechnic University of Turin, Corso Duca degli Abruzzi 24, I-10129 Torino, Italy*
[2] *Chair for Embedded Systems (CES), Department of Computer Science, Karlsruhe Institute of Technology (KIT) Kaiserstrasse 12, 76131 Karlsruhe, Germany*
[3] *Ericsson Research, Torshamnsgatan 23, 164 83 Stockholm, Sweden*

Correspondence should be addressed to Maurizio Martina, maurizio.martina@polito.it

Received 26 August 2012; Accepted 26 August 2012

1. Introduction

In the last ten years image and video compression research developed new techniques and tools to achieve high compression ratios even when high quality is required. These techniques involve several parts of the compression/decompression chain including transform stages, entropy coding, motion estimation, intraprediction, and filtering.

At the same time image and video compression standards were ready to identify the most relevant novelties and incorporate them. In this scenario JPEG2000 and MPEG4-part 10, also known as H.264/advanced video coding (AVC), are two important examples of standards able to perform significantly better than their ancestors JPEG and MPEG2. Recently, stemming from the AVC standard, video compression started facing new challenges. The increasing request for high quality, virtual reality, and immersive gaming highlighted how video compression standards have to face not only high definition and high rate sequences but also 3D or multiview systems. These challenges are part of the high efficiency video coding (HEVC) and multiview video c oding (MVC) standards. Unfortunately, both HEVC and MVC standards yield to high complexity burden.

In particular, even if some works on the implementation of the MVC standard have already been proposed in the literature, several critical aspects, including performance, memory management, power consumption, and reconfigurability are still open issues. On the other hand, HEVC is the ultimate standard for video compression issued by the Joint Collaborative Team on Video Coding. As a consequence, works available in the literature mainly address new tools and performance of the HEVC standard. Thus, results on complexity and implementation issues are still on going. In this scenario, advances in camera and display technologies and continuously increasing users' sensation for true three-dimensional (3D) reality have led to the evolution of new 3D video services with multiview videos, like true-3DTV, free viewpoint TV (FTV), realistic-TV, in-car 3D-infotainment, 3D-personal video recording and playback, 3D-surveillance, immersive video conferencing, high-end medical imaging, remote eHealth applications like 3D-teleoperation theaters, telework office, teleshopping, and so forth. The feasibility of multiview video recording of 2–4 views on mobile devices has been demonstrated by the early case studies on 3D-camcorders/3D-mobile phones by Panasonic, Sharp Lynx, Fujifilm, Sony, Minox, etc. However, multiview video processing of 4–8 views for mobile devices and of 16−>100 views for advanced 3D applications (3DTV/FTV, 3D-medical imaging, 3D surveillance, etc.) is foreseen to be adopted in 2015–2020 at a wide range.

On the other hand, HEVC [4] is a next generation video standard, which is regarded as a next major step in video coding standardization after AVC. The standardization has been conducted jointly by VCEG and MPEG in a Joint Collaborative Team on Video Coding (JCT-VC). The standardization has reached a committee draft stage in February 2012. When starting the standardization, the goal was to achieve twice better compression efficiency compared to

AVC standard. That means that the same subjective picture quality should be achieved at twice lower bitrate compared to High Profile of AVC. The standard is primarily focused on coding higher-resolution video, such as HD and beyond. Higher-resolution videos often have different signal statistics from the lower-resolution content. Moreover, coding higher-resolution video also generates higher requirements to the compression ratio as well as to computational complexity of both encoding and decoding. Higher resolutions and higher frame rates set restriction on standard's computational complexity, which was taken into account when working on the standard. Another aspect of video compression that was addressed in developing HEVC is a requirement of easy parallelization, which facilitates higher resolution coding by performing encoding and decoding of video in parallel on multiple processors.

In order to meet up the present and future demands of different multimedia applications, one interesting research direction concerns the development of unified video decoders that can support all popular video standards on a single platform. It is worth noting that several video compression standards use the DCT-based image compression. Since the DCT computation and quantization process are computation intensive, several algorithms are proposed in literature for computing them efficiently in dedicated hardware. Research in this domain can be classified into three parts: (i) reduction of the number of arithmetic operators required for DCT computation, (ii) computation of DCT using multiple constant multiplication schemes, (iii) optimization of the DCT computation in the context of image and video encoding.

Another important aspect related to the complexity of video-coding standards is the large amount of coding tools used to achieve high compression rates while preserving the video visual quality. As an example, the prediction steps of H.264/AVC (composed by intraframe prediction and interframes prediction) are responsible for the main contribution of compression provided by this standard, which results in about 50% fewer bits to represent a video when compared to MPEG-2. This result is achieved through the insertion of a large number of coding modes in the prediction steps and selecting the best one to encode each macroblock (MB). However, the computation of a large number of prediction modes provided by H.264/AVC standard is extremely expensive in terms of computational complexity. In particular for HD formats several millions of complete encoding operations are needed for each video frame. In order to support real time video encoding and decoding, specific architectures are developed. As an example, multicore architectures have the potential to meet the performance levels required by the realtime coding of HD video resolutions. However, to exploit multicore architectures, several problems have to be faced, such as the subdivision of an encoder application in modules that can be executed in parallel. Once a good partitioning is achieved, the optimization of a video encoder should take advantage of the data level parallelism to increase the performance of each encoder module running on the processing element of the architecture. A common approach is to use single instruction multiple data instructions to exploit the data level parallelism during the execution. However, several points have not been addressed yet, for example, how the data level parallelism is exploited by SIMD and which instructions are more useful in video processing. Moreover, different instruction set architectures (ISAs) are available in modern processors and comparing them is an important step toward efficient implementation of video encoders on SIMD architectures.

In 2007, the notion of electronic system level design (ESLD) has been introduced as a solution to decrease the time to market using high-level synthesis. In this context, CAL was introduced as a general-use data flow target agnostic language based on the data flow process network model of computation. The MPEG community standardized the RVC-CAL language in the reconfigurable video coding (RVC) standard. This standard provides a framework to describe the different functions of a codec as a network of functional blocks developed in RVC-CAL and called actors. Some hardware compilers of RVC-CAL were developed, but their limitation is the fact that they cannot compile high-level structures of the language so these structures have to be manually transformed. Thus, this research field requires further investigation. This special issue is dedicated to research problems and innovative solutions introduced above in all aspects of design and architecture addressing realization issues of cutting-edge standards for image and video compression. The authors have focused on different aspects including (i) VLSI architectures for computationally intensive blocks, such as the DCT and the intraframe coding mode, (ii) automatic code generation and multicore implementation of complex MPEG4 and H.264 video encoders. Due to the increasing importance of stereo and 3D video processing an invited paper dealing with this topic is included in the issue.

2. VLSI Architectures for Computationally Intensive Blocks

As long as new standards have been developed, several research efforts were spent to design efficient VLSI architectures for computationally intensive blocks. Stemming from the works of Chen and Loeffler, several techniques were proposed to reduce the complexity and to increase the flexibility of architectures for the computation of the DCT. Formal techniques as subexpression elimination and canonical sign digit (CSD) representation are viable techniques to optimize architectures for the computation of the DCT. The paper "optimized architecture using a novel subexpression elimination on loeffler algorithm for DCT based image compression" by M. Jridi et al. presents a novel common sub-expression elimination technique that is used with the canonical sign digit (CSD) representation to reduce the complexity of the Loeffler algorithm for DCT implementation. An FPGA-based implementation is provided with video quality results in terms of peak signal to noise ratio (PSNR). Other approaches are based on the factorizing the DCT by the means of simpler and modular structures. Some these ideas were already described in Rao Kamisetty works, but their impact on VLSI implementation has not been completely studied especially when flexibility and multistandard

requirement have to be taken into account. In the paper "N point DCT VLSI architecture for emerging HEVC standard" by A. Ahmed et al. a variable-sized DCT architecture is presented. The architecture stems from the Walsh-Hadamard transform and uses the lifting scheme to reduce the number of multiplications required. The authors provide hardware implementation results for a 90 nm ASIC technology. The paper "low cost design of a hybrid architecture of integer inverse DCT for H.264, VC-1, AVS, and HEVC" by M. Martuza and K. A. Wahid proposes a unified architecture for computation of the integer inverse DCT of multiple modern video codecs. Moreover, the authors show both FPGA- and ASIC-based implementation results.

It is known that the DCT is only one of the most computationally intensive blocks in video compression systems. Several other blocks, including motion estimation and entropy coding, are known to be a significant part of the computational burden of video compression systems. With the H.264/AVC standard very high quality is obtained even with very low bit rates. Such an impressive improvement is obtained due to the possibility to employ a large number of new tools, as intraprediction, coupled with several coding modes. As a consequence, the optimal choice of a coding mode is a very computationally intensive task. Thus, techniques for the fast identification of the optimal or nearly-optimal coding mode are of paramount importance. The paper "low complexity hierarchical mode decision algorithms targeting VLSI architecture design for the H.264/AVC video encoder" by G. Corrêa et al. presents a set of heuristic algorithms targeting hardware architectures that lead to earlier selection of one encoding mode. The resulting solution leads to a significant complexity reduction in the encoding process at the cost of a relatively small compression performance penalty.

3. Automatic Code Generation and Multicore Implementation

The development of VLSI architectures, circuits, and systems for video compression is a time-consuming task. As a consequence, industries are always working hard to be ready with product that are on the cutting-edge of the available technology. Automatic code generation is a very appealing strategy to speed up the design process and to reduce recurrent costs. In the paper "automatic generation of optimized and synthesizable hardware implementation from high-level dataflow programs" by K. Jerbi et al., the authors describe a methodology that from a high-level language called Cal Actor Language (CAL), which is target agnostic, automatically generates image/video hardware implementations able to largely satisfy the real time constraints for an embedded design on FPGA.

Other directions have been investigated to reduce the time required to develop hardware components. The availability of high performance multicore processors (e.g. GPUs) is pushing several researchers to use software solutions even for very complex systems as video encoders. This direction is investigated in the paper "an efficient multi-core SIMD implementation for H.264/AVC encoder", by M. Bariani et al. where the optimization process of a H.264/AVC encoder on three different architectures with emphasis on the use of SIMD extensions is described. Experimental results are given for all the three architectures.

4. Depth Maps Extraction

The availability of devices for 3D displaying together with standards for 3D and multiview video processing has highlighted how 3D video is a very challenging topic. One of the main issues is related to the fact that the signals acquired from cameras are 2D, and signal processing is required to merge data together to create a 3D video sequence or to create a new view of the scene. In particular, it is widely recognized that extracting depth maps from 2D images is one of the key elements to create 3D video. In this perspective, the invited paper "hardware design considerations for edge-accelerated stereo correspondence algorithms" by C. Ttofis and T. Theocharides deals with this hot topic: extracting depth maps from 2D images. In particular, in this work the authors present an overview of the use of edge information as a means to accelerate hardware implementations of stereo correspondence algorithms. Their approach restricts the stereo correspondence algorithm only to the edges of the input images rather than to all image points, thus resulting in a considerable reduction of the search space. The resulting algorithms are suited to achieve real-time processing speed in embedded computer vision systems. For both algorithms, optimized FPGA architectures are presented.

Maurizio Martina
Muhammad Shafique
Andrey Norkin

Hindawi Publishing Corporation
VLSI Design
Volume 2012, Article ID 298396, 14 pages
doi:10.1155/2012/298396

Research Article

Automatic Generation of Optimized and Synthesizable Hardware Implementation from High-Level Dataflow Programs

Khaled Jerbi,[1,2] Mickaël Raulet,[1] Olivier Déforges,[1] and Mohamed Abid[2]

[1] *IETR/INSA. UMR CNRS 6164, 35043 Rennes, France*
[2] *CES Laboratory, National Engineering School of Sfax, 3038 Sfax, Tunisia*

Correspondence should be addressed to Khaled Jerbi, khaled.jerbi@insa-rennes.fr

Received 16 December 2011; Revised 18 April 2012; Accepted 15 May 2012

Academic Editor: Maurizio Martina

In this paper, we introduce the Reconfigurable Video Coding (RVC) standard based on the idea that video processing algorithms can be defined as a library of components that can be updated and standardized separately. MPEG RVC framework aims at providing a unified high-level specification of current MPEG coding technologies using a dataflow language called Cal Actor Language (CAL). CAL is associated with a set of tools to design dataflow applications and to generate hardware and software implementations. Before this work, the existing CAL hardware compilers did not support high-level features of the CAL. After presenting the main notions of the RVC standard, this paper introduces an automatic transformation process that analyses the non-compliant features and makes the required changes in the intermediate representation of the compiler while keeping the same behavior. Finally, the implementation results of the transformation on video and still image decoders are summarized. We show that the obtained results can largely satisfy the real time constraints for an embedded design on FPGA as we obtain a throughput of 73 FPS for MPEG 4 decoder and 34 FPS for coding and decoding process of the LAR coder using a video of CIF image size. This work resolves the main limitation of hardware generation from CAL designs.

1. Introduction

User requirements of high quality video are growing which causes a noteworthy increase in the complexity of the algorithms of video codecs. These algorithms have to be implemented on a target architecture that can be hardware or software. In 2007, the notion of Electronic System Level Design (ESLD) has been introduced in [1] as a solution to decrease the time to market using high-level synthesis which is an automatic compilation of high-level description into a low-level one called register transfer level (RTL). The high-level description is governed by models of computation which are the rules defining the way data is transferred and processed. Many solutions were developed to automate the hardware generation of complex algorithms using ESLD. Synopsys developed a C to gate compiler called synphony [2]. Mentor Graphics also created a C to HDL compiler called Catapult C [3, 4]. For their NIOS II, Altera introduces C2H as a converter from C to HDL [5, 6]. To extend Matlab for hardware generation from functional blocks, Mathworks created a hardware generator for FPGA design [7]. In the university research field, STICC laboratory in France developed a high-level synthesis tool called GAUT that extracts parallelism and generates VHDL code from a pure C description [8, 9]. The common point between all previously quoted tools is the fact that they are application-specific generators which means that they are not always efficient on an entire multi-component system description.

In this context, CAL [10] was introduced in the Ptolemy II project [11] as a general-use dataflow target agnostic language based on the dataflow Process Network (DPN) Model of Computation [12] related to the Kahn Process Network (KPN) [13]. The MPEG community standardized the RVC-CAL language in the MPEG RVC (Reconfigurable Video Coding) standard [14]. This standard provides a framework to describe the different functions of a codec as a network of functional blocks developed in RVC-CAL and called actors. Some hardware compilers of RVC-CAL were developed but their limitation is the fact that they cannot compile high-level

structures of the language so these structures have to be manually transformed.

In [15], we presented an original functional method to quicken the HDL generation using a software platform for rapid design and validation of a high complexity dataflow architecture but going from high to low-level representation used to be manual. Therefore, we proposed to add automatic transformations to make any RVC-CAL design synthesizable.

This paper extends a preliminary work presented in [16] by introducing efficient optimizations and their impact on the area and time consumption of the design. The transformation tool analyzes the RVC-CAL code and performs the required transformations to obtain synthesizable code whatever the complexity of the considered actor. In Section 2, we explain the main advantages of using MPEG RVC standard for signal processing algorithms and the key notions of the RVC-CAL language and its behavioral structures and mechanisms. The proposed transformation process is detailed in Section 4 and finally hardware implementation results of MPEG4 Part2 decoder and LAR codec are presented in Sections 5 and 6.

2. Background

Since the beginning of ISO/IEC/WG11 (MPEG) in 1988 with the appearance of MPEG-1, many video codecs have been developed (MPEG-4 part2, MPEG SVC, MPEG AVC, HEVC, etc.) with an increasing complexity and so they take longer time to be produced. In addition, every standard has a set of profiles depending on the implementation target or the user specifications. Consequently, it became a tough task for standard communities to develop, test, and standardize a decoder at any given time. Moreover, the standards specification is monolithic which makes it harder to reuse or update some existing algorithms. This ascertainment originated a new conception methodology standard called Reconfigurable Video Coding introduced by MPEG.

In the following, we present an overview of MPEG RVC standard and associated tools and frameworks, we also present the main features of CAL actor language and the limitations that motivated this work.

2.1. MPEG RVC. RVC presents a modular library of elementary components (actors). The most important and attractive features of RVC are reconfigurability and flexibility. An RVC design is a dataflow directed graph with actors as vertices and unidirectional FIFO channels as edges. An example of a graph is shown in Figure 1.

Actually, defining video processing algorithms using elementary components is very easy and rapid with RVC since every actor is completely independent from the rest of the other actors of the network. Every actor has its own scheduler, variables, and behavior. The only communication of an actor are its input ports connected to the FIFO channels to check the presence of tokens and as explained later an internal scheduler is going to allow or not the execution of elementary functions called actions depending on their corresponding firing rules (see Section 3). Thus, RVC insures concurrency, modularity, reuse, scalable parallelism, and encapsulation. In [17], Janneck et al. show that, for hardware designs, *RVC standard allows a gain of 75% of development time* and considerably reduces the number of lines compared with the manual HDL code. To manage all the presented concepts of the standard, RVC presents a framework based on the use of the following.

(i) A subset of the CAL actor language called RVC-CAL that describes the behavior of the actors (see details in Section 2.2).

(ii) A language describing the network called FNL (Functional unit Network Language) that lists the actors, the connections and the parameters of the network. FNL is an XML dialect that allows a multilevel description of actors hierarchy which means that a functional unit can be a composition of other functional units connected in another network.

(iii) Bitstream syntax Description Language (BSDL) [18, 19] to describe the structure of the bitstream.

(iv) An important Video Tool Library (VTL) of actors containing all MPEG standards. This VTL is under development and it already contains 3 profiles of MPEG 4 decoders (Simple Profile, Progressive High Profile and Constrained Baseline Profile).

(v) Tools for edition, simulation, validation and automatic generation of implementations:

  (a) open DF framework [20] is an interpreter infrastructure that allows the simulation of hierarchical actors network. Xilinx contributed to the project by developing a hardware compiler called OpenForge (available at http://openforge.sourceforge.net/) [21] to generate HDL implementations from RVC-CAL designs.

  (b) open RVC-CAL Compiler (Orcc) (available at http://orcc.sourceforge.net/) [19] is an RVC-CAL compiler under development. It compiles a network of actors and generates code for both hardware and software targets. Orcc is based on works on actors and actions analysis and synthesis [22, 23]. In the front-end of Orcc, a graph network and its associated CAL actors are parsed into an abstract syntax tree (AST) and then transformed into an intermediate representation that undergoes typing, semantic checks and several transformations in the middle-end and in the back-end. Finally, pretty printing is applied on the resulting IR to generate a chosen implementation language (C, Java, Xlim, LLVM, etc.).

At this level, the question is that *why RVC-CAL and not C*? Actually, a C description involves not only the specification of the algorithms but also the way inherently parallel computations are sequenced, the way data is exchanged throw inputs and outputs, and the way computations are mapped. Recovering the original intrinsic properties of the algorithms by analyzing the software program is impossible.

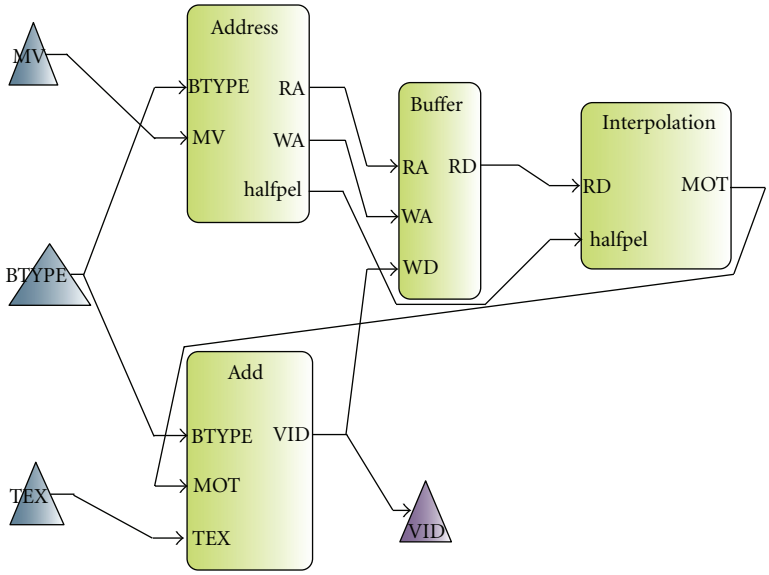

Figure 1: Graph example. Block diagram of the motion compensation of an MPEG 4 part 2 decoder.

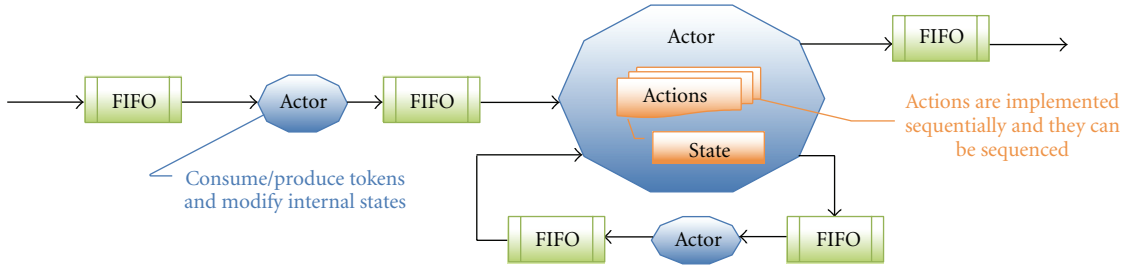

Figure 2: CAL actor model.

In addition, the opportunities for restructuring transformations on imperative sequential code are very limited compared to the parallelization potential available on multi-core platforms. For these reasons, RVC adopted the CAL language for actors specification. The main notions of this language are presented below.

2.2. CAL Actor Language. The execution of an RVC-CAL code is based on the exchange of data tokens between computational entities (actors). Each actor is independent from the others since it has its own parameters and finite state machine if needed. Actors are connected to form an application or a design, this connection is insured by FIFO channels. Executing an actor is based on *firing* elementary functions called *actions*. This action firing may change the state of the actor in case of an FSM. An RVC-CAL dataflow model is shown in the network of Figure 2.

Figure 3 presents an example of a CAL actor realizing the sum between two tokens read from its two input ports.

Like in VHDL, an actor definition begins by defining the I/O ports and their types then actions are later listed. An action begins also by defining the I/O ports it uses from the list of ports of the actor and this definition includes the number of tokens this action have to find in the FIFO to be fireable. In the "sum" actor, the internal scheduler allows action "add" only when there is at least one token in the FIFO of port "INPUT1" and one token in the FIFO of port "INPUT2" and this property explains how an actor can be totally independent and can neither read nor modify the state of any other actor. Of course, an actor may contain any number of actions that can be governed by an internal finite state machine. At a specific time two or more actions may have the required conditions to be fired so the notion of priority was introduced (see details in Section 3).

For the same behavior, an actor may be defined in different ways. Let us consider the "sum-5" actor of Figure 4 that reads 5 tokens in a port "IN," computes their sum and produces the result in a port "OUT."

In Figure 4(a), the required algorithm is defined in only one action. The condition of 5 required tokens is expressed by the instruction "repeat 5." Action "add" fires by consuming the 5 tokens from the FIFO into an internal buffer "I." After data storage, the algorithm of the action is applied. Finally the action firing finishes by writing the result in the port "OUT".

Such description is very fast to develop and implement on software targets but for hardware implementations a multitoken read is not appropriate. This is the reason of

```
actor sum ()
(int size=8) INPUT1, (int size=8) INPUT2 ==> int(size=8) OUTPUT:

 add: action INPUT1:[ i1 ], INPUT2[i2] ==> OUTPUT:[s]
 var
   int s
 do
   s:= i1 + i2 ;
 end
end
```

Figure 3: Example of sum actor.

```
actor sum-5 () int (size=8) IN
==> int(size=8) OUT:

 add: action IN:[ i ] repeat 5
 ==> OUT:[ s ]
 var
   int s := 0
 do
   foreach int k in 0 .. 4 do
     s := s + i[k] ;
   end
 end
end
```

(a) SW-oriented definition

```
actor sum-5 () int (size=8) IN
==> int(size=8) OUT:

 List (type: int (size=8), size = 5) data;
 int counter :=0 ;

 read: action IN:[ i ] ==>
 do
   data[counter] := i ;
   counter := counter + 1 ;
 end

 read_done: action ==>
 guard
   counter = 5
 do
   counter := 0 ;
 end

 process: action ==> OUT:[ s ]
 var
  int s := 0
 do
   foreach int k in 0 .. 4 do
     s := s + data[k] ;
   end
 end

 schedule fsm state0:
   state0 (read)      --> state0;
   state0 (read_done) --> state1;
   state1 (process)   --> state0;
 end

 priority
   read_done > read;
 end

end
```

(b) HW-oriented definition

Figure 4: Two-way definition example of sum-5 actor behavior.

developing the equivalent monotoken code of Figure 4(b). In this description, we use a finite state machine to lock the actor in the state "state0." While counter¡5, only the action "read" can be fired to store tokens one per one in "data" buffer. Once the condition of action "read_done" (counter = 5) is true, both of "read" and "read_done" actions are fireable. This is why the priority "read_done ≻ read" is important to keep the determinism of the actor. Finally, the firing of "read_done" action involves an FSM update to "state1" where only "process" action can be fired and the actor is back to the initial state.

3. Actor Behavior Formalism

Actor execution is governed by a set of conditions called firing rules. Moreover, during this firing many internal features of the actor are updated (state, state variables, etc.). All these concepts and behavior evolutions are detailed below. The actor execution, so called firing, is based on the dataflow Process Network (DPN) principle [12] derived from the Kahn Process Network (KPN) [13]. Let Ω be the universe of all tokens values exchanged by the actors and $\mathbb{S} = \Omega^*$, the set of all finite sequences in Ω. We denote the length of a sequence $s \in \mathbb{S}^k$ by $|s|$ and the empty sequence by λ. Considering an actor with m inputs and n outputs, $\mathbb{S}^m$ and $\mathbb{S}^n$ are the set of m-tuples and n-tuples consumed and produced. For example, $s_0 = [\lambda[t_0, t_1, t_2]]$ and $s_1 = [[t_0], [t_1]]$ are sequences of tokens that belong to $\mathbb{S}^2$ and we have $|s_0| = [0, 3]$ and $|s_1| = [1, 1]$.

3.1. Actor Firing. A dataflow actor is defined with a pair $\langle f, R \rangle$ such as:

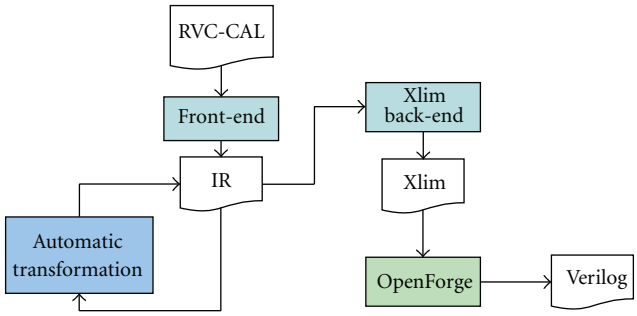

Figure 5: Automatic transformation localization in Orcc compiling process.

(i) $f : \mathbb{S}^m \rightarrow \mathbb{S}^n$ is the firing function;

(ii) $R \subset \mathbb{S}^m$ are the firing rules;

(iii) for all $r \in R$, $f(r)$ is finite.

An actor may have N firing rules which are finite sequences of m patterns (one for each input port). A pattern is an acceptable sequence of tokens for an input port. It defines the nature and the number of tokens necessary for the execution of at least one action. RVC-CAL also introduces the notion of *guard* as additional conditions on tokens values. An example of firing rule r_j in $\mathbb{S}^2$ is

$$\begin{aligned} g_{j,k} &: [x] \mid x > 0 \\ r_j &= \left[t_0 \in g_{j,k}, [t_1, t_2, t_3]\right], \end{aligned} \tag{1}$$

Equation (1) means that if there is a positive token in the FIFO of the first input port and 3 tokens in the FIFO of the second input port then the actor will select and execute a fireable action. An action is fireable or schedulable iff:

(i) the execution is possible in the current state of the FSM (if an FSM exists);

(ii) there are enough tokens in the input FIFO;

(iii) a guard condition returns true.

An action may be included in a finite state machine or untagged making it higher priority than FSM actions.

3.2. Actor Transition. The FSM transition system of an actor is defined with $\langle \sigma_0, \Sigma, \tau, \prec \rangle$ where Σ is the set of all the states of the actor, σ_0 is the initial state, $\prec$ is a priority relation and $\tau \subseteq \Sigma \times \mathbb{S}^m \times \mathbb{S}^n \times \Sigma$ is the set of all possible transitions. A transition from a state σ to a state σ' with a consumption of sequence $s \in \mathbb{S}^m$ and a produced sequence $s' \in \mathbb{S}^n$ is defined with (σ, s, s', σ') and denoted.

$$\sigma \xrightarrow[\tau]{s \mapsto s'} \sigma'. \tag{2}$$

To solve the problem of the existence of more than one possible transition in the same state, RVC-CAL introduced the notion of priority relation such as for the transitions $t_0, t_1 \in \tau$, t_0 a higher priority than t_1 is written $t_0 \succ t_1$. As explained in [24] a transition $\sigma \xrightarrow[\tau]{s \mapsto s'} \sigma'$ is enabled iff:

$$\neg \exists \sigma \xrightarrow[\tau]{p \mapsto q} \sigma'' \in \tau : p \in \mathbb{S} \wedge \sigma \xrightarrow[\tau]{s \mapsto s'} \sigma'' \succ \sigma \xrightarrow[\tau]{s \mapsto s'} \sigma'. \tag{3}$$

This section presented and explained the main RVC-CAL principles. In the next section we present an automatic transformation as a solution to avoid these limitations without changing the overall macrobehavior of the actor.

3.3. Hardware Generation Problematic. A firing rule is called *multitoken* iff : $\exists e \in |s| : e > 1$ otherwise it is called a *monotoken* rule. The limitation of OpenForge is the fact that it does not support multitoken rules which are omnipresent in most actors. The observation of Figure 4 shows the incontestable complexity difference between the multitoken (a) and the monotoken (b) code. Moreover, manually changing a CAL code from high-level to low-level by creating the new actions, variables and state machine is contradictory to the main purpose of RVC standard which is the fact that CAL is a target agnostic language so we have to write in CAL the same way for hardware of software implementation. Our work consists in automatically transforming the data read/write processes from multitoken to monotoken while preserving the same actor behavior. All the required actions, variables and finite state machines are created and optimized directly in the Intermediate representation of Orcc compiler. The following section explains the achieved transformation mechanism.

4. Methodology for Hardware Code Generation

As shown in Figure 5, our transformation acts on the IR of Orcc. The HDL implementation is later generated using OpenForge.

4.1. Actor Transformation Principle. Let us consider an actor with a multitoken firing rule $r \in \mathbb{S}^k$ such as $|r| = [r_0, r_1, \ldots, r_{k-1}]$, this rule fires a multitoken action a realizing the transition $source \xrightarrow[\tau]{a} target$ and $\mathbb{I}$ the set of all input ports. The transformation creates for every input port an internal buffer with read-and-write indexes and clips r into a set $\mathbb{R}$ of k firing rules so that:

$$\forall i \in \mathbb{I}, \exists! \rho \in \mathbb{R} : \begin{cases} \rho : \mathbb{S}^1 \longrightarrow \mathbb{S}^0 \\ |r| = 1 \\ g_\rho : IdxWrite_i - IdxRead_i \leq sz_i, \end{cases} \tag{4}$$

with ρ a monotoken firing rule of an untagged action $untagged_i$, g_ρ is the guard of ρ, and sz_i the size of the associated internal buffer defined as the closest power of 2 of r_i. This guard checks that the buffer contains an empty place for the token to read. The multitoken action is consequently removed, and new *read actions* that read one token from the internal buffers are created. While reading tokens another firing rule may be validated and causes the firing of an unwanted action. To avoid the nondeterminism of such a case, we use an FSM to put the actor in a reading loop so it can only read tokens. The loop is entered using a *transition*

```
actor A () int IN1, int IN2, int IN3 ==> int OUT1, int OUT2:
   a: action
   in1:[in1] repeat 2, IN2:[in2] repeat 3, IN3:[in3] ==>
   OUT1:[out1], OUT2:[out2] repeat 2
   do
      {treatment}
   end
end
```

Figure 6: RVC-CAL code of actor A.

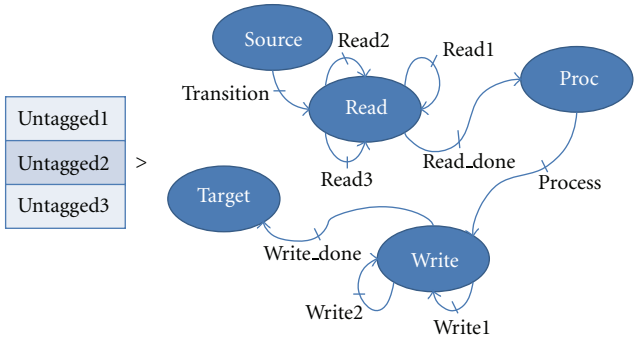

Figure 7: Created FSM macroblock.

action realizing the FSM passage $source \xrightarrow[\tau]{transition} read$ and has the same priority order of the deleted multitoken action but has no process. The read actions loop in the read state with the transition $t = read \xrightarrow[\tau]{read} read$. Then the loop is exited when all necessary tokens are read using a *read done* action and a transition to the process state $t' = read \xrightarrow[\tau]{read\ Done} process \succ t$. The treatment of the multitoken action is put in a *process action* with a transition $process \xrightarrow[\tau]{process} write$. The multitoken outputs are also transformed into a writing loop with *write actions* that store data directly in the output FIFO associated with a transition $w = write \xrightarrow[\tau]{write} write$ and a *write done* action that insures the FSM transition $w' = write \xrightarrow[\tau]{write\ Done} target \succ w$.

For example, the actor A of Figure 6 is defined with $f{:}\mathbb{S}^3 \to \mathbb{S}^2$ with a multitoken firing rule: $r \in \mathbb{S}^3 : r = [[t_0, t_1], [t_2, t_3, t_4], [t_5]]$.

Consequently, $|r| = [2, 3, 1]$ which means that there is an action in A that fires if 2 tokens are present in $IN1$ port, 3 tokens are present in $IN2$ and one token is present in $IN3$. The transformation creates the FSM macroblock of Figure 7.

4.2. FSM Creation Cases. We consider an example of an actor defined as $f : \mathbb{S}^3 \to \mathbb{S}^2$ containing the actions $a1 \cdots a5$ such as $a3$ is the only action applying a multitoken firing rule $r \in \mathbb{S}^3$.

Creating an FSM only for action $a3$ is not appropriate because $a1$, $a2$, $a4$, $a5$ will be a higher priority which may not be true. The solution is to create an initial state containing all the actions and add the created FSM macroblock of $a3$ (previously presented in Figure 7). The resulting FSM is presented in Figure 8.

We now suppose the same actor scheduled with an initial FSM as shown in Figure 9.

The transition $t = S1 \xrightarrow[\tau]{s \mapsto s'} S2$ is substituted with the macroblock of $a3$ as shown in Figure 10.

4.3. Optimizations. To improve the transformation, some optimization solutions were added. In the previously presented transformation method we used the untagged actions to store data in the internal buffers, then we used read actions to peek the required tokens from the internal buffers using R/W indexes and masks. To preserve the schedulability, the action is split into a transition action that contains the firing rule and a process action that applies the algorithm. The proposed optimization consists in making the action reading directly from the internal buffers. The firing rule of the action is transformed as presented in (4) to detect the presence of enough data in the internal buffers. Let us reconsider the basic example of the "sum-5" actor of Figure 4 of Section 2.2. The transformation explained above and the optimized transformation of this actor are presented in Figure 11. This actor is transformed this way. First an internal buffer and an untagged action are created to store data inside the actor. The input pattern of the *read* action is transformed into a connection to the internal buffer. Every read or write from the internal buffer must be masked to make the modulo of th buffer size since it is circular.

5. RVC Case of Study: MPEG 4 SP Intradecoder

To assess the performance of the previously presented transformation, we applied it on the whole MPEG 4 simple profile intradecoder. This choice is explained by the fact that there exists a stable design in the VTL and also because this decoder includes various image processing algorithms with more or less complexity. In the following we present an overview of this codec architecture and basic actors. We also present the implementation results and a comparison with an academic high-level synthesis tool called GAUT.

5.1. Concept. MPEG codecs have all a common design. It begins with a parser that extracts motion compensation and texture reconstruction data. The parser is then followed by reconstruction blocs for texture and motion and a merger as presented in Figure 12. This decoder is a full example of coding techniques that encapsulates predictions, scan, quantization, IDCT transform, buffering, interpolation, merging and especially the very complex step of parsing.

Table 1 gives an idea about the complexity of parsers in MPEG 4 Simple Profile and MPEG Advanced Video Coding (AVC).

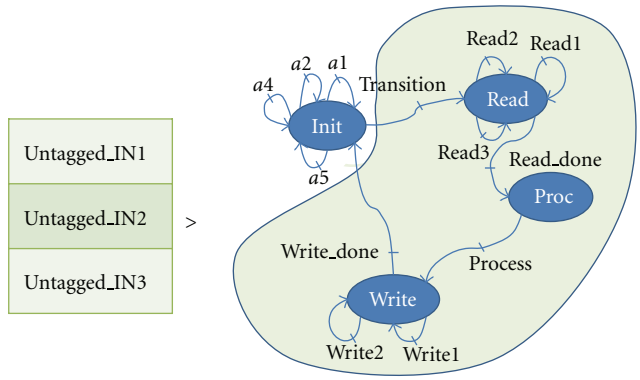

Figure 8: FSM with created initial state.

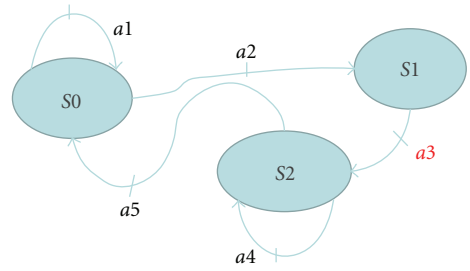

Figure 9: Initial FSM of an actor.

Table 1: Composition of MPEG-4 simple profile and MPEG-4 advanced video coding RVC-CAL description.

	Actors	Levels	Parser size kSLOC	Decoder size kSLOC
MPEG-4 SP	27	3	9.6	2.9
MPEG-4 AVC	45	6	19.8	3.9

Table 2: MPEG4 decoder area consumption.

Criterion	Transformed design	Optimized design
Slice flip flops	21,624/135,168 (15%)	13,575/135,168 (10%)
Occupied slices	45,574/67,584 (67%)	18,178/67,584 (26%)
4 input LUTs	68,962/135,168 (51%)	34,333/135,168 (25%)
FIFO16/RAMB16s	14/288 (4%)	14/288 (4%)
Bonded IOBs	107/768 (13%)	107/768 (13%)

Actors of Figure 12 are the main functional units some of them are hierarchical composition of actor networks. An actor may be instantiated more than one time so for 27 FU there are 42 actor instantiations.

5.2. Implementation and Results. The achieved automatic transformation was applied on MPEG4 SP intradecoder (see design in Orcc Applications (available at http://orcc.sourceforge.net/)) which contains 29 actors. We omitted the inter decoder part because it is very memory consuming. The HDL generated code was implemented on a virtex4 (xc4vlx160-12ff1148) and the area consumption results we obtained are presented in Table 2. The removal of read actions buffers and process actions had an important impact on the area consumption since it has decreased about 50%.

Table 3: MPEG4 decoder timing results.

Criterion	Transformed design	Optimized design
Maximum frequency (MHz)	26.4	26.67
Latency (μs)	381.8	306.4
Cadency (MHz)	1.9	2.33
Processing time (ms/image)	13.55	11.01
Throughput frequency (MHz)	1.8	2.2
Global image processing (FPS)	73.8	90.82

After the synthesis of the design, we applied a simulation stream of compressed videos. Table 3 below presents the timing results of a CIF (352×288) image size video.

We notice that timing results were partially improved. This is due to the presence of division operations in some actors. In our transformation we replaced divisions by an Euclidean division which is very costly and time consuming. The impact is noticeable since these divisions reduced the maximum frequency by 60%. Therefore, we applied the transformation on the inverse discrete cosine 2D transform (IDCT2D). We chose this actor because it contains very complex algorithm, functions and procedures. We tried to compare with an optimal low-level architecture designed by Xilinx experts and also with an existing implementation study of a direct VHDL written algorithm in [25]. For a significant comparison, we used the same implementation target

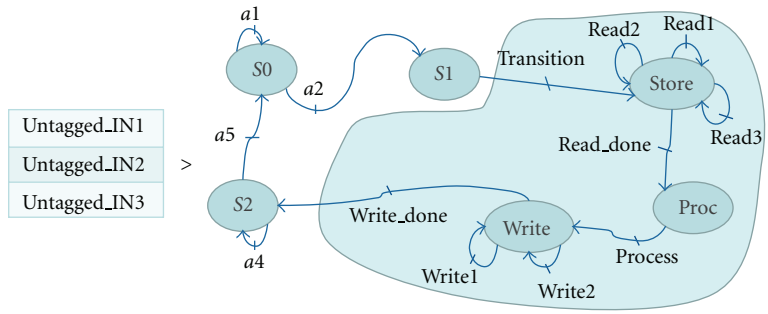

FIGURE 10: Resulting FSM transformation.

```
actor sum-5 () int (size=8) IN
==> int(size=8) OUT:

 List (type: int (size=8), size = 8) buffer;
 // closest power of 2 for circular buffer
 List (type: int (size=8), size = 5) data;
 int readIdx := 0;
 int writeIdx := 0;
 int counter :=0 ;

 action IN:[ i ] ==> // untagged action
 guard
   readIdx - writeIdx < 8
   // condition that the buffer is not full
 do
   buffer[readIdx & 7] := i ;
   // masked read index
   readIdx := readIdx + 1 ;
 end

 read: action ==>
 do
   data[counter] := buffer[writeIdx & 7] ;
   // masked write index
   counter := counter + 1 ;
 end

 read_done: action ==>
 guard
   counter = 5
 do
   counter := 0 ;
 end

 process: action ==> OUT:[ s ]
 var
  int s := 0
 do
   foreach int k in 0 .. 4 do
     s := s + data[k] ;
   end
   writeIdx := writeIdx + 5; // update writeIdx
 end

 schedule fsm state0:
   state0 (read)      --> state0;
   state0 (read_done) --> state1;
   state1 (process)   --> state0;
 end

 priority
   read_done > read;
 end

end
```

(a) Transformed equivalent CAL

```
actor sum-5 () int (size=8) IN
==> int(size=8) OUT:

 List (type: int (size=8), size = 8) buffer;
 int readIdx := 0;
 int writeIdx := 0;

 action IN:[ i ] ==>
 guard
   readIdx - writeIdx < 8
 do
   buffer[readIdx & 7] := i ;
   readIdx := readIdx + 1 ;
 end

 process: action ==> OUT:[ s ]
 guard
        readIdx - writeIdx > 5
 var
  int s := 0
 do
   foreach int k in 0 .. 4 do
     s := s + buffer[k + (writeIdx&7)] ;
   end
   writeIdx := writeIdx + 5; // update writeIdx
 end

end
```

(b) Optimized equivalent CAL

FIGURE 11: Transformed and optimized sum-5 actor.

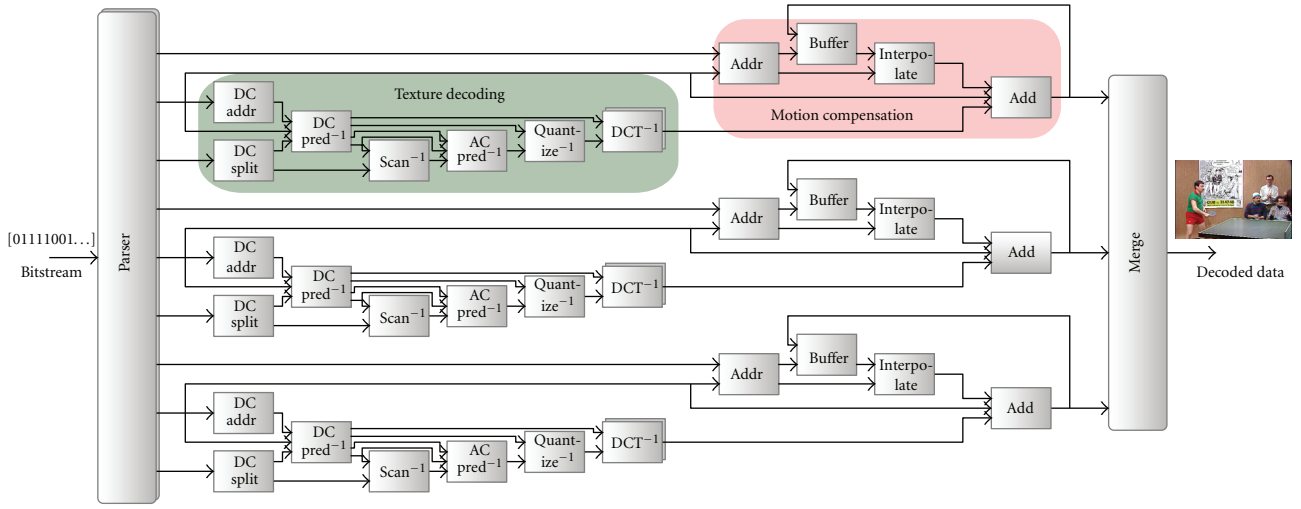

Figure 12: MPEG 4 SP architecture.

Table 4: IDCT2D timing results.

Image size	Xilinx design	Transformed design	Optimized design	VHDL design
Maximum frequency (MHz)	37	37	43	41
Latency (μs)	11.52	82.7	28.4	*
Cadency (MHz)	30	18.49	21.7	71
Processing time (μs/64 Tokens)	1.99	3.4	2.8	0.89
Throughput frequency (MHz)	26.62	0.72	2.43	62.4
Global image processing (FPS)	1064	31	101	2518

*Not mentioned in the literature.

of the study which is the Xilinx Spartan 3 XC3S4000. Timing and area consumption results comparison are presented in Tables 4 and 5.

Obviously, Table 5 reveals that area results for the optimized design are very close to those of the Xilinx low-level design. This property is noted for all actors containing more computing algorithms then data control and management algorithms. Concerning the area consumption of the VHDL design, it is expectable to find results nearby the optimal design and clearly worse than the Xilinx design and this is due to the synthesis constraints indicated in [25] that favor treatment speed in spite of the surface. This is what explains also the very high FPS rate of the design presented in Table 4. Timing results of the other designs show that the optimized design performances are far from the optimal Xilinx design. This is due to the low level architecture made by Xilinx experts which is completely different and oriented for hardware generation. This architecture is a pipelined set of actors realizing the IDCT2D (rowsort, fairmerge, IDCT1D, separate, transpose, retranspose, and clip) which is a relatively complex design compared with the high-level IDCT2D code used for the transformation.

After comparing with the Xilinx design and a VHDL directly written design, we compared our results with existing generation tools and we considered GAUT hardware generator. This tool is an academic high-level synthesizer from C to VHDL. It extracts the parallelism and creates a scheduled dependency graph made of elementary operators. Potentially, GAUT synthesizes a pipe-lined design with memory unit, communication interface and a processing unit. However, like most existing hardware generators, GAUT is not able to manage a system level design with very high complexity and a variety of processing algorithms. Moreover, there are so many restrictions on the C input code to have a functioning design. As it was impossible to test the whole MPEG 4 decoder we chose the IDCT2D algorithm to have a comparison with previously presented results.

The IDCT2D is so generated with GAUT and we obtained the results of Table 6 below.

Results show that the optimized transformation generates a better design even for the specific case of study of the IDCT2D.

6. Still Image Codec: LAR Case of Study

The LAR is a still-image coder [26] developed at the IETR/INSA of Rennes laboratory. It is based on the idea that the spatial coding can be locally dependent on the activity in the image. Thus, the higher the activity the lower the resolution is. This activity is dependent from the variation or the uniformity of the local luminance which can be detected using a morphological gradient. In the following, we detail

Table 5: IDCT2D area consumption.

Criterion	Xilinx design	Transformed design	Optimized design	VHDL design
Slice flip flops	1415/55296 (2%)	4002/55296 (7%)	2113/55296 (3%)	*
Occupied slices	1308/27648 (4%)	5238/27648 (18%)	2523/27648 (9%)	3571/27648 (12%)
4 input LUTs	2260/55296 (4%)	9861/55296 (17%)	4777/55296 (8%)	4640/55296 (8%)
Bonded IOBs	48/489 (9%)	49/489 (10%)	49/489 (10%)	*

*Not mentioned in the literature.

Table 6: IDCT2D area consumption with GAUT.

Criterion	GAUT design	Optimized design
Slice flip flops	2.080/135.168 (2%)	1.988/135.168 (2%)
Occupied slices	2.477/67.584 (3%)	2.353/67.584 (3%)
4 input LUTs	4.243/135.168 (3%)	4.458/135.168 (3%)
Bonded IOBs	627/768 (81%)	49/768 (6%)

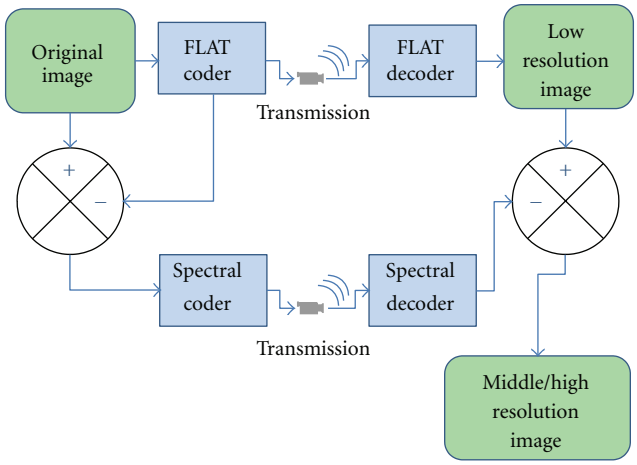

Figure 13: LAR concept.

coding principle of the LAR and we present the implementation techniques and results using the automatic transformation approach.

6.1. Concept. The LAR coding is based on considering that an image is a superposition of a global information image (mean blocks image), and the local texture image, which is given by the difference between the original image and the global one. This principle is modeled by

$$I = \overline{I} + \left(\underbrace{I - \overline{I}}_{E} \right), \tag{5}$$

where I is the original image, $\overline{I}$ is the global information image and $I - \overline{I}$ is the error image, E. The dynamic range of the error image is consequently dependent on the local activity. In uniform regions, $\overline{I}$ values are close or equal to I consequently $I - \overline{I}$ values are around zero with a low dynamic range.

Considering these principles, the LAR coder concept (Figure 13) is composed of two parts: the FLAT LAR [27] which is the part insuring the global information coding and the spectral part which is the error spectral coder.

Different profiles have been designed to fit with different types of application. In this paper, we focus on the baseline coder. Its mechanisms are detailed in the following.

The FLAT LAR. The Flat LAR is composed of 3 main parts: the partitioning, the block mean value computation and the DPCM (Differential Pulse Coding Modulation). In our work, only the DPCM is not yet developed with RVC-CAL.

(i) Partitioning: in this part, a Quad-tree partitioning is applied on the image pixels. The principle is to consider the lowest block size (2×2) then to compare the difference between the maximum (MAX) and the minimum (MIN) values of the block with a threshold (THD) defined as a generic variable for the design. If (MAX−MIN) > THD then the actual block size is considered. In the other case, the $(N\times2)\times(N\times2)$ size block is required. this process is recursively applied on the whole image blocks. The output of the overall is the block size image.

(ii) Block mean values computation process: this process is based on the Quad-tree output image. For each block of the variable size image, a mean value is put in the block as presented in the example of Figure 14.

(iii) The DPCM: the DPCM process is based on the prediction of neighbor values and the quantization of the block mean value image. The observation that a pixel value is mostly equal to a neighbor one led to the following estimation algorithm. If we consider the pixels in Figure 15, X value is estimated with the following algorithm:

If $|B - C| < |A - B|$ then $X = A$ else $X = C$.

The spectral coder, also called the texture coder, is composed of a variable block size Hadamard transform [28] and the Golomb-Rice [29, 30] entropy coder. The Golomb-Rice coder is still in development with the RVC-CAL specifications.

The Hadamard transform derives from a generalized class of the Fourier transform. It consists of a multiplication of a $2^m \times 2^m$ matrix by an Hadamard matrix (H_m) that has the same size. The transform is defined as follows.

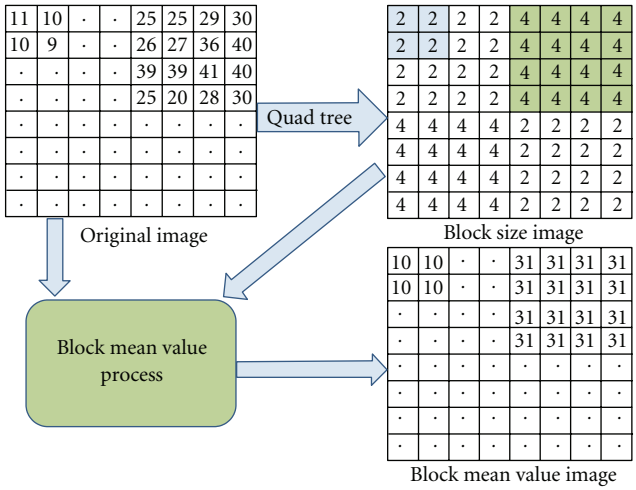

Figure 14: Block mean value process example.

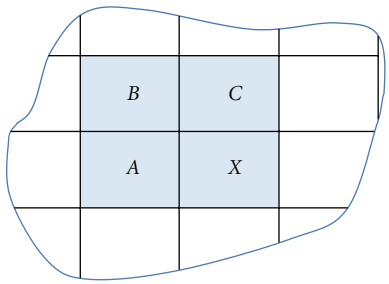

Figure 15: DPCM prediction of neighbor pixels.

H_0 is the identity matrix so $H_0 = 1$. For any $m > 0$, H_m is then deducted recursively by:

$$H_m = \frac{1}{\sqrt{2}} \begin{vmatrix} H_{m-1} & H_{m-1} \\ H_{m-1} & -H_{m-1} \end{vmatrix}. \quad (6)$$

Here are examples of Hadamard matrices:

$$H_0 = 1,$$

$$H_1 = \frac{1}{\sqrt{2}} \begin{vmatrix} 1 & 1 \\ 1 & -1 \end{vmatrix},$$

$$H_2 = \frac{1}{\sqrt{2}} \begin{vmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{vmatrix}, \quad \text{and so forth.} \quad (7)$$

6.2. Implementation and Results. This Section explains the mechanisms of the Hadamard transform and the Quad-tree used in the implementation.

6.2.1. Hardware Implementation. The LAR coding is dependent from the content of the image. It applies in the Quad-Tree a morphological gradient to extract information about the local activity on the image. The output is the block size image represented by variable size blocks: 2×2, 4×4 or 8×8. Using the block size image, the Hadamard transform applies the adequate transform on the corresponding block. It means that if we have a block size of 2×2 in the size image this block will undergo a 2×2 Hadamard (H_1) and a normalization specific to the 2×2 blocks. This process is identically applied for 4×4 and 8×8 blocks. A quantization step, adapted to current block size, is applied on the Hadamard output image. For each block size, a quantization matrix is predefined. Practically, the normalization during the Hadamard transform is postponed to be achieved with the quantization step so that to decrease the noise due to successive divisions.

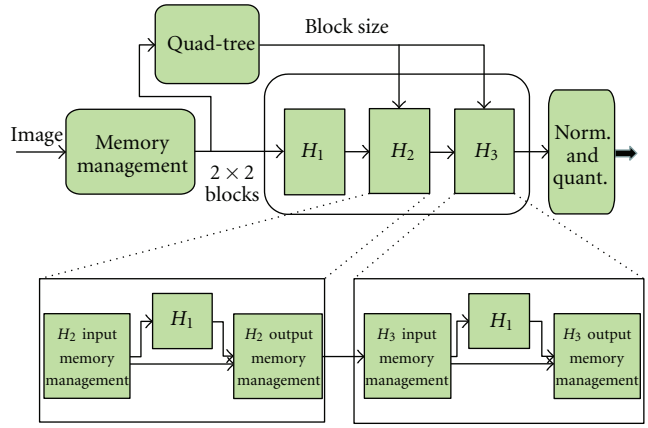

Figure 16: LAR baseline developed model.

The implemented LAR is presented in Figure 16.

As a first step, the memory management block stores the pixels values of the original image line by line. Once an 8×8 block is obtained, the actor divides it into sixteen 2×2 blocks and sends them in a specific order as presented in Figure 18.

This order is very important to improve the performance of remaining actors. In fact, considering the Figure 18, when the tokens are so ordered the first 4 tokens correspond to the first 2×2 block, the first 16 tokens to the first 4×4 block, and so forth Consequently, and as presented in Figure 16, the output of the H_1 is automatically the input of the H_2 and the output of the H_2 is automatically the input of the H_3.

In the Quad-tree, this order is also crucial. As presented in Figure 17, the superposition of the same actor (max for example) three times provides in the output of the first actor the maximum values of 2×2 blocks, in the output of the second actor the maximum values of 4×4 block and finally the maximum values of 8×8 blocks in the output of the third one. Using the maximum values and the minimums the morphological gradient in the Gradstep actors can process to extract the block size image. The same tip is used to calculate the block sums with three superposed sum actors. The block mean value actor considers the sums and the sizes to build the block mean value image.

We also notice that an (H_2) transform can be achieved using the (H_1) results of the four 2×2 blocks constituting the 4×4 block. The same observation can be made for the (H_3) one. This ascertainment is very important to decrease the complexity of the process. In fact, the Hadamard transform

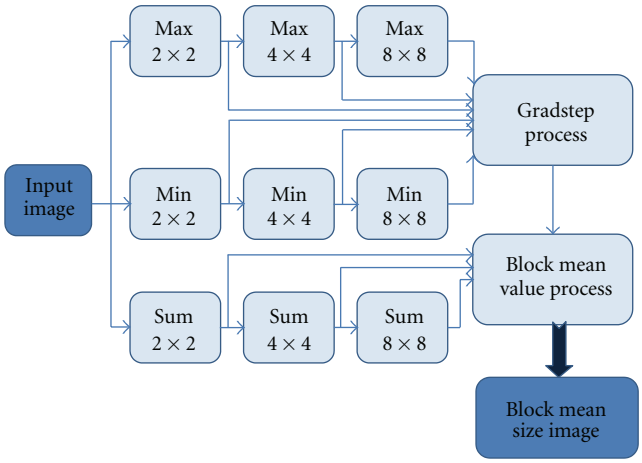

Figure 17: Quad-tree design.

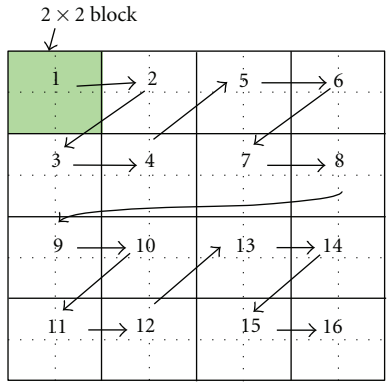

Figure 18: Memory management unit output order.

of the LAR applies an (H_1) transform for the whole image then it applies the (H_2) transform only for the 4×4 and 8×8 blocks and the (H_3) transform only for the 8×8 blocks. The (H_2) and the (H_3) transforms are different from the full transforms as they are much less complex. Consequently, as shown in Figure 16, we designed the H_2 and the H_3 using H_1 actors associated with memory management units. They sort tokens in the adequate order and, considering the block size, whether the block is going to undergo the transform or not.

It is very important to mention that almost actors have been developed with generic variables for memory sizes or gradsteps which means that the design are flexible for easy transformation from an image size to another or for adding higher Hadamard process (H_4, H_5, etc.).

In [15], we added some optimizations on the processes using a Ping-Pong memory management algorithm [31] to pipeline the process.

6.2.2. Results and Comparison. As mentioned above, this work aims at comparing hardware implementation performances of the same LAR architecture generated with the optimized automatic transformation and with a manual transformation. The achieved automatic transformation was applied on the 23 actors of the LAR using Orcc. The HDL generated code was implemented on a virtex4 (xc4vlx160-12ff1148). The area consumption results obtained are presented with those of manual transformations in Table 7.

Table 7: LAR coder area consumption.

Transformation	Automatic	Manual
Slice flip flops	20.452/135.168 (15%)	12.157/135.168 (8%)
Occupied slices	47.576/67.584 (70%)	43.602/67.584 (67%)
4 input LUTs	59.868/135.168 (44%)	53.417/135.168 (39%)
Bonded IOBs	41/768 (5%)	41/768 (5%)

Table 8: LAR timing results.

Transformation	Automatic	Manual
Development time	30%	100%
Maximum frequency (MHz)	61.43	85.27
Latency (ms)	0.42	0.12
Throughput frequency (MHz)	3.5	5.6
Processing time (ms/image)	35	19
Global image processing (FPS)	34	53

After the synthesis of the design, we applied a simulation stream of compressed videos. Table 8 below presents the timing results of a CIF (352×288) image size video.

For area consumption, the difference is not considerable for LUTs and occupied slices and it can be explained by the fact that the transformation applies a general modification whatever the complexity of the actor. Also, the fact of creating an internal buffer for every input port involves more area consumption.

Concerning the timing results, the automatic and the manual transformed designs performances remain close and acceptable. The latency difference is explained by the fact that the untagged actions, as always given priority over the rest of actions, promote the data reading. It means that, as long as there is data in the FIFO, the untagged action fires even if there are enough data to fire the processing actions. This problem will also be resolved by further optimizations of the buffer size.

7. Conclusion

This paper presented an automatic transformation of RVC-CAL from high- to low-level description. The purpose of this work is to find a general solution to automate the whole hardware generation flow from system level. This transformation allows avoiding structures that are not understandable by RVC-CAL hardware compilers. We applied this automatic transformation on the 29 actors of MPEG4 part2 video intradecoder and successfully obtained the same behavior of the multitoken design and a synthesizable hardware implementation. To change the test context, we automatically transformed a high-level design of the LAR still image codec and obtained relatively acceptable results.

Several optimization processes were added to the transformation to reduce the area consumption about 50%. The transformation process is currently generalized for all actors.

The most important in this work is that we contributed in making RVC-CAL hardware generation very rapid with an average gain of 75% of conception, development, and validation time compared with manual approach. We insured that the generation is applicable at system level whatever the complexity of the actor.

Currently, improvements are also in progress to customize the transformation depending on the actor complexity analysis. A future work will be the study of the impact of the transformation on the power consumption of the generated implementation.

Acknowledgments

Special thanks to Matthieu Wipliez, Damien De Saint-Jorre, and Hervé Yviquel for their relevant contributions in the source code.

References

[1] B. Bailey, G. Martin, and A. Piziali, *ESL Design and Verification: A Prescription for Electronic System-Level Methodology*, The Morgan Kaufmann Series in Systems on Silicon, Morgan Kaufmann, 2007.

[2] "Synopsys: Synphony C compiler," In ESL design and verification: a prescription for electronic system-level methodology, http://www.synopsys.com/systems/blockDesign/HLS/pages/SynphonyC-Compiler.aspx

[3] "Mentor Graphics: Designing High-Performance DSP Hardware Using Catapult C Synthesis and the Altera Accelerated Libraries," In ESL design and verification: a prescription for electronic system-level methodology, http://www.altera.com/literature/wp/wp-01039.pdf.

[4] "Mentor Graphics: Catapult C," In ESL design and verification: a prescription for electronic system-level methodology, 2010, http://www.mentor.com/esl/catapult/overview.

[5] F. Plavec, Z. Vranesic, and S. Brown, "Towards compilation of streaming programs into FPGA hardware," in *Proceedings of the Forum on Specification, Verification and Design Languages (FDL '08)*, pp. 67–72, September 2008.

[6] D. Lau, O. Pritchard, and P. Molson, "Automated generation of hardware accelerators with direct memory access from ANSI/ISO standard C functions," in *Proceedings of the 14th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM '06)*, pp. 45–56, April 2006.

[7] T. M. Bhatt and D. McCain, "Matlab as a development environment for FPGA design," in *Proceedings of the 42nd annual Design Automation Conference (DAC '05)*, ACM, New York, NY, USA, 2005.

[8] P. Coussy, D. D. Gajski, M. Meredith, and A. Takach, "An introduction to high-level synthesis," *IEEE Design and Test of Computers*, vol. 26, no. 4, pp. 8–17, 2009.

[9] P. Coussy, C. Chavet, P. Bomel, D. Heller, E. Senn, and E. Martin, "GAUT: a high-level synthesis tool for DSP applications," in *High-Level Synthesis: From Algorithm to Digital Circuits*, P. C. A. Morawiec, Ed., Springer, 2008.

[10] J. Eker and J. Janneck, "CAL language report," ERL Technical Memo UCB/ERL M03/48, University of California at Berkeley, 2003.

[11] C. Brooks, E. Lee, X. Liu, S. Neuendorer, and Y. Zhao, Eds., "HZ: PtolemyII—heterogeneous concurrent modeling and design in Java (Volume 1: introduction to ptolemyII)," Technical Memorandum UCB/ERL M04/27, University of California, Berkeley, Calif, USA, 2004.

[12] E. A. Lee and T. M. Parks, "Dataow process networks," *Proceedings of the IEEE*, vol. 83, no. 5, Article ID 773801, 1995.

[13] G. Kahn, in *Proceedings of the IFIP Congress The Semantics of a Simple Language for Parallel Programming. In Information Processing*, J. L. Rosenfeld, Ed., pp. 471–475, North-Holland, New York, NY, USA, 1974.

[14] M. Mattavelli, I. Amer, and M. Raulet, "The reconfigurable video coding standard," *IEEE Signal Processing Magazine*, vol. 27, no. 3, pp. 159–167, 2010.

[15] K. Jerbi, M. Wipliez, M. Raulet, M. Babel, O. Deforges, and M. Abid, "Automatic method for efficient hardware implementation from rvc-cal dataflow: a lar coder baseline case study," *Journal Of Convergence*, vol. 1, Article ID 8592, 2010.

[16] K. Jerbi, M. Raulet, O. Deforges, and M. Abid, "Automatic generation of synthesizable hardware implementation from high level RVC-CAL design," in *Proceedings of the 37th International Conference on Acoustics Speech and Signal Processing (ICASSP '12)*, pp. 1597–1600, 2012.

[17] J. Janneck, I. Miller, D. Parlour, G. Roquier, M. Wipliez, and M. Raulet, "Synthesizing hardware from dataflow programs: an mpeg-4 simple profile decoder case study," *Journal of Signal Processing Systems*, vol. 63, no. 2, pp. 241–249, 2009.

[18] M. Mattavelli, J. W. Janneck, and M. Raulet, "MPEG reconfigurable video coding," in *Handbook of Signal Processing Systems*, S. S. Bhattacharyya, E. F. Deprettere, R. Leupers, and J. Takala, Eds., pp. 43–67, Springer, 2010.

[19] J. W. Janneck, M. Mattavelli, M. Raulet, and M. Wipliez, "Reconfigurable video coding: a stream programming approach to the specification of new video coding standards," in *Proceedings of the ACM SIGMM Conference on Multimedia Systems (MMSys '10)*, pp. 223–234, New York, NY, USA, February 2010.

[20] S. Bhattacharyya, G. Brebner, J. Eker et al., "OpenDF—a dataflow toolset for reconfigurable hardware and multicore systems," in *1st Swedish Workshop on MultiCore Computing (MCC '08)*, Ronneby, Sweden, November 2008.

[21] R. Gu, J. W. Janneck, S. S. Bhattacharyya, M. Raulet, M. Wipliez, and W. Plishker, "Exploring the concurrency of an MPEG RVC decoder based on dataflow program analysis," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 19, no. 11, pp. 1646–1657, 2009.

[22] G. Roquier, M. Wipliez, M. Raulet, J. W. Janneck, I. D. Miller, and D. B. Parlour, "Automatic software synthesis of dataflow program: an MPEG-4 simplpe profile decoder case study," in *Proceedings of IEEE Workshop on Signal Processing Systems (SiPS '08)*, pp. 281–286, Washington, DC, USA, October 2008.

[23] M. Wipliez, G. Roquier, and J. F. Nezan, "Software code generation for the RVC-CAL language," *Journal of Signal Processing Systems*, vol. 63, no. 2, pp. 203–213, 2011.

[24] J. Eker and J. W. Janneck, "A structured description of dataflow actors and its application," Technical Memorandum UCB/ERL M03/13, Electronics Research Laboratory, University of California at Berkeley, 2003.

[25] R. K. Megalingam, K. B. Venkat, S. V. Vineeth, M. Mithun, and R. Srikumar, "Hardware implementation of low power, high speed DCT/IDCT based digital image watermarking," in *Proceedings of the International Conference on Computer Technology and Development (ICCTD '09)*, pp. 535–539, November 2009.

[26] O. Déforges, M. Babel, L. Bédat, and J. Ronsin, "Color LAR codec: a color image representation and compression scheme based on local resolution adjustment and self-extracting

region representation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 17, no. 8, pp. 974–987, 2007.

[27] O. Deforges and M. Babel, "LAR method: from algorithm to synthesis for an embedded low complexity image coder," in *Inproceedings of the 3rd International Design and Test Workshop (IDT '08)*, pp. 187–192, December 2008.

[28] J. Poncin, "Utilisation de la transformation de Hadamard pour le codage et la compression de signaux d'images," *Annales des Télécommunications*, vol. 26, no. 7-8, pp. 235–252, 1971.

[29] R. F. Rice, "Some practical universal noiseless coding techniques," Technical Report 79–22, 1979.

[30] S. W. Golomb, "Run length codings," *IEEE Transactions on Information Theory*, vol. 12, no. 7, Article ID 399401, 1966.

[31] C. H. Chang, M. H. Chang, and W. Hwang, "A flexible two-layer external memory management for H.264/AVC decoder," in *Inproceedings of the 20th Anniversary IEEE International SOC Conference*, pp. 219–222, September 2007.

Hindawi Publishing Corporation
VLSI Design
Volume 2012, Article ID 752024, 13 pages
doi:10.1155/2012/752024

Research Article

N Point DCT VLSI Architecture for Emerging HEVC Standard

Ashfaq Ahmed, Muhammad Usman Shahid, and Ata ur Rehman

Department of Electronics & Telecommunication, Politecnico di Torino, 10129 Torino, Italy

Correspondence should be addressed to Ashfaq Ahmed, ashfaq.ahmed@polito.it

Received 21 December 2011; Revised 13 April 2012; Accepted 6 May 2012

Academic Editor: Andrey Norkin

This work presents a flexible VLSI architecture to compute the N-point DCT. Since HEVC supports different block sizes for the computation of the DCT, that is, 4×4 up to 32×32, the design of a flexible architecture to support them helps reducing the area overhead of hardware implementations. The hardware proposed in this work is partially folded to save area and to get speed for large video sequences sizes. The proposed architecture relies on the decomposition of the DCT matrices into sparse submatrices in order to reduce the multiplications. Finally, multiplications are completely eliminated using the lifting scheme. The proposed architecture sustains real-time processing of 1080P HD video codec running at 150 MHz.

1. Introduction

As the technology is evolving day by day, the size of hardware is shrinking with an increase of the storage capacity. High-end video applications have become very demanding in our daily life activities, for example, watching movies, video conferencing, creating and saving videos using high definition video cameras, and so forth. A single device can support all the multimedia applications which seemed to be dreaming before, for example, new high-end mobile phones and smart phones. As a consequence, new highly efficient video coders are of paramount importance. However, high efficiency comes at the expense of computational complexity. As pointed out in [1, 2], several blocks of video codecs, including the transform stage [3], motion estimation and entropy coding [4], are responsible for this high complexity. As an example the discrete-cosine-transform (DCT), that is used in several standards for image and video compression, is a computation intensive operation. In particular, it requires a large number of additions and multiplications for direct implementation.

HEVC, the brand new and yet-to-release video coding standard, addresses high efficient video coding. One of the tools employed to improve coding efficiency is the DCT with different transform sizes. As an example, the 16-point DCT of HEVC is shown in [5]. In video compression, the DCT is widely used because it compacts the image energy at the low frequencies, making easy to discard the high frequency components. To meet the requirement of real-time processing, hardware implementations of 2-D DCT/inverse DCT (IDCT) are adopted, for example, [6]. The 2-D DCT/IDCT can be implemented with the 1-D DCT/IDCT and a transpose memory in a row-column decomposition manner. In the direct implementation of DCT, float-point multiplications have to be tackled, which cause precision problems in hardware. Hence, we propose a Walsh-Hadamard transform-based DCT implementation [7]. Then, inspired by the DCT factorizations proposed in [8, 9], we factorize the remaining rotations into simpler steps through the lifting scheme [10]. The resulting lifting scheme-based architecture, inspired by [11–13], is simplified, exploiting the techniques proposed in [9, 14] to achieve a multiplierless implementation. Other techniques can be employed to achieve multiplierless solutions, such as the ones proposed in [8, 15–18], but they are not discussed in this work. In this work, the proposed multisize DCT architecture supports all the block sizes of HEVC and is proposed for the real-time processing of 1080P HD video sequences.

The rest of this paper is organized as follows. Section 2 provides reviews of 2-D DCT transform. Section 3 shows the matrix decompositions for different DCT sizes. Section 4 presents the proposed hardware architecture. The VLSI implementation and the simulation results in Section 5. Finally, Section 6 concludes this paper.

2. Review of 2D Transform

According to [19], the DCT in the so-called II-form for an N-point block of samples $x = \{x_0, x_1, \ldots, x_{N-1}\}$ is obtained as

$$X_k = \sqrt{\frac{2}{N}}\epsilon_k \sum_{n=0}^{N-1} x(n) \cdot \cos\left[\frac{\pi(2n+1)k}{2N}\right], \quad (1)$$

where

$$\epsilon_k = \begin{cases} \frac{1}{\sqrt{2}}, & \text{if } k = 0, \\ 1, & \text{otherwise} \end{cases} \quad (2)$$

and $k = 0, 1, 2, \ldots, N-1$. The same result expressed in (1) can be rewritten as the product of a matrix (C_N^{II}) for x as follws:

$$X = C_N^{II} \cdot x^t, \quad (3)$$

where $(\cdot)^t$ is the transposition operator. The DCT matrix can be expressed in terms of a reduced number of angles by exploiting the symmetry properties of the trigonometric functions. For 16 points, DCT matrix can be represented as

$$C_{16}^{II} = \frac{1}{2\sqrt{2}} \cdot \hat{C}_{16}^{II}, \quad (4)$$

where $\hat{C}_{16}^{II}$ is shown in (20), with

$$c_{p,q} = \cos\left(\frac{p \cdot \pi}{2^q}\right), \qquad s_{p,q} = \sin\left(\frac{p \cdot \pi}{2^q}\right), \quad (5)$$

where $4 \le 2^q \le 2N$ and p is an odd integer such that $p < 2^{q-2}$.

In [20], it is shown that every even-odd transform can be represented in terms of any other evenodd transform through a conversion matrix. In particular, in [7] it is shown that the DCT can be expressed in terms of the Walsh-Hadamard transform (WHT) [21], and the conversion matrix has a block diagonal structure. In [22], it is shown that WHT can be implemented with a fast algorithm based on a butterfly structure. Among the possible WHT matrix representations, the Walsh-ordered one is applied by deriving it from the corresponding Hadamard-ordered matrix. The N-order Hadamard matrix can be expressed as

$$H_N = \begin{bmatrix} H_{N/2} & H_{N/2} \\ H_{N/2} & -H_{N/2} \end{bmatrix}, \quad (6)$$

where $H_1 = 1$. The corresponding Walsh matrix W_N is obtained by applying a two step procedure [23] as follows:

(1) bit reverse the order of the rows of H_N,

(2) gray coding to the row indices.

Therefore, (4) can be written as

$$C_N^{II} = \frac{1}{\sqrt{N}} \cdot B_N \cdot T_N \cdot B_N \cdot W_N, \quad (7)$$

where B_N is the N-point bit reversal matrix, and

$$T_N = \begin{bmatrix} T_{N/2} & 0 \\ 0 & U_{N/2} \end{bmatrix} \quad (8)$$

is a block diagonal matrix with a recursive structure, $T_2 = I_2$ is the 2×2 identity matrix and

$$U_2 = \begin{bmatrix} c_{1,3} & s_{1,3} \\ -s_{1,3} & c_{1,3} \end{bmatrix}. \quad (9)$$

It is worth noting that $U_{N/2}$ can be factorized in terms of Givens rotations, where the Givens rotation matrix for a rotation angle θ is

$$\begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}. \quad (10)$$

In particular $U_{N/2}$ can be decomposed in the product of permutation matrices $P_{N/2}$ and Givens rotation matrices $V_{N/2,q}$ with $3 \le q \le m$ and $m = \log_2(N) + 1$ as

$$U_{N/2} = P_{N/2} \cdot V_{N/2,m} \cdot \ldots \cdot V_{N/2,q} \cdot \ldots \cdot V_{N/2,3} \cdot P_{N/2}. \quad (11)$$

The permutation matrix $P_{N/2}$ is obtained by applying the following permutation:

$$\Phi_{N/2} = \begin{pmatrix} 0 & 1 & \cdots & \frac{N}{2} - 1 \\ \phi_{N/2}(0) & \phi_{N/2}(1) & \cdots & \phi_{N/2}\left(\frac{N}{2} - 1\right) \end{pmatrix} \quad (12)$$

to the rows or to the columns of $I_{N/2}$, the $N/2 \times N/2$ identity matrix. It is worth observing that $\phi_{N/2}(x)$ can be defined recursively as

$$\phi_{N/2}(x) = \begin{cases} 2\phi_{N/4}(x), & 0 \le x \le \frac{N}{4} - 1, \\ 2\phi_{N/4}(x) + 1, & \frac{N}{4} \le x \le N - 1, \end{cases} \quad (13)$$

where $\phi_2(0) = 0$ and $\phi_2(1) = 1$.

The Givens rotations matrices can be described as follows: $V_{N/2,m}$ contains $N/4$ Givens rotations disposed in $N/4$ concentric squares with the rotation angle increasing from the outer square to the inner one. The definition of each $c_{p,q}$ is given in (5). In the outermost circle $p = 1$, whereas the value of p increases from the outer square to the inner one. Since p is the multiplied value in the numerator of (5), the angle of the Givens rotations increases from outer to inner squares. The general structure of $V_{N/2,m}$ is shown in (15). In the next sections, the expression shown in (15) will be detailed for different values of N

$$V_{N/2,m} = \begin{pmatrix} c_{1,m} & 0 & 0 & \cdots & 0 & 0 & s_{1,m} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \cdots & 0 & c_{p,m} & \cdots & s_{p,m} & 0 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \cdots & 0 & -s_{p,m} & \cdots & c_{p,m} & 0 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ s_{1,m} & 0 & 0 & \cdots & 0 & 0 & c_{1,m} \end{pmatrix}. \quad (14)$$

The remaining matrices $V_{N/2,q}$, for $q \geq 3$, are

$$V_{N/2,q} = \begin{pmatrix} V_{N/2^r,q} & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & V_{N/2^r,q} \end{pmatrix}, \tag{15}$$

where $r = m - q + 1$ and $V_{2,3} = U_2$. Finally, each Givens rotation can be factorized into lifting steps by the means of the lifting scheme [10] as suggested in [9] as follows:

$$\begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix} = \begin{pmatrix} 1 & \frac{1-\cos\theta}{\sin\theta} \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 \\ -\sin\theta & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & \frac{1-\cos\theta}{\sin\theta} \\ 0 & 1 \end{pmatrix}. \tag{16}$$

3. Matrix Decompositions for Different N

Matrices for different DCT are derived using the factorization presented in Section 2. In the following paragraphs factorizations for N ranging from 4 to 32 are explicitly shown.

3.1. 4×4 *DCT.* Equation (6) can be given as

$$H_4 = \begin{pmatrix} H_2 & H_2 \\ H_2 & -H_2 \end{pmatrix}. \tag{17}$$

Equation (7) can be given as

$$C_4^{II} = \frac{1}{2} \cdot B_4 \cdot T_4 \cdot B_4 \cdot W_4, \tag{18}$$

where

$$T_4 = \begin{pmatrix} I_2 & 0 \\ 0 & U_2 \end{pmatrix}, \tag{19}$$

where U_2 is shown in (9) and I_2 is a 2×2 identity matrix. W_4 is a Walsh-Hadamard matrix which is calculated using (17)

$$\widehat{C}_{16}^{II} = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ c_{1,5} & c_{3,5} & c_{5,5} & c_{7,5} & s_{7,5} & s_{5,5} & s_{3,5} & s_{1,5} & -s_{1,5} & -s_{3,5} & -s_{5,5} & -s_{7,5} & -c_{7,5} & -c_{5,5} & -c_{3,5} & -c_{1,5} \\ c_{1,4} & c_{3,4} & s_{3,4} & s_{1,4} & -s_{1,4} & -s_{3,4} & -c_{3,4} & -c_{1,4} & -c_{1,4} & -c_{3,4} & -s_{3,4} & -s_{1,4} & s_{1,4} & s_{3,5} & c_{3,4} & c_{1,4} \\ c_{3,5} & s_{7,5} & s_{1,5} & -s_{5,5} & -c_{5,5} & -c_{1,5} & -c_{7,5} & -s_{3,5} & s_{3,5} & c_{7,5} & c_{1,5} & c_{5,5} & s_{5,5} & -s_{1,5} & -s_{7,5} & -c_{3,5} \\ c_{1,3} & s_{1,3} & -s_{1,3} & -c_{1,3} & -c_{1,3} & -s_{1,3} & s_{1,3} & c_{1,3} & c_{1,3} & s_{1,3} & -s_{1,3} & -c_{1,3} & -c_{1,3} & -s_{1,3} & s_{1,3} & c_{1,3} \\ c_{5,5} & s_{1,5} & -c_{7,5} & -c_{3,5} & -s_{3,5} & s_{7,5} & c_{1,5} & s_{5,5} & -s_{5,5} & -c_{1,5} & -s_{7,5} & s_{3,5} & c_{3,5} & c_{7,5} & -s_{1,5} & -c_{5,5} \\ c_{3,4} & -s_{1,4} & -c_{1,4} & -s_{3,4} & s_{3,4} & c_{1,4} & s_{1,4} & -c_{3,4} & -c_{3,4} & s_{1,4} & c_{1,4} & s_{3,4} & -s_{3,4} & -c_{1,4} & -s_{1,4} & c_{3,4} \\ c_{7,5} & -s_{5,5} & -c_{3,5} & -s_{1,5} & c_{1,5} & s_{3,5} & -c_{5,5} & -s_{7,5} & s_{7,5} & c_{5,5} & -s_{3,4} & -c_{1,5} & -s_{1,5} & c_{3,5} & s_{5,5} & -c_{7,5} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ s_{7,5} & -c_{5,5} & -s_{3,5} & c_{1,5} & -s_{1,5} & -c_{3,5} & s_{5,5} & c_{7,5} & -c_{7,5} & -s_{5,5} & c_{3,5} & s_{1,5} & -c_{1,5} & s_{3,5} & c_{5,5} & -s_{7,5} \\ s_{3,4} & -c_{1,4} & s_{1,4} & c_{3,4} & -c_{3,4} & -s_{1,4} & c_{1,4} & -s_{3,4} & -s_{3,4} & c_{1,4} & -s_{1,4} & -c_{3,4} & c_{3,4} & s_{1,4} & -c_{1,4} & s_{3,4} \\ s_{5,5} & -c_{1,5} & s_{7,5} & s_{3,5} & -c_{3,5} & c_{7,5} & s_{1,5} & -c_{5,5} & c_{5,5} & -s_{1,5} & -c_{7,5} & c_{3,5} & -s_{3,5} & -s_{7,5} & c_{1,5} & -s_{5,5} \\ s_{1,3} & -c_{1,3} & c_{1,3} & -s_{1,3} & -s_{1,3} & c_{1,3} & -c_{1,3} & s_{1,3} & s_{1,3} & -c_{1,3} & c_{1,3} & -s_{1,3} & -s_{1,3} & c_{1,3} & -c_{1,3} & s_{1,3} \\ s_{3,5} & -c_{7,5} & c_{1,5} & -c_{5,5} & s_{5,5} & s_{1,5} & -s_{7,5} & c_{3,5} & -c_{3,5} & s_{7,5} & -s_{1,5} & -s_{5,5} & c_{5,5} & -s_{1,5} & c_{7,5} & -s_{3,5} \\ s_{1,4} & -s_{3,4} & c_{3,4} & -c_{1,4} & c_{1,4} & -c_{3,4} & s_{3,4} & -s_{1,4} & -s_{1,4} & s_{3,4} & -c_{3,4} & c_{1,4} & -c_{1,4} & c_{3,4} & -s_{3,4} & s_{1,4} \\ s_{1,5} & -s_{3,5} & s_{5,5} & -s_{7,5} & c_{7,5} & -c_{5,5} & c_{3,5} & -c_{1,5} & c_{1,5} & -c_{3,5} & c_{5,5} & -c_{7,5} & s_{7,5} & -s_{5,5} & s_{3,5} & -s_{1,5} \end{bmatrix} \tag{20}$$

3.2. 8×8 *DCT.* Equation (6) can be given as

$$H_8 = \begin{pmatrix} H_4 & H_4 \\ H_4 & H_4 \end{pmatrix}. \tag{21}$$

Equation (7) can be given as

$$C_8^{II} = \frac{1}{2\sqrt{2}} \cdot B_8 \cdot T_8 \cdot B_8 \cdot W_8, \tag{22}$$

where

$$T_8 = \begin{pmatrix} I_2 & 0 & 0 \\ 0 & U_2 & 0 \\ 0 & 0 & U_4 \end{pmatrix}, \tag{23}$$

where U_2 is shown in (9) and I_2 is a 2×2 identity matrix. W_8 is a Walsh-Hadamard matrix which is calculated using (21). According to (11), $m = \log_2(8) + 1 = 4$ and $3 \leq q \leq 4$. So U_4 can be written as

$$U_4 = P_4 \cdot V_{4,4} \cdot V_{4,3} \cdot P_4, \tag{24}$$

where, according to (14)

$$V_{4,4} = \begin{pmatrix} c_{1,4} & 0 & 0 & s_{1,4} \\ 0 & c_{3,4} & s_{3,4} & 0 \\ 0 & -s_{3,4} & c_{3,4} & 0 \\ -s_{1,4} & 0 & 0 & c_{1,4} \end{pmatrix} \tag{25}$$

and, according to (15), $r = m - q + 1 = 4 - 3 + 1 = 2$ and $V_{N/2^2,q} = V_{8/4,3} = V_{2,3} = U_2$, so

$$V_{4,3} = \begin{pmatrix} U_2 & 0 \\ 0 & U_2 \end{pmatrix}. \tag{26}$$

Using (12) and (13) the permutation is computed as

$$\Phi_4 = \begin{pmatrix} 0 & 1 & 2 & 3 \\ 0 & 2 & 1 & 3 \end{pmatrix}. \tag{27}$$

Finally the permutation matrix P_4 is obtained by applying the permutation, shown in (27), to the columns of 4×4 identity matrix as,

$$P_4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{28}$$

3.3. 16×16 *DCT.* Equation (6) can be given as

$$H_{16} = \begin{pmatrix} H_8 & H_8 \\ H_8 & -H_8 \end{pmatrix}. \tag{29}$$

Equation (7) can be given as

$$C_8^{II} = \frac{1}{4} \cdot B_{16} \cdot T_{16} \cdot B_{16} \cdot W_{16}, \tag{30}$$

where

$$T_{16} = \begin{pmatrix} I_2 & 0 & 0 & 0 \\ 0 & U_2 & 0 & 0 \\ 0 & 0 & U_4 & 0 \\ 0 & 0 & 0 & U_8 \end{pmatrix}, \tag{31}$$

where U_2 and U_4 are shown in (9) and (24). To calculate U_8, according to (11), $m = \log_2(16) + 1 = 5$ and $3 \le q \le 5$. So U_8 can be written as

$$U_8 = P_8 \cdot V_{8,5} \cdot V_{8,4} \cdot V_{8,3} \cdot P_8, \tag{32}$$

where, according to (14),

$$V_{8,5} = \begin{pmatrix} c_{1,5} & 0 & 0 & 0 & 0 & 0 & 0 & s_{1,5} \\ 0 & c_{3,5} & 0 & 0 & 0 & 0 & s_{3,5} & 0 \\ 0 & 0 & c_{5,5} & 0 & 0 & s_{5,5} & 0 & 0 \\ 0 & 0 & 0 & c_{7,5} & s_{7,5} & 0 & 0 & 0 \\ 0 & 0 & 0 & -s_{7,5} & c_{7,5} & 0 & 0 & 0 \\ 0 & 0 & -s_{5,5} & 0 & 0 & c_{5,5} & 0 & 0 \\ 0 & -s_{3,5} & 0 & 0 & 0 & 0 & c_{3,5} & 0 \\ -s_{1,5} & 0 & 0 & 0 & 0 & 0 & 0 & c_{1,5} \end{pmatrix}. \tag{33}$$

Similarly, to calculate $V_{8,4}$, according to (15), $r = m - q + 1 = 5 - 4 + 1 = 2$ and $V_{N/2^2,q} = V_{16/4,4} = V_{4,4}$, so

$$\mathrm{V}_{8,4} = \begin{pmatrix} V_{4,4} & 0 \\ 0 & V_{4,4} \end{pmatrix}, \tag{34}$$

where $V_{4,3}$ is calculated in (25). For $V_{8,3}$, $r = m - q + 1 = 5 - 3 + 1 = 3$ and $V_{N/2^3,q} = V_{16/8,3} = V_{2,3} = U_2$, so

$$V_{8,3} = \begin{pmatrix} U_2 & 0 & 0 & 0 \\ 0 & U_2 & 0 & 0 \\ 0 & 0 & U_2 & 0 \\ 0 & 0 & 0 & U_2 \end{pmatrix}. \tag{35}$$

Using (12) and (13) the permutation is computed as

$$\Phi_8 = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 0 & 4 & 2 & 6 & 1 & 5 & 3 & 7 \end{pmatrix}. \tag{36}$$

Finally the permutation matrix P_8 is obtained by applying the permutation, shown in (37), to the columns of 8×8 identity matrix as

$$P_4 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}. \tag{37}$$

3.4. 32×32 *DCT.* Equation (6) can be given as

$$H_{32} = \begin{pmatrix} H_{16} & H_{16} \\ H_{16} & -H_{16} \end{pmatrix}. \tag{38}$$

Equation (7) can be given as

$$C_{32}^{II} = \frac{1}{4\sqrt{2}} \cdot B_{32} \cdot T_{32} \cdot B_{32} \cdot W_{32}, \tag{39}$$

where

$$T_{32} = \begin{pmatrix} I_2 & 0 & 0 & 0 & 0 \\ 0 & U_2 & 0 & 0 & 0 \\ 0 & 0 & U_4 & 0 & 0 \\ 0 & 0 & 0 & U_8 & 0 \\ 0 & 0 & 0 & 0 & U_{16} \end{pmatrix}, \tag{40}$$

where U_2, U_4 and U_8 are shown in (9), (24), and (32). To calculate U_{16}, according to (11), $m = \log_2(32) + 1 = 6$ and $3 \le q \le 56$. So U_{16} can be written as

$$U_{16} = P_{16} \cdot V_{16,6} \cdot V_{16,5} \cdot V_{16,4} \cdot V_{16,3} \cdot P_{16}, \tag{41}$$

where, according to (14),

$$V_{16,6} = \begin{pmatrix} c_{1,6} & 0 & 0 & 0 & 0 & 0 & 0 & s_{1,6} \\ 0 & c_{3,6} & 0 & 0 & 0 & 0 & s_{3,6} & 0 \\ 0 & 0 & \ddots & 0 & 0 & \iddots & 0 & 0 \\ 0 & 0 & 0 & c_{15,6} & s_{15,6} & 0 & 0 & 0 \\ 0 & 0 & 0 & -s_{15,6} & c_{15,6} & 0 & 0 & 0 \\ 0 & 0 & \iddots & 0 & 0 & \ddots & 0 & 0 \\ 0 & -s_{3,6} & 0 & 0 & 0 & 0 & c_{3,6} & 0 \\ -s_{1,6} & 0 & 0 & 0 & 0 & 0 & 0 & c_{1,6} \end{pmatrix}. \tag{42}$$

Similarly, to calculate $V_{16,5}$, according to (15), $r = m-q+1 = 6-5+1 = 2$ and $V_{N/2^2,q} = V_{32/4,5} = V_{8,5}$, so

$$V_{16,5} = \begin{pmatrix} V_{8,5} & 0 \\ 0 & V_{8,5} \end{pmatrix}, \tag{43}$$

where $V_{8,3}$ is calculated in (33). For $V_{16,4}$, $r = m-q+1 = 6-4+1 = 3$ and $V_{N/2^3,q} = V_{32/8,4} = V_{4,4}$, so

$$V_{16,4} = \begin{pmatrix} V_{4,4} & 0 & 0 & 0 \\ 0 & V_{4,4} & 0 & 0 \\ 0 & 0 & V_{4,4} & 0 \\ 0 & 0 & 0 & V_{4,4} \end{pmatrix}, \tag{44}$$

where $V_{4,4}$ is calculated in (25). For $V_{16,3}$, $r = m-q+1 = 6-3+1 = 4$ and $V_{N/2^4,q} = V_{32/16,3} = V_{2,3} = U_2$, so

$$V_{16,3} = \begin{pmatrix} U_2 & 0 & 0 & 0 \\ 0 & U_2 & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & U_2 \end{pmatrix}. \tag{45}$$

Using (12) and (13) the permutation is computed as

$$\Phi_{16} = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\ 0 & 8 & 4 & 12 & 2 & 10 & 6 & 14 & 1 & 9 & 5 & 13 & 3 & 11 & 7 & 15 \end{pmatrix} \tag{46}$$

finally the permutation matrix P_{16} is obtained by applying the permutation, shown in (47), to the columns of 16×16 identity matrix as

$$P_{16} = \begin{bmatrix} 1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0 \\ 0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0 \\ 0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0 \\ 0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0 \\ 0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0 \\ 0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0 \\ 0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0 \\ 0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1 \end{bmatrix}. \tag{47}$$

4. Proposed Architecture

The complete hardware architecture of the DCT is shown in Figure 1. Each frame is loaded in the input frame memory. The complete frame is divided into $N \times N$ blocks. The control unit reads the rows of each block from the input memory. At the same time, the control unit passes a "1" to the input multiplexers. Also the address and other control signals are passed to the DCT block. After complete calculation of the DCT, the transformed row is input to the transpose memory, along with its corresponding address. In this way, for the first N clock cycles, the rows from the input memory are input to the DCT and are written on the corresponding addresses in the transpose memory. After the N clock cycles, the control unit passes a "0" for the input multiplexers for the next N clock cycles. So in this way, each column from the transpose memory is input to DCT block, and the outputs of the DCT block are written back in the transpose memory, on the same location from where they are read. When all the columns are read and processed by the DCT, the control unit again starts reading the next $N \times N$ block from the input memory and at the same time, each row from the transpose memory is written to the output transformed memory. In this way all the $N \times N$ blocks are read, processed, and written in the output transformed memory.

When the last row is processed through the DCT, it is written to the transpose memory. At the same time, the first column from the transpose memory is read in order to be processed through DCT block. As the last row was not written, so the last data of the first column is not valid. So "Data0" multiplexer is used for forwarding. In this way, the first output of last transformed row of a $N \times N$ block is forwarded to the input to DCT and also written to the transpose memory.

4.1. DCT Block. DCT block is the main block of the complete architecture. The DCT block takes the input data, the corresponding control signals, and the corresponding addresses. The internal architecture of the DCT block is shown in Figure 2.

DCT block has 4 pipeline stages. The data is passed through the Hadamard block. The Hadamard block is designed with a fully parallel architecture. The Hadamard block takes 32 data at its inputs and passes to the butterfly_32, while the first 16 are input to the butterfly_16, the first 8 are input to the butterfly_8, and the first 4 are input to the butterfly_4 as well. Multiplexers are placed at the inputs of each different size butterfly in order to have correct result from Hadamard block. The select signals for the multiplexers are controlled by the control unit. The Hadamard block has 32 outputs. To have a Walsh transform from a Hadamard one, the bit_reversal and gray_code blocks are placed after the Hadamard block.

In the bit_reversal block, the data at input port number X is moved to output port number Y, where the output port number Y is determined by representation the input port number X and reversing the bits. For example, in case of DCT_16, the Hadamard block will produce 16 valid outputs. So the 16 inputs to the bit_reversal block are shuffled according to bit_reversal rule, for example, $X = 0$ means $X =$ "0000" and the bit_reversal is also $Y =$ "0000". So the first input port will be connected to the first output port. Similarly, for $X =$ "0001", the bit reversal will $Y =$ "1000" which means that the second input port is connected to the eight output port. In this way all the inputs are connected to the outputs according to bit reversal rule. As the architecture supports four different sizes of DCT, it means that the bit

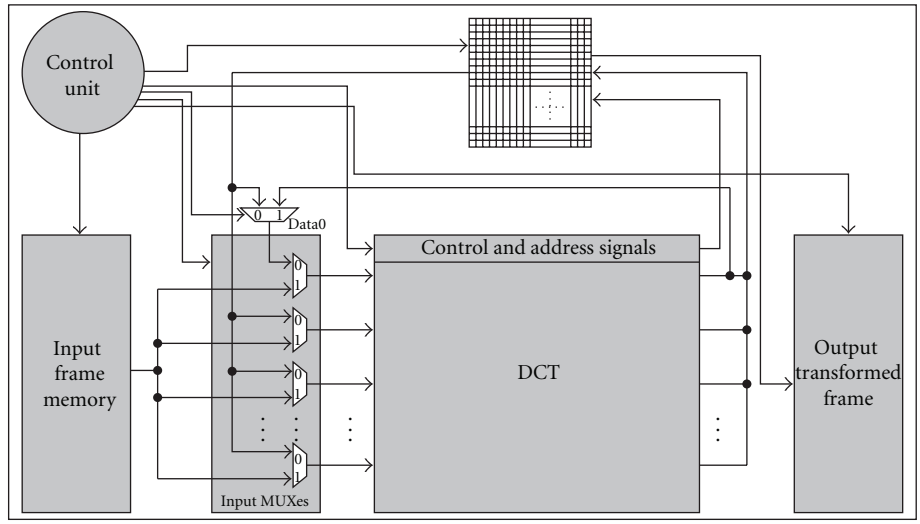

Figure 1: Top level hardware architecture of DCT.

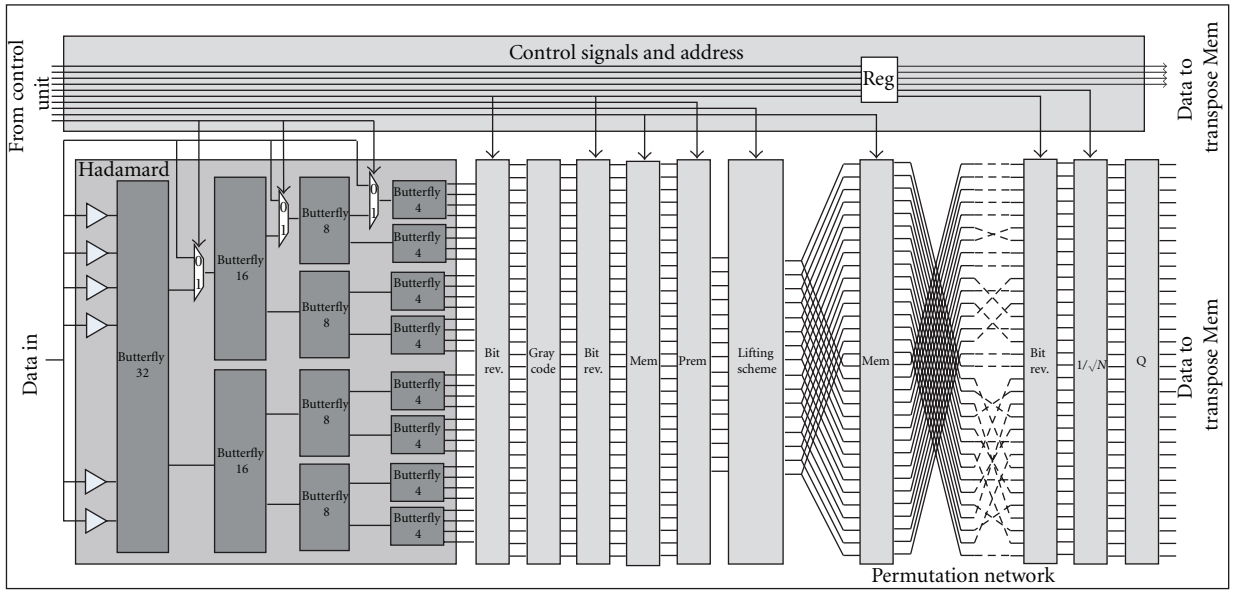

Figure 2: DCT block internal structure.

reversal rule will be different for each DCT size. For example, for DCT_4, X = "01" will be connected to Y = "10", that is, 2nd output port, while in case of DCT_16, it will be connected to the 8th output port. So multiplexers are placed in order to support all the DCT sizes in the bit reversal block.

Gray code block works in the same principle as bit reversal block, but according to gray code law. In gray code block, the output port is determined by applying gray code on the input addresses. For example, for DCT_32 if X = "01101", Y = "01011". So the input port number 13 is connected with the output port number 11. Gray code calculation does not depend on the DCT size. For example for DCT_16, if X = "1101", Y = "1011", which means that input port 13 is connected to the output port number 11, which is same as that for DCT_32.

The architecture of mem.block is shown in Figure 3. The memory block connects the first 16 inputs directly to the output ports, while the last 16 outputs are multiplexed with the latched inputs and the direct inputs. The last 16 outputs are used in case of DCT_32, while the last 16 last inputs are bypassed in case of DCT_4, DCT_8, and DCT_16.

The permutation block is implemented using (27), (36), and (46). The block takes 32 inputs, and it sends 16 of the inputs to the outputs according to the permutation law. The first 16 inputs to the permutation block are passed to the outputs in the first clock cycle, while in the second clock cycle the last 16 inputs of the permutation network are passed to the outputs. In case of $DCT < 32$, the selection line of the multiplexers is always set to "0", while in case of $DCT = 32$, the selection line remains "0" in first clock cycle, while remains "1" in the next clock cycle.

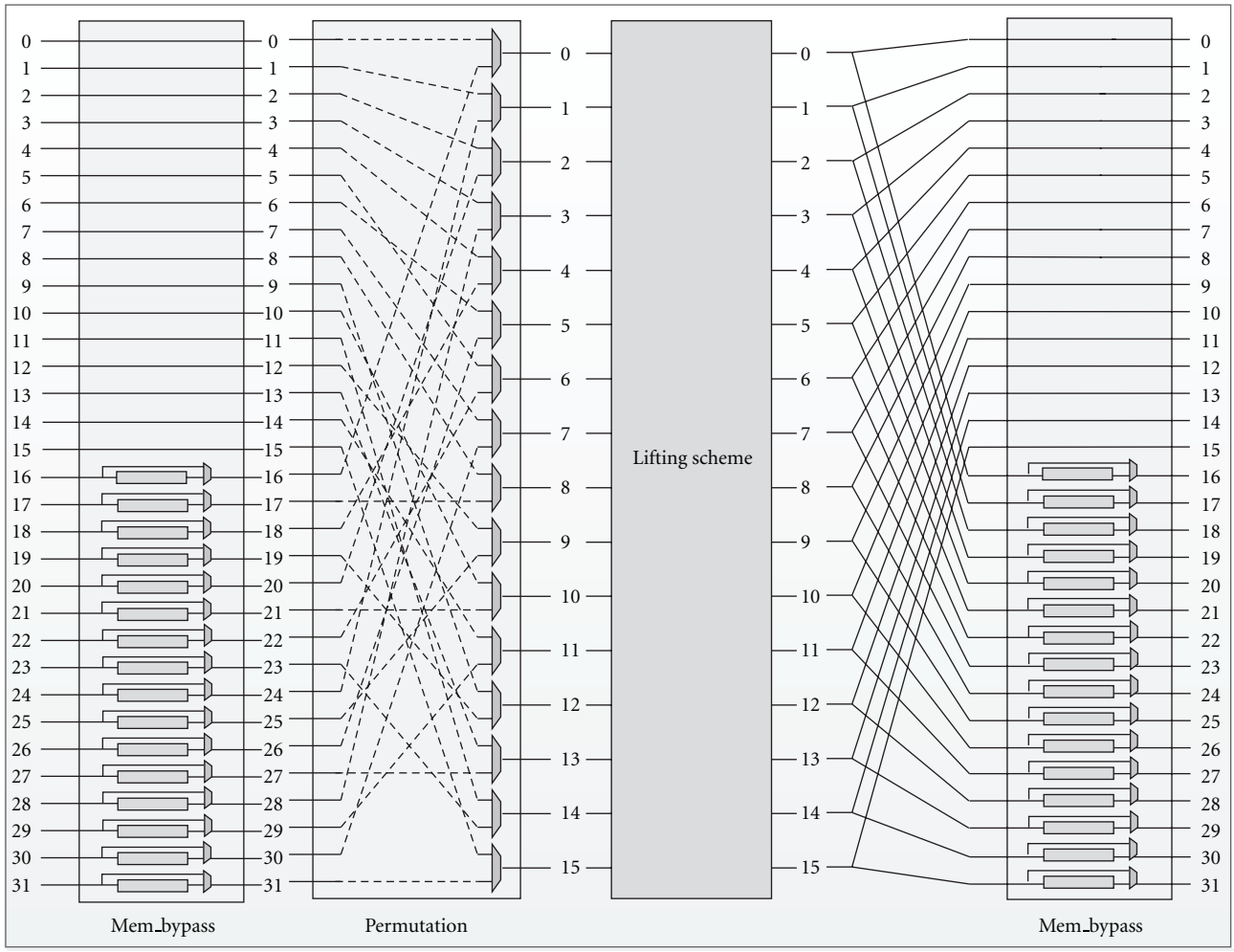

Figure 3: Memory block and permutation block.

The lifting scheme is implemented using (19), (23), (31), and (40). The lifting block is implemented with a folded architecture. Where the fully parallel lifting block is used for DCT sizes of 4, 8, and 16, while DCT_32, the block is reused. Each row of DCT_32 takes 2 clock cycles for completion. During the first clock cycle, the upper 16 inputs are processed by the lifting scheme and are stored in the memory block. In the next clock cycle, the lower 16 inputs are processed by the lifting block and the result along with the previously calculated stored values is forwarded to the next block. The lifting block is shown in Figure 4.

Lifting scheme is designed for 15 Givens rotations. The basic lifting structure, shown in Figure 5, is implemented using (16), where

$$\frac{a}{2^m} \approx \frac{1-\cos\theta}{\sin\theta}, \qquad \frac{b}{2^n} \approx -\sin\theta, \tag{48}$$

as suggested in [9].

The lifting structures takes two values at the inputs. For each Givens rotation, the a, b, m, and n are approximated integer values, in order to have the approximated DCT equal to the actual DCT. As a and b are integers, the multiplications are implemented using adders and shift operations. The result of each lifting structure is quantized to 16-bits resolution to have a reasonable PSNR value. So the final outputs of the lifting block are 16-bit wide. The results of some lifting structures are bypassed using the multiplexers at their outputs. In fact, the select line for the multiplexers will always remain "0" for DCT_4, DCT_8, and DCT_16, while for DCT_32, the select line remains "0" for first clock cycle and "1" the for the next clock cycle.

In case of DCT_32, the Hadamard block produces 32 results in parallel. The outputs of the Hadamard block are fed into the bit reversal block, gray code block, and in the following bit reversal block, and the output of the bit reversal block is fed into the memory block. The memory block passes the upper 16 inputs directly to the permutation block, while the lower 16 inputs are stored in the registers. The permutation block forwards the upper 16 inputs to the lifting scheme, where the select lines for the multiplexers in the lifting scheme are set to "0", and the lifting scheme outputs the results and the results are stored in the following memory

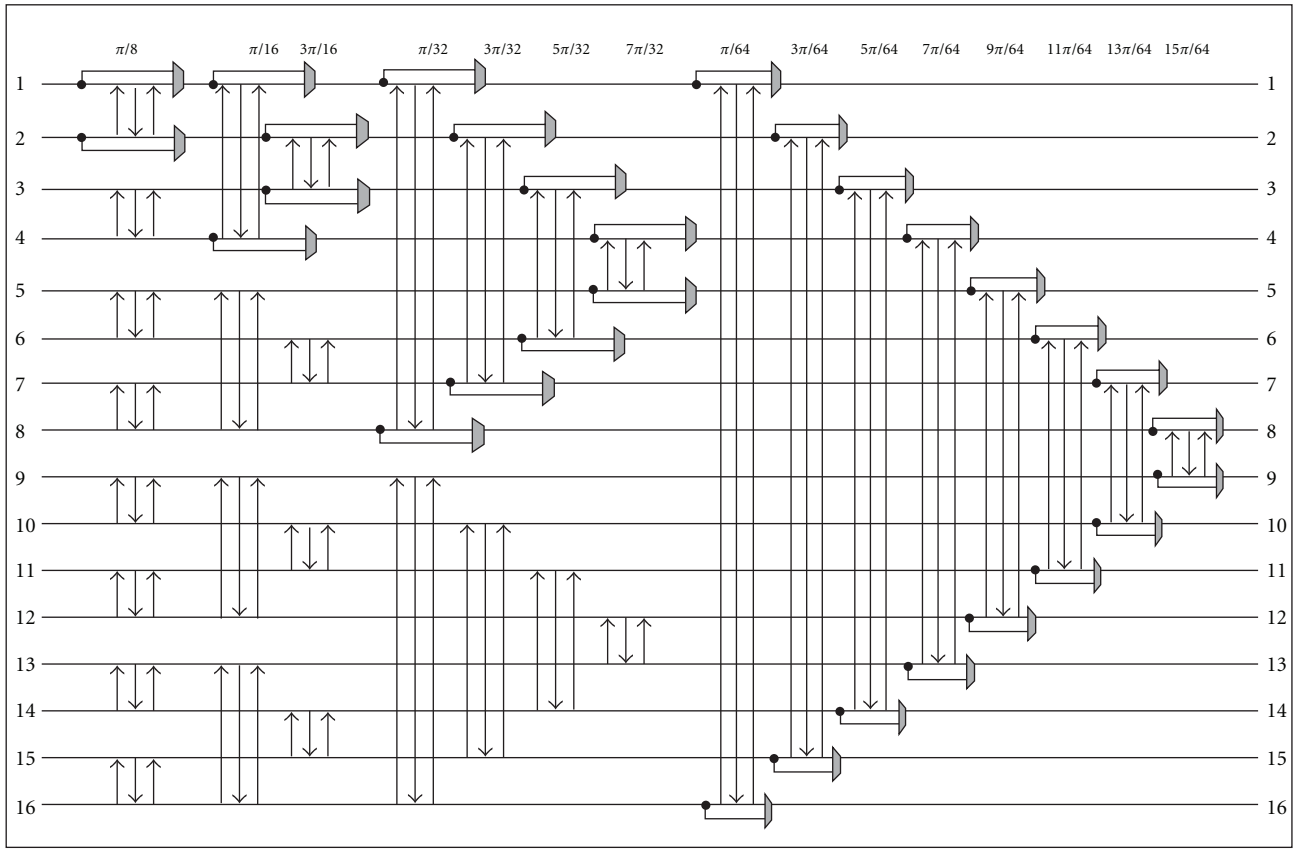

Figure 4: Folded lifting scheme.

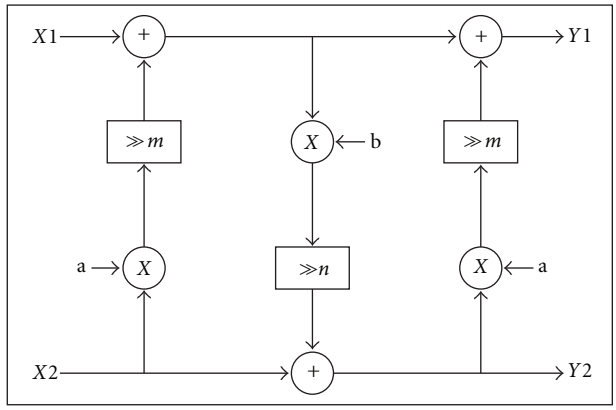

Figure 5: Basic lifting structure.

block. In the next clock cycle the lower 16 values stored in the first memory block are passed to the lifting scheme, through the permutation block. The selection line for the multiplexers in the lifting scheme are set to "1". The 16 results are calculated and are passed to the memory block. At the same time, the memory block forwards the previously stored values along with the new arrived ones.

In case of DCT_4, DCT_8, and DCT_16, the lower 16 values from the first memory blocks are invalid and never used. So the valid upper 16 inputs are fed into the lifting scheme via permutation network. The selection line for the lifting scheme multiplexers is always set to "0" in case of $DCT < 32$.

$DCT < 32$ takes one clock cycle to calculate one row or one column, while $DCT = 32$ takes 2 clock cycles. The outputs of the second memory block are passed to the third bit reversal block, passing through a fully parallel permutation network. The outputs of the bit reversal are then divided by square root of N, where N is the DCT size. The square roots are calculated as

$$\frac{1}{\sqrt{N}} \approx \begin{cases} \dfrac{1}{2}, & N = 2, \\ \dfrac{8+4-1}{32}, & N = 8, \\ \dfrac{1}{4}, & N = 16, \\ \dfrac{8+4-1}{64}, & N = 32. \end{cases} \tag{49}$$

The hardware architecture of the one square root block is shown in Figure 6. The input is divided by $\sqrt{N}$ and the calculated values are fed into the output multiplexer, where the valid result is sent to output depending on the DCT size. Finally, the outputs from the square root block are quantized from 16 bits to 13 bits using the Q block.

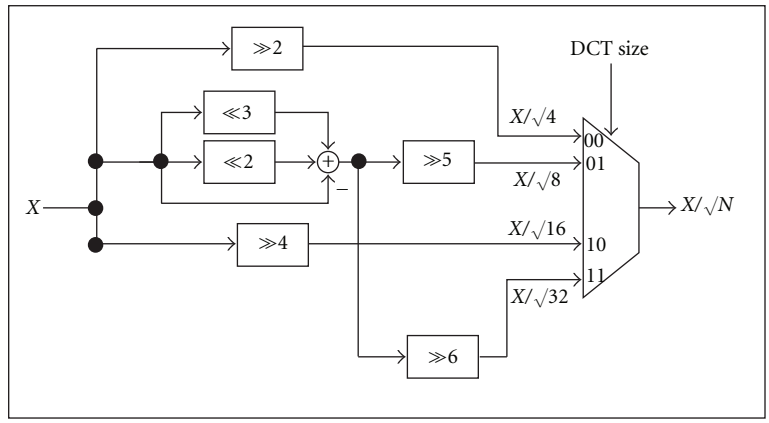

Figure 6: $1/\sqrt{N}$ block.

4.2. Transpose Buffer. Transpose buffer is designed using registers. The buffer is designed to support maximum DCT size, that is, DCT_32. So the buffer is of size $N \times N \times B$, where $N = 32$ and $B = 13$, where B is the width of each data. So a total of 13 kbits memory is utilized to implement transpose buffer. The inputs of the buffer are the clock, reset, transpose signal, the row number, the column number, read enable signal, and the write enable signal. During the direct cycle, all the rows from the input frame memory are transformed through DCT block, and the results are stored on the corresponding rows in the transpose buffer. When all the rows of a input frame memory are transformed and written to the transform buffer, the columns of the transform buffer are read and the columns are transformed via DCT block, and the results are again written back to the transform buffer in transpose way, that is, on each column. When all the columns of the transform buffer are read, transformed, and written back to the buffer, the rows of the input frame memory are read row wise, and at the same time the rows of the transform buffer are written to the output frame memory. In this way, the complete frame is transformed and written to the output memory.

4.3. Input MUXes Block. The DCT transforms the rows of the block and the results are stored in the buffer. So the select signal for the input MUXes block is set to "1". After N clock cycles, where N is the size of DCT, the select signal of the input MUXes is set to "0", so that the columns from the transform buffer is fed into the DCT block. So, input MUXes block switches the inputs for the DCT block for the direct or transformed cycles.

4.4. Data0 Multiplexer. During the direct cycles, the rows from the input memory are transformed and the results are written in the transform buffer. When the last row from the input memory is transformed, first column is read in the next clock cycle from the transform memory. At this point, the last data of the first column is not the valid one, as the last transformed row has not yet been written to the memory. So data0 multiplexer is used for forwarding, where the first data out of the 32 transformed dates is selected from the data0 memory. The select line of the data0 multiplexer is set to "1" for just one clock cycle during transformation of $N \times N$ block, that is, when the first column is read from transform buffer and the last row is transformed via DCT block, otherwise the select signal is always set to "0".

4.5. Control Unit. Control Unit controls the activities of all the blocks in each clock cycle. This unit is responsible for a correct sequence of operations. Control unit is designed using 4 memories, where each memory contains the control signals for each DCT size. There are 4 counters in the unit, where each counter produces the addresses for its corresponding memories. In response to the addresses, the memories output the control signals. The outputs of the memories are multiplexed, where the selection line of the multiplexer decides which input to go out. The hardware architecture of the control unit is shown in Figure 7. MEM_CU_4 is a $8 \times 128 = 1$ kbits size, MEM_CU_8 is a $16 \times 128 = 2$ kbits size, and MEM_CU_16 is a $32 \times 128 = 16$ kbits bits size while MEM_CU_32 is a $128 \times 128 = 16$ kbits size. The memories contains N control signals for direct cycle and N clock cycles for the transpose cycle, where N is the DCT size. But MEM_CU_32 contains $2(N + N)$ control signals, because each row or column takes two clock cycles for completion in case of DCT_32. So the control unit generates 128-bits wide control signals, for all the functioning blocks of the complete DCT in each clock cycle.

5. Results

The computation of N-point DCT by the means of the WHT factorization requires the following.

(1) $(N/2) \cdot \log_2(N)$ 2-inputs/2-output butterfly for the N-point WHT.

(2) $1 + (N/2) \cdot [\log_2(N) - 2]$ 2-input/2-output lifting-based (3-lifting-step) structures for the Givens rotations.

The WHT is implemented with fully parallel architectures using maximum number of resources, while the lifting

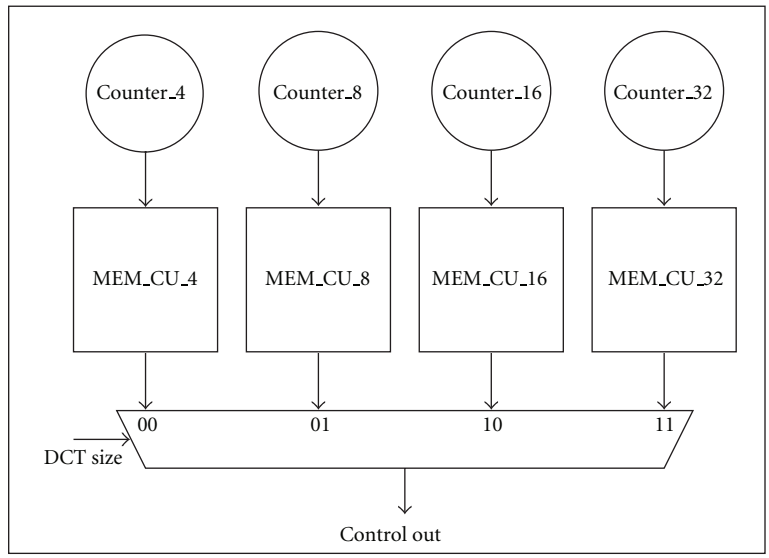

Figure 7: Control unit.

scheme is implemented with a folded architecture. Hence, 80 2-input/2-output butterflies are used to implement WHT for $N = 32$. The number of adders required to implement the 80 2-input/2-output butterflies is 160. The total number of 2-input/2-output lifting structures to implement the 15 Givens rotation is 49, but with folded architecture we have reused the data path and reduced the number of lifting structures to 32.

The factorization of the matrices is applied to H.265 DCT. The lifting coefficients are approximated with the following condition:

$$\left\lfloor \frac{1}{N} \cdot B_N \cdot T_N \cdot B_N \cdot W_N \cdot 2^7 \right\rfloor = Q, \tag{50}$$

where Q is the N-point DCT obtained from MATLAB function dctmtx, scaled with 2^7. Table 1 shows the approximated values, calculated from the conditions in (48), of all the coefficients and the number of bits required to normalize the results.

According to [14], multiplications for $a_{1,3} = 51$ and $b_{1,3} = -98$ can be implemented with a minimum number of additions resorting to the n-dimensional reduced adder graph (RAG-n) technique. The total number of adders required for all the coefficients is shown in Table 1.

The DCT block contains $2 \times N/2 \times \log_2(N) = 160$ adders to implement the Hadamard block. Using Tables 1 and 2, the number of adders use to implement the DCT can be calculated. The first stage of lifting steps ($\pi/8$ rotation) requires $3 \times 8 = 24$ adders to implement the lifting structure, and further $6 \times 8 = 48$ adders are required to implement all the steps involving $a_{1,3}$ and $b_{1,3}$. The first stage of lifting steps ($\pi/16$ rotation) requires $3 \times 4 = 12$ adders to implement the lifting structure, and further $8 \times 4 = 32$ adders are required to implement all the steps involving $a_{1,4}$ and $b_{1,4}$. The first stage of lifting steps ($3\pi/16$ rotation) requires $3 \times 4 = 12$ adders to implement the lifting structure, and further $9 \times 4 = 36$ adders are required to implement all the steps involving $a_{3,4}$ and $b_{3,4}$. The first stage of lifting steps ($\pi/32$ rotation) requires $3 \times 2 = 6$ adders to implement the lifting structure, and further $6 \times 2 = 12$ adders are required to implement all the steps involving $a_{1,5}$ and $b_{1,5}$. The first stage of lifting steps ($3\pi/32$ rotation) requires $3 \times 2 = 6$ adders to implement the lifting structure, and further $7 \times 2 = 14$ adders are required to implement all the steps involving $a_{3,5}$ and $b_{3,5}$.

The first stage of lifting steps ($5\pi/32$ rotation) requires $3 \times 2 = 6$ adders to implement the lifting structure, and further $2 \times 2 = 4$ adders are required to implement all the steps involving $a_{5,5}$ and $b_{5,5}$. The first stage of lifting steps ($7\pi/32$ rotation) requires $3 \times 2 = 6$ adders to implement the lifting structure, and further $9 \times 2 = 18$ adders are required to implement all the steps involving $a_{7,5}$ and $b_{7,5}$. The first stage

Table 1: Number of adders for lifting coefficients.

a	Adders	b	Adders
51	2	−98	2
101	3	−569	2
311	3	−200	3
25	2	−50	2
152	2	−297	3
64	0	−121	2
183	3	−325	3
25	2	−50	2
19	2	−38	2
63	1	−124	1
178	3	−345	4
115	3	−219	3
71	2	−132	1
169	3	−305	3
99	3	−172	3

Table 2: Approximated valued of lifting structures coefficients and the number of bits required for quantization of results.

Givens rotations	m, n	a	b
$\pi/8$	8	51	−98
$\pi/16$	10	101	−569
$3\pi/16$	10	311	−200
$\pi/32$	9	25	−50
$3\pi/32$	10	152	−297
$5\pi/32$	8	64	−121
$7\pi/32$	9	183	−325
$\pi/64$	10	25	−50
$3\pi/64$	8	19	−38
$5\pi/64$	9	63	−124
$7\pi/64$	10	178	−345
$9\pi/64$	9	115	−219
$11\pi/64$	8	71	−132
$13\pi/64$	9	169	−305
$15\pi/64$	8	99	−172

of lifting steps ($\pi/64$ rotation) requires 3 adders to implement the lifting structure, and further 6 adders are required to implement all the steps involving $a_{1,6}$ and $b_{1,6}$. The first stage of lifting steps ($3\pi/64$ rotation) requires 3 adders to implement the lifting structure, and further 6 adders are required to implement all the steps involving $a_{3,6}$ and $b_{3,6}$. The first stage of lifting steps ($5\pi/64$ rotation) requires 3 adders to implement the lifting structure, and further 3 adders are required to implement all the steps involving $a_{5,6}$ and $b_{5,6}$. The first stage of lifting steps ($7\pi/64$ rotation) requires 3 adders to implement the lifting structure, and further 10 adders are required to implement all the steps involving $a_{7,6}$ and $b_{7,6}$. The first stage of lifting steps ($9\pi/64$ rotation) requires 3 adders to implement the lifting structure, and further 9 adders are required to implement all the steps involving $a_{9,6}$ and $b_{9,6}$. The first stage of lifting steps ($11\pi/64$ rotation) requires 3 adders to implement the lifting structure, and further 5 adders are required to implement all the steps involving $a_{11,6}$ and $b_{11,6}$. The first stage of lifting steps ($13\pi/64$ rotation) requires 3 adders to implement the lifting structure, and further 9 adders are required to implement all the steps involving $a_{13,6}$ and $b_{13,6}$. The first stage of lifting steps ($15\pi/64$ rotation) requires 3 adders to implement the lifting structure, and further 9 adders are required to implement all the steps involving $a_{15,6}$ and $b_{15,6}$. The square root block contains 2 adders, and there are 32 square root blocks. Therefore, 64 adders are calculating square roots in parallel. The total number of adders required for Hadamard and lifting scheme is $160 + 72 + 44 + 48 + 18 + 20 + 10 + 24 + 9 + 9 + 6 + 13 + 12 + 15 + 12 + 12 + 64 = 548$.

Figure 8 shows the experiment setup carried out to calculate the PSNR. The original frames are transformed using the proposed DCT. Then the transformed data is quantized to 13 bits. The quantized coefficients are then passed through the inverse quantization block and inverse DCT. The PSNR is then calculated between the original frames and the reconstructed frames. The inverse quantization is taken using (51)

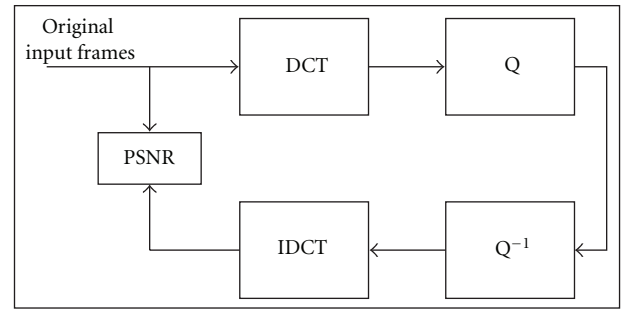

Figure 8: PSNR calculation and experiment setup.

$$x = \left(C_N^{II}\right)^{-1} \cdot X^t. \tag{51}$$

Table 3 shows the PSNR values for different sequences with different DCT sizes. The PSNR is calculated as shown in (52) and (53)

$$\text{PSNR} = 20 \cdot \log\left(\frac{\text{MAX}_\text{I}^2}{\sqrt{\text{MSE}}}\right), \tag{52}$$

where MAX_I is the maximum possible value of the image, and MSE, mean square error, can be defined as

$$\text{MSE} = \frac{1}{m \cdot n} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} [I(i,j) - K(i,j)]^2, \tag{53}$$

where $I(i, j)$ and $K(i, j)$ are the input frame and the reconstructed frame, after inverse quantization and IDCT, respectively. From Table 3, it is quite clear that the DCT is showing great efficiency with respect to PSNR. PSNR of Y frames is very close to 50 dB.

In Tables 4, 5, and 6, the number of multiplications, additions, and shifts, required to calculate different sizes of DCT, are shown. The proposed architecture has no multiplications, where all the multiplications are implemented using shifts and adds. As it can be observed, the number of additions required to compute the 32-point DCT with the proposed architecture is less than the original DCT implementation and the other proposed ones.

The net list is written in VHDL language. Synopsys Design Vision is used for synthesis purpose. The code is synthesized on 90 nm standard cell library at a clock frequency of 150 MHz. Table 7 shows the results of the synthesis.

The time required by the proposed architecture to completely process an $N \times N$ macro block is

$$T_{\text{MB}} = \frac{2}{f_{\text{clk}}} \cdot (\delta \cdot N + 4), \tag{54}$$

where $\delta = 2$ if $N = 32$ and $\delta = 1$ otherwise. Thus, the total time to process one $W \times H$ pixel frame is

$$T = \frac{W \cdot H}{N^2} \cdot T_{\text{MB}} \cdot K = \frac{W \cdot H}{N^2} \cdot K \cdot \frac{2\delta \cdot N + 8}{f_{\text{clk}}}, \tag{55}$$

Table 3: PSNR (dB) of different sequences for different DCT sizes.

Sequences	DCT size											
	$N = 4$			$N = 8$			$N = 16$			$N = 32$		
	Y	U	V	Y	U	V	Y	U	V	Y	U	V
BQ terrace_1920 × 1080_60	47.1	44.4	44.1	48.8	45.9	44.1	48.9	45.4	45.3	48.8	45.7	44.1
BQ square_416 × 240_60	47.4	45.1	45.2	48.1	45.2	45.2	48.1	45.7	44.1	48.2	45.2	44.7
BQ mall_832 × 480_60	46.9	44.1	44.5	47.9	44.9	45.5	48.7	44.6	44.9	48.4	45.7	45.3
Basketball drive_1920 × 1080_50	47.8	44.5	44.1	48.2	44.8	45.7	48.5	45.1	45.7	48.3	45.1	45.3
Basketball drill_832 × 480_50	47.0	45.4	45.6	48.6	45.0	44.8	48.2	45.9	45.0	48.7	45.4	45.5

Table 4: Proposed architecture.

N	M	A	S
4	0	17	5
8	0	74	39
16	0	232	132
32	0	548	249

* N is the DCT size.
* M is the number of multiplications.
* A is the number of additions.
* S is the number of shifts.

Table 5: Comparison for $N = 32$.

	M	A	S
Original	1024	992	0
Proposed	0	548	249

Table 6: Comparison for $N = 16$.

	M	A	S
[5]	0	242	58
Proposed	0	232	132

* N is the DCT size.
* M is the number of multiplications.
* A is the number of additions.
* S is the number of shifts.

Table 7: Synthesis results.

Parameter	Value
Technology	90 nm
Frequency	150 MHz
Area	0.42 mm^2
Power	884.1 μW
Memory	36 kbits

where K accounts for the chroma subsampling, for example, $K = 3$ for 4 : 4 : 4, $K = 2$ for 4 : 2 : 2, and $K = 1.5$ for 4 : 2 : 0. So from (55), taking $H = 1920$, $W = 1080$, and $K = 1.5$ we obtain $T = 2.8$ ms and $T = 20.7$ ms for $N = 32$ and $N = 4$, respectively. As a consequence, in the worst case ($N = 4$) the proposed architecture sustains up to 48 frames per second.

6. Conclusion

In this work, a dynamic N-point DCT for HEVC is proposed. A partially folded architecture is adopted to maintain speed and to save area. The DCT supports 4, 8, 16, and 32 points. The simulation results show that the PSNR is very close to 50 dB, which is reasonably good. Multiplications are removed from the architecture by introducing lifting scheme and approximating the coefficients.

References

[1] T. Wiegand, G. J. Sullivan, G. Bjøntegaard, and A. Luthra, "Overview of the H.264/AVC video coding standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 560–576, 2003.

[2] K. Ugur, K. Andersson, A. Fuldseth et al., "High performance, low complexity video coding and the emerging hevc standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 20, no. 12, pp. 1688–1697, 2010.

[3] S. Song, C. Seo, and K. Kim, "A unified transform unit for H.264," in *Proceedings of the International SoC Design Conference (ISOCC '08)*, pp. 130–133, November 2008.

[4] S. Saponara, M. Martina, M. Casula, L. Fanucci, and G. Masera, "Motion estimation and CABAC VLSI co-processors for real-time high-quality H.264/AVC video coding," *Microprocessors and Microsystems*, vol. 34, no. 7-8, pp. 316–328, 2010.

[5] M. N. Haggag, M. El-Sharkawy, and G. Fahmy, "Efficient fast multiplication-free integer transformation for the 2-D DCT H.265 standard," in *Proceedings of the 17th IEEE International Conference on Image Processing (ICIP '10)*, pp. 3769–3772, September 2010.

[6] Y. M. Huang, J. L. Wu, and C. T. Hsu, "A refined fast 2-D discrete cosine transform algorithm with regular butterfly structure," *IEEE Transactions on Consumer Electronics*, vol. 44, no. 2, pp. 376–383, 1998.

[7] D. Hein and N. Ahmed, "On a real-time Walsh-Hadamard cosine transform image processor," *IEEE Transactions on Electromagnetic Compatibility*, vol. EMC-20, pp. 453–457, 1978.

[8] J. Liang and T. D. Tran, "Fast multiplierless approximations of the DCT with the lifting scheme," *IEEE Transactions on Signal Processing*, vol. 49, no. 12, pp. 3032–3044, 2001.

[9] Y. J. Chen, S. Oraintara, and T. Nguyen, "Video compression using integer DCT," in *International Conference on Image Processing (ICIP'00)*, pp. 844–845, September 2000.

[10] I. Daubechies and W. Sweldens, "Factoring wavelet transforms into lifting steps," *Journal of Fourier Analysis and Applications*, vol. 4, no. 3, pp. 247–268, 1998.

[11] M. Martina, G. Masera, G. Piccinini, and M. Zamboni, "A VLSI architecture for IWT (Integer Wavelet Transform)," in *Proceedings of the 43rd IEEE Midwest Symposium on Circuits and Systems*, pp. 1174–1177, August 2000.

[12] M. Martina, G. Masera, G. Piccinini, and M. Zamboni, "Novel JPEG 2000 compliant DWT and IWT VLSI implementations," *Journal of VLSI Signal Processing Systems for Signal, Image, and Video Technology*, vol. 35, no. 2, pp. 137–153, 2003.

[13] M. Martina and G. Masera, "Folded multiplierless lifting-based wavelet pipeline," *IET Electronics Letters*, vol. 43, no. 5, pp. 27–28, 2007.

[14] A. G. Dempster and M. D. Macleod, "Use of minimum-adder multiplier blocks in FIR digital filters," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 42, no. 9, pp. 569–577, 1995.

[15] M. Martina and G. Masera, "Low-complexity, efficient 9/7 wavelet filters VLSI implementation," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 53, no. 11, pp. 1289–1293, 2006.

[16] M. Martina and G. Masera, "Multiplierless, folded 9/7–5/3 wavelet VLSI architecture," *IEEE Transactions on Circuits and Systems II*, vol. 54, no. 9, pp. 770–774, 2007.

[17] R. Joshi, Y. A. Reznik, and M. Karczewicz, "Efficient large size transforms for high-performance video coding," in *33rd Proceedings of the Applications of Digital Image Processing*, Proceedings of the SPIE, San Diego, Calif, USA, August 2010.

[18] M. Martina, G. Masera, and G. Piccinini, "Scalable low-complexity B-spline discrete wavelet transform architecture," *IET Circuits, Devices and Systems*, vol. 4, no. 2, pp. 159–167, 2010.

[19] V. Britanak, P. C. Yip, and K. R. Rao, *Discrete Cosine and Sine Transforms: General Properties, Fast Algorithms and Integer Approximations*, Elsevier, 2007.

[20] H. W. Jones, D. N. Hein, and S. C. Knauer, "The Ratio CO/CO_2 of oxidation on a burning carbon surface," pp. 87–98, November 1978.

[21] N. Ahmed and K. R. Rao, *Orthogonal Transforms for Digital Signal Processing*, Springer, 1975.

[22] J. W. Manz, "A sequency-ordered fast Walsh transform," *IEEE Transactions on Audio Electroacoustics*, vol. AU-20, pp. 204–205, 1972.

[23] A. T. S. Claire and C. Sabido-David, "Unified matrix treatment of the fast Walsh-Hadamad transfor," *IEEE Transactions on Computers*, vol. 25, no. 11, pp. 1142–1146, 1976.

Hindawi Publishing Corporation
VLSI Design
Volume 2012, Article ID 602737, 17 pages
doi:10.1155/2012/602737

Research Article

Hardware Design Considerations for Edge-Accelerated Stereo Correspondence Algorithms

Christos Ttofis and Theocharis Theocharides

Department of Electrical and Computer Engineering, University of Cyprus, P.O. Box 20537, 1678 Nicosia, Cyprus

Correspondence should be addressed to Christos Ttofis, ttofis.christos@ucy.ac.cy

Received 25 February 2012; Accepted 23 March 2012

Academic Editor: Muhammad Shafique

Stereo correspondence is a popular algorithm for the extraction of depth information from a pair of rectified 2D images. Hence, it has been used in many computer vision applications that require knowledge about depth. However, stereo correspondence is a computationally intensive algorithm and requires high-end hardware resources in order to achieve real-time processing speed in embedded computer vision systems. This paper presents an overview of the use of edge information as a means to accelerate hardware implementations of stereo correspondence algorithms. The presented approach restricts the stereo correspondence algorithm only to the edges of the input images rather than to all image points, thus resulting in a considerable reduction of the search space. The paper highlights the benefits of the edge-directed approach by applying it to two stereo correspondence algorithms: an SAD-based fixed-support algorithm and a more complex adaptive support weight algorithm. Furthermore, we present design considerations about the implementation of these algorithms on reconfigurable hardware and also discuss issues related to the memory structures needed, the amount of parallelism that can be exploited, the organization of the processing blocks, and so forth. The two architectures (fixed-support based versus adaptive-support weight based) are compared in terms of processing speed, disparity map accuracy, and hardware overheads, when both are implemented on a Virtex-5 FPGA platform.

1. Introduction

Depth extraction from stereoscopic images is a vital step in several emerging embedded applications, such as robot navigation, obstacle detection for autonomous vehicles, and space and avionics [1]. Depth information is extracted by solving the challenging problem of stereo correspondence, which aims at finding corresponding points (or conjugate pairs) between the input stereo images (usually referred to as reference image I_r and target image I_t). Given that the input images are rectified, the correspondence of a pixel at coordinate (x, y) can only be found at the same vertical coordinate y, and within a maximum horizontal bound, called *disparity range* D (d_m–d_M) [2]. The disparity is computed as the absolute difference between the coordinates of the corresponding pixels in I_r and I_t. The disparities of all corresponding pixels form a disparity map, which once estimated, can be used to extract the depth of the scene by using triangulation [1].

In general, stereo correspondence algorithms mostly follow four steps: (1) matching cost computation, (2) cost (support) aggregation, (3) disparity computation/optimization, and (4) disparity map refinement [2]. Moreover, they are classified into two broad categories: *global* and *local* [2]. Global algorithms can produce very accurate results but are slower and computationally more demanding compared to local algorithms, due to their iterative nature and high memory needs [3]. On the other hand, local algorithms are faster and less computationally expensive, hence suitable for the majority of embedded stereo vision applications.

Local algorithms determine the disparity of pixel p in I_r by finding the pixel of I_t that yields the lowest score of a similarity measure (see [3] for a review) computed on a support (correlation) window centered on p and on each of the d_M–d_m candidates defined by the disparity range. Early local algorithms have followed a simple approach that uses a fixed (typically square) support window during the cost aggregation step. This approach, however, is prone to errors

as it blindly aggregates pixels belonging to different disparities, resulting in incorrect disparity estimation at depth discontinuities and regions with low texture [3]. Recent local algorithms improve this basic approach by using multiple-window and adaptive-support weight (ADSW) methods [4]. The latter methods represent state-of-the-art in local stereo correspondence algorithms, as they can generate disparity maps that approach the accuracy of global algorithms [3]. These methods operate by assigning different weights to the pixels in a support window based on the spatial and color distances to the center pixel [4], or on information extracted from image segmentation [5, 6]. In this way, they aggregate only those neighboring pixels that are at the same disparity. Despite their improvements in accuracy, ADSW algorithms are generally slower than other local algorithms. A survey for different approaches can be found in [2, 3].

The real-time and low-power constraints of most embedded stereo vision applications complicate the realization of stereo correspondence algorithms. Software implementations usually struggle to meet these constraints, and as a result, some form of hardware acceleration or even a complete custom hardware implementation is preferred [3]. Hardware acceleration of stereo correspondence algorithms has been done extensively using Digital Signal Processors (DSPs) and Graphics Processing Units (GPUs) [3]. These systems, however, involve architectures that are generally not suitable for embedded applications. DSPs do not provide enough computational power to achieve parallel processing, while GPUs consume excessive power. The strong embedded requirements of such applications imply the use of dedicated hardware architectures such as ASICs and FPGAs, which can provide the necessary computational power and the energy efficiency.

The majority of dedicated hardware architectures implement fixed-support algorithms, providing adequate frame rates. However, numerous embedded vision applications require not only real-time processing speed, but also reliable depth computation. As such, there have been a few attempts recently to implement more complex algorithms in hardware such as ADSW algorithms, but these algorithms are computationally more expensive and thus the frame rate suffers, especially when high resolution images are used. It is imperative that the key to providing a successful realization of an embedded stereo vision system is the careful design of the core architecture so as to provide a good tradeoff between the processing speed and the quality of the resulting disparity map.

This paper presents an approach to the acceleration of hardware implementations of stereo correspondence algorithms for embedded stereo vision systems, by using edge information. The presented approach lies on the extraction of feature points (edges) that are used to restrict the stereo correspondence algorithm only to specific features of the input images (rather than to all image points), thus resulting in a considerable reduction of the search space. The paper presents and analyzes design issues and considerations associated with the edge-directed approach when it is applied to different hardware implementations of stereo correspondence algorithms, including both fixed-support and even more complex ADSW algorithms.

The paper extends our initial work on a hardware implementation of an edge-directed fixed-support stereo correspondence algorithm presented in [7]. In particular, the extended work generalizes the idea of using edge information as part of the stereo correspondence process and presents design principles and considerations that arise when integrating edge information on the architecture of an ADSW algorithm as well. The paper also explores the effects of different edge detection algorithms on the ADSW-based architecture, showing the impact of each algorithm on the quality of the disparity maps produced by that architecture. Furthermore, the paper compares the two architectures (fixed-support based versus ADSW-based) in terms of processing speed, disparity map accuracy and hardware overheads, when both are implemented on a Virtex-5 FPGA platform. The experimental results indicate that the presented edge-directed approach for accelerating hardware implementations of stereo correspondence algorithms exhibits high potential for applications with hard real-time constraints, as both architectures achieve remarkable speedups relative to existing hardware architectures that implement equally complex algorithms. The architecture based on the fixed-support algorithm can meet the real-time requirements of such applications, even for high-resolution images. Moreover, the combination of the edge-directed approach with the ADSW algorithm enables the realization of accurate disparity computation systems that can also satisfy the real-time requirements of many embedded vision applications such as pedestrian and obstacle detection.

The rest of this paper is organized as follows. Section 2 discusses related work and Section 3 presents the edge-based stereo correspondence process. Section 4 presents the proposed edge-directed disparity map computation architectures. Section 5 presents the experimental framework and results, and Section 6 concludes the paper, discussing future work.

2. Related Work

There exists a large number of implementations in the literature that solve the stereo correspondence algorithm. Several existing works feature algorithms running on general purpose processors, as well as clusters and multiprocessor systems [8–10]. However, most of these implementations require high-end computational equipment in order to compute the disparity map in real-time, especially as the image resolution increases. As a result, the real-time processing of those implementations is limited only to small-sized images (smaller than VGA). Alternatively, specialized hardware, such as Intel MMX [11], Graphics Processing Units (GPUs) [12, 13], and Digital Signal Processors [14–16], has been used to accelerate the disparity map computation. [11] attempts to improve the accuracy of the simple SAD correlation technique by using a multiple window approach that decreases errors at object boundaries. This however requires a lot of computational resources. [12, 13] present GPU-based implementations that achieve high frame rates

that address the computational needs of stereo vision algorithms; however, they are not currently suitable for embedded and mobile applications due to their power demands. Approaches implemented on high-end fixed point DSPs [14–16] consume lower power; however, they do not provide the parallelism of FPGAs and ASICs, thus are less suitable for stereo vision applications with hard real-time requirements. There exist also attempts to implement stereo vision algorithms on the Cell processor [17, 18], but are subject to restrictions imposed by the Cell platform, mainly due to the limited memory of the Synergistic Processing Elements (SPEs).

The last decade features an emergence in dedicated hardware architectures, as a means to address the aforementioned constraints found in software and specialized hardware approaches. Dedicated hardware architectures, implemented on ASICs and FPGAs, can exploit the parallelism inherent in the stereo correspondence process and optimize memory access patterns to provide fast computation. Most of dedicated hardware implementations have been implemented on FPGA platforms, taking advantage of the reconfiguration offered by these platforms, in order to exploit the intrinsic parallelism of the stereo correspondence algorithm. A stereo depth measurement system on an FPGA is introduced in [19]. It generates disparities on 512×480 images at 30 fps, implementing a window-based correlation search technique. Another system presented in [20] yields 20 fps on 640×480 image sizes. However, the memory access pattern utilized does not provide scalability and performance optimization, limiting the overall performance of the system. In [21], a survey of some FPGA implementations, [22–24], is presented, and the survey highlights the common use of the SAD similarity measure, a relatively simple technique suitable for hardware implementations. Furthermore, [22–24] make extensive use of parallelism and pipelining in order to achieve real-time performance. [24] also raises the impact of the image size on the performance of the disparity map computation; it claims 5063 fps using very small (64×64) images. The frame rate, however, decreases exponentially as the image size increases.

Other dedicated hardware architectures suitable for real-time disparity map computation are given in [25–31]. These works attempt to implement high-performance stereo correspondence algorithms that fail to achieve real-time performance in software. The FPGA-based architectures presented in [25, 26] implement local fixed-support algorithms using the SAD similarity measure and compute intermediate-sized disparity maps at a rate of 768 and 600 fps, respectively. The system in [27] is based on a phase-based computational model, as an alternative to feature correspondence and correlation techniques. That work exploits the parallel computing resources of FPGA devices to produce dense disparity maps for high-resolution images at 52 fps. The supported disparity range, however, is not practical for embedded stereo vision systems, which traditionally utilize much larger disparity ranges, especially when high-resolution images are used. Another implementation of a computationally complex disparity algorithm that is based on locally weighted phase correlation is presented in [28] and utilizes 4 FPGAs to produce dense disparity maps of size 256×360 at the rate of 30 fps. A more recent FPGA implementation of a real-time stereo vision system is presented in [29]. That system generates dense disparity maps based on the Census transform. Lastly, the hardware implementation presented in [31] performs a modified version of the Census transform in both the intensity and the gradient images, in combination with the SAD correlation metric (SAD-IGMCT algorithm), achieving 60 fps on 750×400 images.

The majority of the aforementioned implementations adopt local fixed-support algorithms to achieve real-time performance, as these algorithms can be greatly benefited by the use of parallel and straightforward structures, which are key factors available in dedicated hardware implementations. ADSW algorithms, which traditionally achieve better disparity map accuracy, have been rarely implemented on dedicated hardware architectures. As mentioned above, these algorithms are computationally more expensive compared to fixed-support algorithms and hence require complex and hardware-unfriendly operations. To the best of our knowledge, only the work in [30] implements a local ADSW stereo correspondence algorithm. That work proposes an accurate, hardware-friendly disparity estimation algorithm called mini-census ADSW, and its corresponding real-time VLSI architecture that achieves 42 fps on 352×288 image sizes. There has been a good effort in [30] to reduce the computational complexity of the ADSW and achieve real-time performance. However, that work achieves real-time performance for relatively small-sized images.

This work investigates the integration of edge-directed information into hardware architectures of stereo correspondence algorithms, as a means to restrict the stereo correspondence process only to the specific features (edges) of the input images (rather than to all image points), thus reducing the search space considerably. The proposed approach targets real-time embedded vision applications and can benefit both fixed-support and ADSW algorithms. When applied to fixed-support algorithms, it leads to very efficient implementations that can effectively address the hard real-time constraints of such applications even when using high resolution images and large disparity ranges. When applied to ADSW algorithms, the proposed approach provides a good tradeoff between accuracy of the disparity maps and processing speed; the reduction of the search space leads to real-time implementations for image sizes which are larger than the ones used in [30]. A comparison between the proposed work and other existing disparity map computation systems is given in Table 7.

3. Edge-Based Stereo Correspondence Process Overview

3.1. Summary of the Algorithm. In this section we present the overall flow of the edge-based stereo correspondence process. The description is generic and applies to both fixed-support and adaptive-support weight (ADSW) algorithms. In general, there are three major tasks associated with an edge-based stereo correspondence algorithm: edge detection, stereo correspondence, and interpolation. Figure 1 illustrates

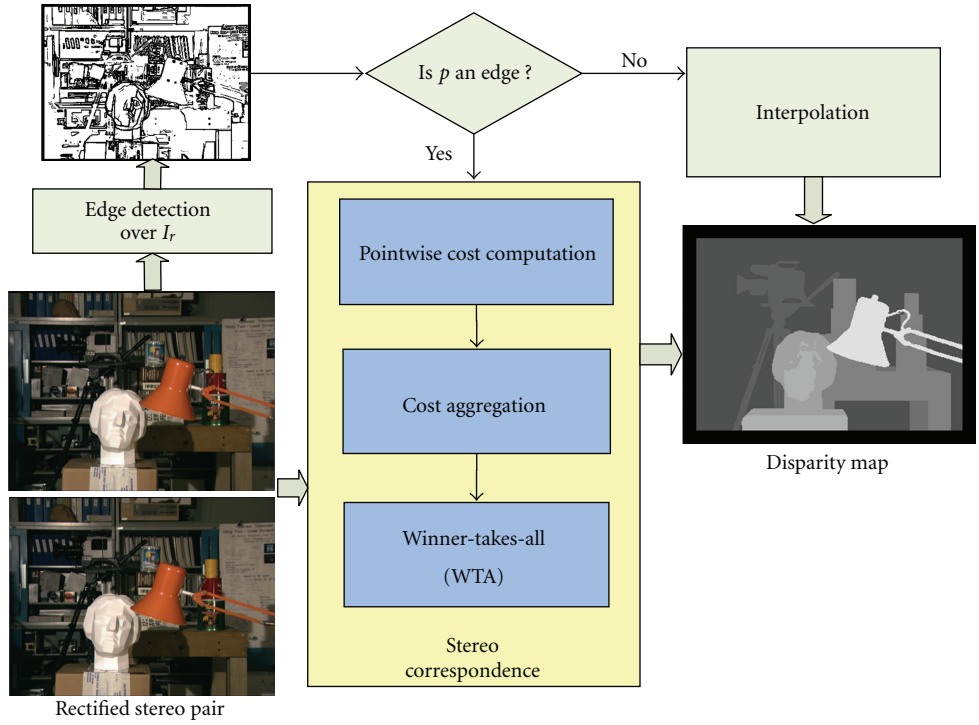

Figure 1: Steps of the edge-based stereo correspondence process.

the algorithm's steps towards computing a disparity value for each pixel p in I_r. The algorithm starts by determining whether a pixel p corresponds to an edge or not. For each pixel p corresponding to an edge, the algorithm extracts an $m \times m$ correlation (or support) window W_r centered on p in I_r, and an $m \times m$ support window W_t centered on q in I_t (the coordinate of q is $(x + d, y)$, where d lies in the range d_m to d_M). The algorithm then computes a pointwise score for any pixel $p_i \in W_r$ corresponding to $q_i \in W_t$ and aggregates the scores spatially over the support windows.

Fixed-support and ADSW stereo correspondence algorithms differ in the way they perform the pointwise score computation and aggregation steps. The fixed-support algorithm computes a pointwise score for any pixel $p_i \in W_r$ corresponding to $q_i \in W_t$ as the absolute different (AD) of p_i and q_i. After the computation of the AD values, the final aggregated cost is computed by summing all pointwise scores as

$$C_{x+d,y} = \sum_{p_i \in W_r, q_i \in W_t} \mathrm{AD}(p_i, q_i), \quad d_m \le d \le d_M. \tag{1}$$

Equation (1) holds only for the case of a fixed-support algorithm. In the case of an ADSW algorithm, each pointwise score is computed by multiplying the absolute difference of p_i and q_i by a weight coefficient $w'_r(p_i, p_c)$ and a weight coefficient $w'_t(q_i, q_c)$. These weight coefficients are assigned to each pixel in a support window, based on the pixel's spatial distance, as well as on its distance in the CIELAB color space with regard to the central pixel (p_c or q_c) in the window. The final aggregated cost is computed by summing up all the weighted pointwise scores, and normalizing by the weights sum as in (2), where, d_p and d_c are the Euclidean distance between two coordinate pairs and two triplets in the CIELAB color space, respectively, and γ_p and γ_c are two parameters of the algorithm

$$C_{x+d,y} = \frac{\sum_{p_i \in W_r, q_i \in W_t} w'_r(p_c, p_i) \cdot w'_t(q_c, q_i) \cdot \mathrm{AD}(p_i, q_i)}{\sum_{p_i \in W_r, q_i \in W_t} w'_r(p_c, p_i) \cdot w'_t(q_c, q_i)}, \quad d_m \le d \le d_M,$$
$$\text{where } w'_{r,t} = \exp\left(-\frac{d_p(p_i, p_c)}{\gamma_p} - \frac{d_c(I_{r,t}(p_i), I_{r,t}(p_c))}{\gamma_c}\right). \tag{2}$$

Both the fixed-support and ADSW algorithms compute an aggregated cost for all disparity levels in the range $[d_m\ d_M]$, and the best disparity for the pixel p is found by locating the disparity with the minimum aggregated cost through a winner-takes-all (WTA) approach. In the case where pixel p does not correspond to an edge, the disparity is obtained by an interpolation method.

3.2. Discussion. The edge-based stereo correspondence algorithm described above applies an edge detection process over the input image I_r prior to performing the correlation step. Edge detection returns locations in the image that indicate the presence of an edge [1]. These locations, described by the *edge points*, determine the outline (i.e., perimeter) of an object and distinguish it from the background and other objects [1, 32]. The correlation window W_r in I_r is moved only to the edges (and not to each possible pixel) along the working scanline, resulting in a considerable reduction in

the search space. As a result, the correlation step computes only a disparity value for the pixels that correspond to an edge, while the disparity values of the remaining pixels are computed using interpolation, which however is considered computationally less expensive compared to stereo correspondence.

Another consideration of the edge-based stereo correspondence process is whether it uses the edges only as directive points or as matching primitives as well (instead of using pixel intensities). Since edges are encoded using only one bit per pixel, rather than 8 bits (as used in grayscale images), matching edges instead of pixel intensities reduce the computational space and complexity of the search and match operations, as well as the data path requirements. However, this can be applied only to the fixed-support algorithm. Computing pointwise scores using binary data returns only zeros or ones, and this would invalidate the weights in the ADSW algorithm.

4. Hardware Realization of Edge-Accelerated Disparity Map Computation Architectures

This section presents the hardware architectures of two different edge-based stereo correspondence algorithms: an SAD-based fixed-support algorithm and an ADSW-based algorithm. We present design principles and considerations about the implementation of these algorithms on hardware, and discuss issues related to the memory structures needed, the organization of the processing blocks, the parallelism exploited, and so forth.

4.1. Disparity Map Computation Hardware Architecture Based on the Fixed-Support Algorithm

4.1.1. Design Issues and Requirements. This architecture implements an SAD-based, fixed-support algorithm and uses edges both as directive points and as matching primitives. This requires that the architecture should be able to perform edge detection on both input images. Moreover, the use of binary data during the matching process reduces the computation of the SAD values (cost computation step) to a hamming distance operation that can be directly implemented in hardware using only addition and 1-bit subtraction operations. The simplicity of the logic required during the cost computation step enables the parallel design of the architecture, so that to target a large number of search and match operations performed in a single cycle. However, the amount of parallelism that can be extracted depends on the ability of the edge detector to generate edge points (as the matching process is performed using edges). In this work we integrate the Sobel detector mainly due to its simple hardware structure, which can be easily parallelized to provide more than one edge points per single cycle. While the use of edge information reduces the search space and simplifies the logic required, the hardware implementation of the algorithm is not straightforward, but the overall design poses significant challenges in terms of the memory structures used in order to account for the irregular rates of data due to the edge only computation. The architecture requires a clever arrangement of the on-chip memory in order to be able to process multiple support windows centered at the edge points, while skipping the nonedge points found between successive edges (every clock cycle).

4.1.2. General Structure. The block diagram of the architecture is shown in Figure 2 and consists of two major hardware units: the Edge Detection Unit (EDU) and the Disparity Computation Unit (DCU). It also consists of a Memory Controller and Control Unit (MCCU) that optimizes memory access based on the algorithm requirements and coordinates data transfers and handshakes between the EDU and the DCU. The EDU converts the rectified input image pair (grayscale images) into a pair of binary images (black and white) that characterize only the edges of the initial images. The black and white images are then fed into the DCU, which performs correlation with the objective to compute the disparity map of the input image pair. While the disparity maps generated by the DCU are sparse (disparity estimates are provided only for points corresponding to edges), the system can generate dense disparity maps as well, by integrating interpolation methods as part of the MCCU. In this work, we use the simple and fast nearest neighbor interpolation method, as our emphasis has been on performance in embedded scenarios.

The EDU and the DCU, which communicate through the use of internal memory (FIFO queues), are pipelined, and thus operate concurrently. They are also provided with scanline buffers, which temporarily store the pixels needed to perform convolution (in the case of the EDU), or correlation (in the case of the DCU). This reduces the clock cycles required to load image data from the input port, by exploiting the fact that working windows moved over the image use overlapping pixels. The scanline buffers are organized into FIFO structures and their size depends on the size of the working window (3×3 for the EDU, $m \times m$ for the DCU) and the width of the image, N. The delay to fill the scanline buffers is proportional to the I/O bandwidth.

4.1.3. EDU Architecture Overview. The Edge Detection Unit (EDU) integrated to the system implements the Sobel edge detector, which performs a 2D spatial gradient measurement on the input grayscale images using a pair of 3×3 convolution masks. The masks hold data values between -2 and 2; thus the overall convolution can be implemented in hardware using shifters instead of multipliers. We integrated this detector mainly due to its simple hardware structure, which can be easily parallelized to provide more than one edge points per single cycle. This is important, as the ability of the DCU to perform parallel computations depends on the ability of the ECU to provide multiple edge points per cycle. The architecture of the EDU is shown in Figure 2(a). The EDU employs hardware features, such as parallelism and pipelining, in an effort to parallelize the repetitive calculations involved in the Sobel operation, and uses optimized memory structures in order to reduce the memory reading redundancy. The detector architecture consists of an I/O

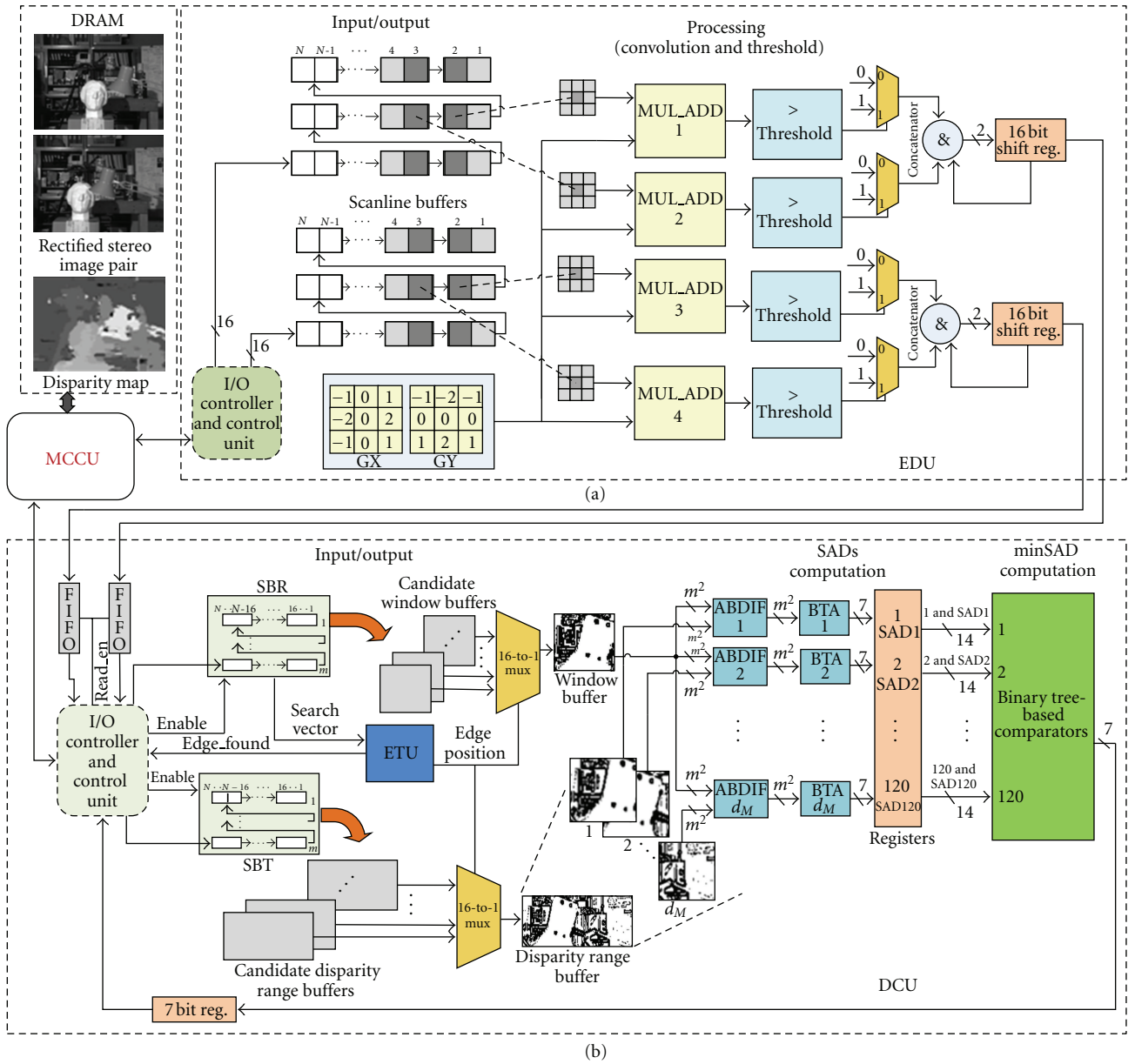

Figure 2: Block diagram of the edge-accelerated hardware architecture that implements the fixed-support algorithm. (a) Edge Detection Unit, (b) Disparity Computation Unit.

controller that reads/writes pixel data from/to the I/O port, on-chip memory for storing pixel data (scanline buffers) and the convolution mask values, a series of convolution units (MUL_ADD units), as well as comparators. The architecture is pipelined into 2 stages: *INPUT/OUTPUT* and *PROCESSING*. In the *INPUT/OUTPUT* stage the I/O controller fetches 4 pixels from the I/O port (2 pixels per image), in a row-wise fashion, and forwards them to the input ports of the scanline buffers (16 bits to each buffer), which have a FIFO structure. During the *PROCESSING* stage, the scanline buffers produce four successive 3×3 windows per cycle, which are convoluted with the 3×3 Sobel masks by the MUL_ADD units. This pipeline stage returns 1-bit pixel intensity values, which are concatenated into two 16-bit registers. The values from the 16-bit registers are forwarded to the output port (FIFO queues of the DCU) once every 8 clock cycles.

4.1.4. DCU Architecture Overview. The DCU has been designed in such a way as to calculate sparse disparity maps covering disparity ranges up to 120 pixels and correlation window sizes from 3×3 to 11×11. The architecture of the DCU is shown in Figure 2(b) and involves three major steps: *INPUT/OUTPUT*, *SADs_COMPUTATION*, and MIN*SAD_COMPUTATION*. The architecture consists of an I/O controller that reads/writes data from/to the I/O port, on-chip memory (FIFO queues, scanline buffers and window buffers) for temporarily storing the edge data, and a unit that searches and finds the position of the edge to be processed

(edge tracking unit (ETU)). It also consists of a collection of adders and subtractors to compute the SAD values (1) for all disparity levels, a unit to compute the minimum SAD value, multiplexers, and intermediate pipeline registers. Apart from reading/writing data from/to the I/O port, the I/O controller also acts as a control unit, coordinating memory accesses from the on-chip memory, as well as data transfers and handshakes between the components of the DCU.

The DCU unit was designed with emphasis on parallelism, targeting a large number of search and match operations performed in a single clock cycle. This is facilitated by the simplicity of the adders and subtractors used (due to the use of binary data), and by the organization of the adders and comparators in tree structures. Due to its pipelined and parallel structure, the DCU presents good scalability in terms of correlation window size and disparity range. Particularly, it can compute the SAD values for all possible positions of the shifting window with a maximum size of 11×11 in two clock cycles, and the minimum SAD value for a maximum of 120 disparity levels in three cycles. However, once the pipeline fills up and the FIFO queues are not empty, the DCU can provide a disparity value at the output every clock cycle.

4.1.5. DCU Memory Architecture and Edge Tracking Process. To keep a constant flow of data in the pipeline, the DCU must be able to locate one edge in the reference image every clock cycle, while discarding the non-edge points found between successive edges. At the same time, the DCU must have parallel access to the $m \times m$ window surrounding the edge found in the reference image, as well as to the corresponding d_M windows from the target image. For these reasons, each scanline buffer used in the DCU consists of a series of 16-bit registers and can store m scanlines from an input image. We avoid using 1-bit registers in order to facilitate more parallelism and to make the process of discarding the non-edge points fast. The 16-bit registers are organized into FIFO structures and allow parallel access to their elements. Specifically, the scanline buffers for the reference image (SBR) output 16 successive $m \times m$ windows (stored in the candidate window buffers), while the scanline buffers for the target image (SBT) output 16 successive $m \times (m + d_M)$ windows (stored in the candidate disparity range buffers). The SBR also outputs a 16-bit vector (search vector) from positions $1 + w$ to $16 + w$ of the $(w + 1)$th scanline, where $w = (m-1)/2$. The search vector is being searched for potential edge points by the edge tracking unit (ETU), which is the connection point between the *INPUT/OUTPUT* stage and the remaining stages. The ETU works by locating an edge and its corresponding position in the 16-bit search vector every clock cycle. The positions of the edges found during the searching process are used to select the window and disparity range buffers (among the 16 candidates) corresponding to the edge points found; the selected buffers become the input of the next pipeline stage. It must be noted that the ETU requires from 1 cycle (in the best case) to 16 cycles (in the worst) to locate all edges in the search vector. During this period, the *edge_found* signal is set to 1 and the scanline buffers are disabled, so that the content of the candidate window and candidate disparity range buffers remains constant. When all edges in the search vector are located, the ETU sets the *edge_found* signal to 0. This informs the I/O controller to fetch new edges from the input FIFO queues and to shift the scanline buffers to the right.

4.1.6. DCU Pipeline Stages Description. The major pipeline stages of the DCU are described below.

(i) *INPUT/OUTPUT.* This pipeline stage fetches pixel data from the I/O port, executes the edge tracking process, and selects the windows corresponding to the edges found. The data fetched from the input port (the two 16-bit vectors produced by the EDU) is stored into FIFO queues. The I/O controller reads data from the input FIFO queues (if they are not empty) and forwards data to the scanline buffers (16-bits to each scanline buffer), until the first m scanlines from both images are stored into the scanline buffers. After the scanline buffers are filled, the I/O controller reads new pixel data from the queues only if the *edge_found* signal is set to 0 by the ETU. While the *edge_found* signal is set to 1, the scanline buffers are disabled, and during this period the edge tracking process described above is performed. If there is new data available at the input during this period, this data is written to the input queues. Furthermore, during this pipeline stage, the I/O controller writes the disparity value computed in the previous cycle to the output port.

(ii) *SADs_COMPUTATION.* The SAD values for all disparity levels are computed during this pipeline stage. The stage consists of d_M absolute difference (ABDIF) units, which compute the absolute difference between the $m \times m$ correlation window (stored in the Window Buffer) and the d_M $m \times m$ windows (stored in the disparity range buffer). Each ABDIF unit receives as input two m^2-bit vectors, whose elements are the edge points of the correlation windows, and consists of m^2 1-bit subtractors that compute the absolute difference of the edge points. The output of each ABDIF unit is an m^2-bit vector, which is next added bitwise using binary tree adders (BTA). Given the 11×11 maximum supported correlation window size, and 1-bit pixel intensities, the maximum value of the addition operation cannot be greater than 121. As such, the outputs of the BTA units are 7-bit values.

(iii) *MINSAD_COMPUTATION.* The SAD values for all disparity levels in the range $[1 : d_M]$ are compared with each other in order to compute the minimum value and its disparity. The comparison is carried out by a collection of 7-bit comparators and registers, arranged in tree structure to reduce the delay of the longest path. As stated previously, this stage was further divided into 3 pipeline stages in order to meet the targeted operating frequency (100 MHz). Figure 3 shows the circuit that computes the minimum SAD value and its disparity. Each minSAD unit receives as input two 14-bit vectors, each of which is a concatenation of an SAD value (7 bits) and its corresponding disparity (7 bits—up to 120 disparity levels). The minSAD unit compares the two SAD values and outputs the minimum of them along with

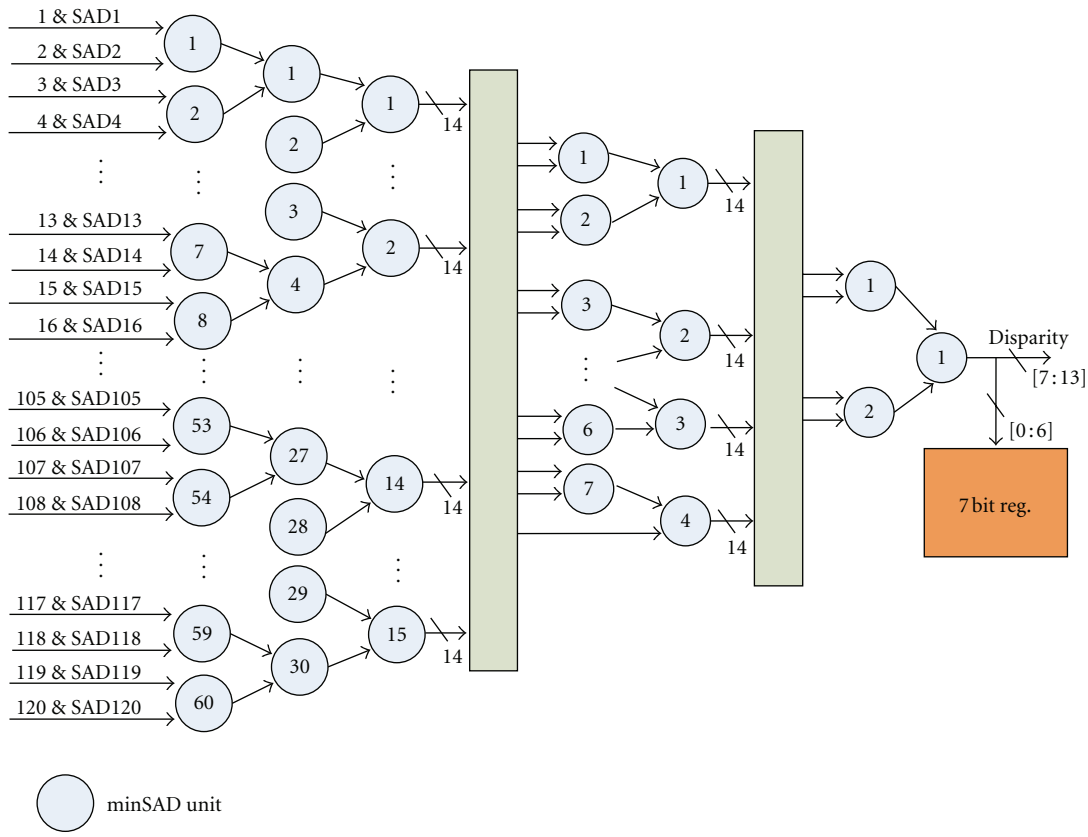

Figure 3: Minimum SAD value computation.

its disparity. The entire circuit for computing the minimum SAD value and its disparity consists of multiple minSAD units, arranged in a structure of a binary tree of $\log_2(d_M)$ levels.

4.2. Disparity Map Computation Hardware Architecture Based on the ADSW Algorithm

4.2.1. Design Issues and Requirements. This architecture implements an ADSW algorithm, which is computationally more expensive compared to the fixed-support algorithm both in terms of calculations and memory requirements. This is attributed in a significant way to the complex equation involved in the algorithm (2). In this work, we adopted some hardware optimization techniques that aim to simplify the algorithm and make it hardware-friendly and suitable for embedded constraints. We first eliminate the use of the spatial distance during the weight computation step based on our observation that it affects the accuracy of the disparity maps slightly. We also use YUV instead of CIELAB color representation during the computation of the weight coefficients. This allows the use of unsigned integers instead of signed floating-point integers, which are complex and hardware-unfriendly. Furthermore, the computation of the color distance between two YUV triplets is performed using Manhattan rather than Euclidean distance. In this way, the square and square root operations are replaced by simple absolute difference and addition operations. In addition, the $\exp(-x)$ function is approximated by the $\lfloor 2^{8-x} \rfloor$ function, which assigns a maximum weight of 256 if the color distance is zero and a weight of 0 if the color distance is greater than 8. This function simplifies the circuits that implement the multiplication of the weight coefficients with the pointwise scores, as multiplications are reduced to left shift operations. The cost function is further simplified by setting γ_c to a power of 2 (32 in our case). This converts the division to a right shift operation. Lastly, the denominator of (2) is approximated by the nearest power of 2 during the cost aggregation step, allowing the division to be replaced by a right shift operation.

4.2.2. General Structure. The architecture implementing the ADSW algorithm consists of two major pipeline stages: the *Input Stage* (IS) and the *Calculation Stage* (CS). The IS consists of a memory controller, 2 RGB to YUV color space converters (rgb2yuv) and 2 RGB to grayscale converters (rgb2gray). The memory controller fetches the RGB color values corresponding to the support windows W_r and W_t in a column-wise fashion (1 pixel value per input image every clock cycle) from the external memory. Those values are then converted to grayscale by the two rgb2gray units, and to their corresponding 8-bit YUV representation by the rgb2yuv units. The image values computed by the IS are temporarily stored into on-chip buffers (memory arrangements (MAs)),

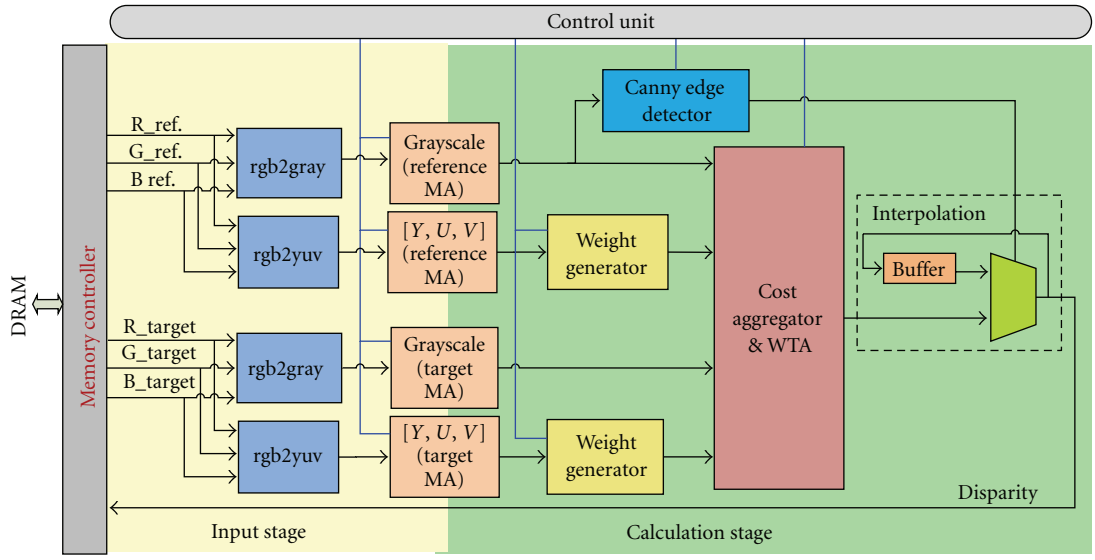

Figure 4: The architecture of the ADSW algorithm.

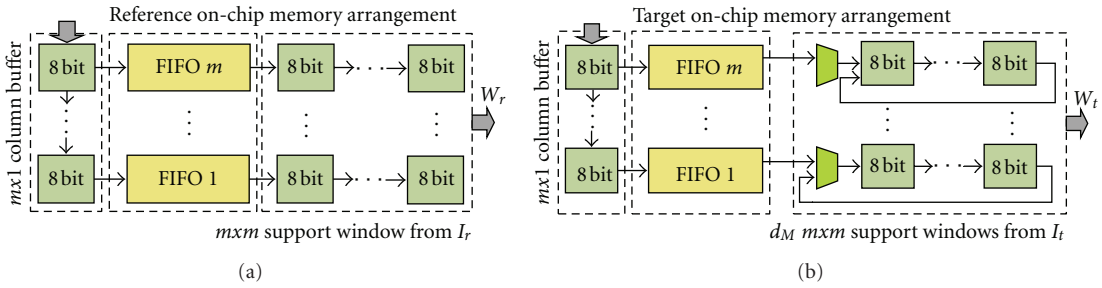

Figure 5: On-chip memory arrangements for the reference and target image data.

ensuring that there is always sufficient data for the CS, which is responsible for the calculation of the disparities. The overall system architecture also consists of a control unit that coordinates data transfers and handshakes between the different system units. Figure 4 shows a block diagram of the architecture and the data flow between units.

4.2.3. Memory Architecture. The IS and CS stages communicate through the use of on-chip buffers (memory arrangements (MAs)), which temporarily store the pixels required to perform correlation between W_r in I_r and the d_M candidate support windows W_t in I_t. The system is provided with 4 MAs per input image, which store the *Y, U, V* color values and the grayscale values. Figures 5(a) and 5(b) show the architectures of the on-chip MAs for the reference and target image, respectively. Both MAs consist of a column buffer, a series of FIFO queues, and window buffers. The column buffer consists of m 8-bit registers that store the pixels of an entire column of a support window. It receives one pixel per clock cycle and outputs a column every m cycles. The output column is stored in a series of m FIFO queues (1 pixel per queue), which are used to allow the memory controller to continuously fetch data from the external memory to the on-chip MAs (given that there is free space in the queues) irrespective of the data consumption rate of the CS. The CS consumes data at irregular rates; it consumes a column in d_M cycles if p is an edge and in a single cycle if p is not an edge.

The data from the queues is forwarded to the window buffers, which form the inputs of the CS. The window buffer of the reference MA consists of $m \times m$ 8-bit registers, while the window buffer of the target MA consists of $m \cdot (m + d_M - 1)$ registers (d_m is set to 0). The use of registers allows parallel access to the window buffers; after an initial delay of $m \cdot (m + d_M - 1)$ cycles per scanline (dominated by the cycles needed to fill in the window buffer of the target MA), both on-chip MAs can provide an $m \times m$ window per cycle. The window buffer of the target MA is organized in a cyclic structure and is provided with a series of multiplexers at its input, which determine whether the input comes from the FIFO queues or from the rightmost registers of the window buffer. This structure is adopted to enable data reuse.

4.2.4. Calculation Stage (CS). The CS consists of an Edge Detection Unit, two units for the generation of the weight coefficients (weight generators), a unit that computes the aggregated costs and selects the disparity with the minimum cost, and a unit responsible for the nearest neighbor (NN)

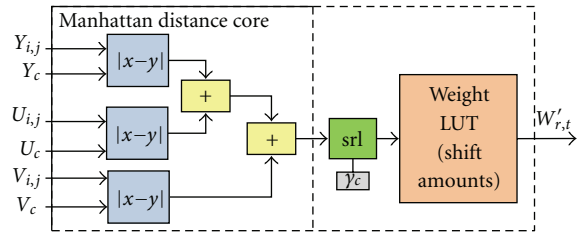

Figure 6: Circuit that computes the weight of a single pixel.

interpolation step. The Edge Detection Unit implements the Canny operator, which performs image smoothing using Gaussian convolution, vertical and horizontal gradient and angle calculation, nonmaximum suppression and thresholding. The detector employs hardware features such as parallelism and pipelining in order to provide an edge point every single clock cycle.

The weight generator computes the weight coefficients $w'_{r,t}$ for a support window $W_{r,t}$ in parallel, based on the YUV color values fetched from the $[Y,U,V]$ MAs, and by utilizing m^2 instances of the circuit shown in Figure 6. That circuit consists of a Manhattan distance core and a weight table (LUT). Since the multiplication of the pointwise scores by the weight coefficients (cost aggregation step) is performed using shifters instead of multipliers, each location x of the LUT stores the shift amount corresponding to the weight coefficient $\lfloor 2^{8-x} \rfloor$. This shift amount is equal to the binary logarithm of $\lfloor 2^{8-x} \rfloor$, except from values of x greater than 8, for which a binary logarithm does not exist. In that special case, the corresponding entries in the LUT are set to a number, which is large enough so that the result of a shift operation by that number is equal to zero. The weight coefficient $w'_{r,t}(i, j)$ is looked up in the LUT using the color distance generated by the Manhattan distance core as index.

The architecture of the cost aggregator and WTA unit is shown in Figure 7. The unit has been design in a parallel manner and utilizes m^2 absolute different circuits that compute the pointwise scores between corresponding pixels in W_r and W_t. Those scores are then shifted by the shift amounts corresponding to the weight coefficients w'_r and w'_t using a series of left shifters (equivalent to multiplying the scores by w'_r and w'_t). The final aggregated cost is computed by summing the outputs of the left shifters using a tree adder, and then normalizing (dividing) it by the weights sum, which, before being used for division, is rounded to the nearest power of 2 by using tree comparators. This enables a cost-effective implementation of the division using a right shifter. Finally, the WTA unit is responsible for selecting the disparity with the minimum cost.

5. Experimental Results

5.1. Experimental Platform and Methodology. The architectures presented in this paper have been implemented on the ML505 Evaluation Platform, which features a Virtex-5 LX110T FPGA. The basic features of the edge-directed disparity map computation architectures are listed in Table 1, while more details are presented in the following subsections. We evaluated the architectures using rectified synthetic and real-world data, initially stored in the compact flash memory card. The synthetic data includes stereo images from the Middlebury database [2] and the pedestrian data-set in [33], and the real-world data includes stereo images taken in the lab. The images were loaded into the on-board DRAM using the Microblaze soft-processor and were used as input to the systems shown in Figures 2 and 4, respectively. The resulting disparity maps were directed to a TFT monitor. The evaluation results of the synthetic images are shown in Figure 8, while the evaluation results of the real-world images are shown in Figure 9. The architectures were compared in terms of processing speed, disparity map accuracy, and FPGA resource utilization. They were also compared against their equivalent hardware systems that do not integrate the edge detectors in order to emphasize on the benefits of the edge-based approach adopted by the architectures.

Table 1: Features of the FPGA Prototypes.

Feature	Fixed-support-based architecture	ADSW-based architecture
Adopted algorithm	SAD-based Fixed-support algorithm	Adaptive-support weight algorithm
Image size	1280 × 1024 pixels (8-bit grayscale)	800 × 600 pixels (24-bit RGB color images)
Disparity range	120	64
Support window size	11 × 11	11 × 11
Device	Virtex-5 LX110T FPGA	Virtex-5 LX110T FPGA
Frequency	100 MHz	155 MHz
Memory bandwidth	32 bits/cycle	48 bits/cycle
Processing speed	150 fps	30 fps

5.2. Disparity Map Quality-Impact of Edge Detection. To evaluate the quality of the disparity maps generated by the proposed edge-directed hardware architectures, and to examine the impact of the edge detection algorithm in the overall system quality, we use Middlebury stereo pairs, as well as stereo pairs from [33], for which the ground truth disparity maps are known and measure the incorrect disparity estimates using the percentage of bad pixels, a commonly accepted metric [2].

In our previous work in [7], we have investigated the impact of three different edge detectors (Sobel, Canny, and Evolvable [34]) on the accuracy of the disparity maps generated by an SAD-based, fixed-support algorithm, in order to select the best detector to be integrated to the system described in [7]. As we discussed in our previous paper, we have selected the Sobel detector, both for its simplicity on an FPGA and also for its good accuracy. In this paper, we also investigate the impact of the same detectors on the ADSW algorithm. The results about the percentage of bad pixels for the Tsukuba and Venus stereo image pairs are given in

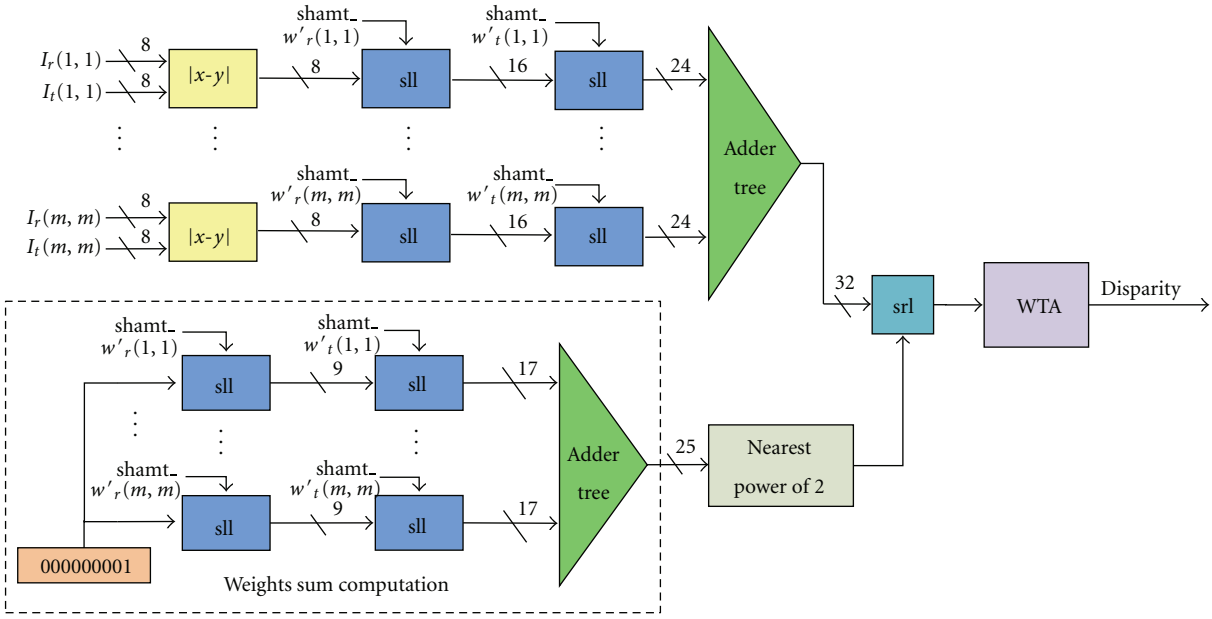

Figure 7: Architecture of the cost aggregator and WTA.

Table 2: Quality comparison for different edge detectors.

Algorithm	Percentage of bad pixels (%)					
	Canny		Sobel		Evolvable	
	Tsukuba	Venus	Tsukuba	Venus	Tsukuba	Venus
Fixed-support	10.30	17.03	8.73	16.73	11.04	12.81
ADSW	7.39	8	5.78	14.61	8.10	8.90

Table 3: Different system configurations.

System architecture	Adopted method
1	SAD-based fixed support without edge detector
2	SAD-based fixed support with Sobel edge detector
3	SAD-based adaptive support weight without edge detector
4	SAD-based adaptive support weight with Canny edge detector

Table 2. The average percentages of data reduction associated to the results presented in the table are 82.95%, 70.23%, and 57.67% for the Canny, Sobel, and Evolvable detectors, respectively.

As can be seen, the Canny detector achieves on average better accuracy (lower percentage of bad pixels) compared to the other two edge detectors when is being applied to the ADSW algorithm, but the quality of the disparity maps is lower when using the Canny detector in the fixed-support algorithm. This difference in the behavior of the Canny detector in the two algorithms lies in the fact that the fixed-support algorithm utilizes edges both as directive points and also as matching primitives. As a result, the fixed support algorithm works better if the edge detector preserves not only strong edges, but also "busy" texture regions (edge regions with numerous small edge elements), which aid the matching process. The Canny detector does not work well with the fixed support algorithm as it is less likely to be fooled by noise, and more likely to detect true weak edges. As such, we have selected the Canny detector to be integrated into the ADSW-based disparity map computation architecture.

For a detailed quality analysis, we compare the quality of the disparity maps generated by the edge-directed architectures to the disparity maps generated by their equivalent hardware architectures that do not integrate the edge detectors (stand-alone fixed-support and ADSW architectures). The stand-alone fixed-support architecture has a similar structure with the edge-directed architecture shown in Figure 2 (without the EDU, the ETU and the candidate window and disparity range buffers), which is able to process 8-bit pixels instead of edges. The stand-alone ADSW architecture has a similar structure with the edge-directed architecture shown in Figure 4 (without the edge detection and interpolation units). For simplicity purposes, we will refer to each hardware system configuration as "System 1–4," as defined in Table 3.

The results extracted by comparing the disparity maps of Systems 1–4 to the ground truth disparity maps are presented in Table 4 for two sample image pairs. The resulting disparity maps are also given in Figure 10 for a qualitative comparison. Obviously, the disparity maps generated by the systems that implement the ADSW algorithm are more accurate, and particularly at depth discontinuity regions and at regions with repetitive patterns, where ADSW algorithms traditionally work better. The results also indicate that the use of the edge detectors does not impact negatively the accuracy of the disparity maps, but, in some cases, they improve the accuracy and especially at depth borders. This is because

Figure 8: Evaluation results for synthetic images. From left to right: Tsukuba, Venus, and pedestrian stereo images. From top to bottom: reference image, target image, output of the Sobel detector, output of the Canny detector, disparity maps generated by the fixed-support-based architecture, disparity maps generated by ADSW-based architecture, ground truth disparity maps.

the stand-alone architectures suffer from changes in pixels intensities, while the edge-directed architectures potentially struggle this problem by limiting the correspondence search to specific reliable features in the images (edges in our case). It must be noted that the aforementioned results are based on a nearest neighbor interpolation method. It is anticipated that the proposed edge-directed hardware systems can produce disparity maps with better quality by integrating more complex interpolation methods such as bilinear and bicubic interpolation; this however is left as part of future work.

5.3. Processing Speed. The processing speed of the edge-directed disparity computation architectures described in

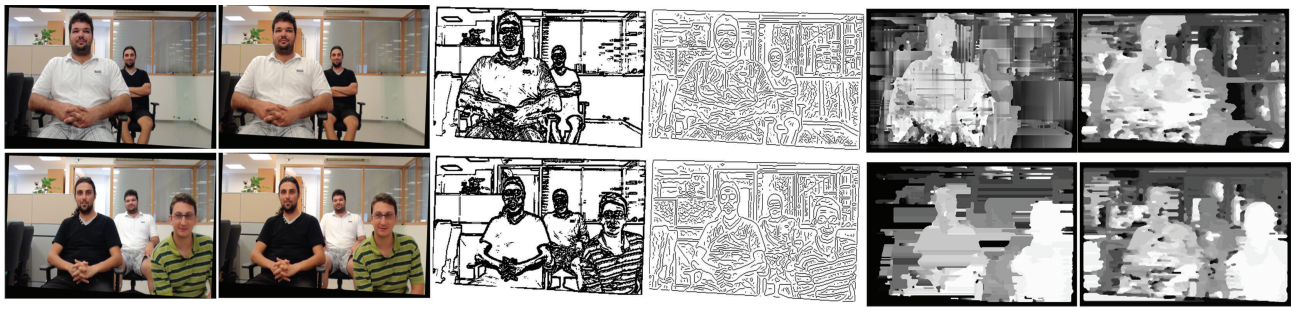

Figure 9: Evaluation results for real-world images. From left to right: reference image, target image, output of the Sobel detector, output of the Canny detector, disparity map generated by the fixed-support-based architecture, disparity map generated by the ADSW-based architecture.

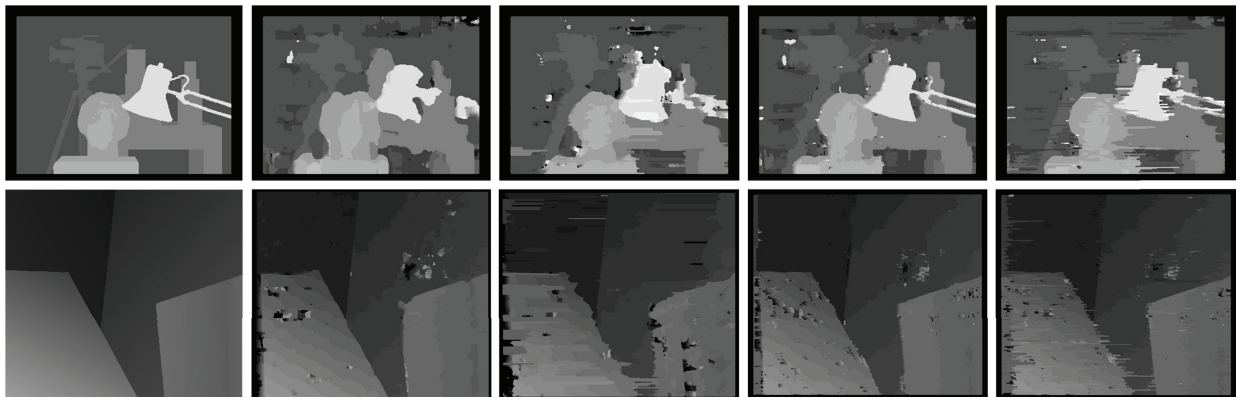

Figure 10: Disparity maps of the Tsukuba and Venus image pairs for the system configurations listed in Table 3. From left to right: ground truth disparity map and disparity maps generated by System 1, System 2, System 3, and System 4, respectively.

Section 4 is measured in frames per seconds (fps) and is affected by several algorithmic and implementation-specific parameters. The algorithmic parameters include the amount of data reduction (the percentage of non-edge points over the total image points) obtained from the edge detectors, the image size and the representation of the input images (grayscale or color), as well as the disparity range. The implementation-specific parameters include the operating frequency, the I/O bandwidth from the external memory, and the available FPGA resources.

The architecture based on the fixed-support approach (System 2) has lower requirements in terms of input bandwidth, as it processes grayscale stereo images. Therefore, the external memory bandwidth can be exploited to fetch multiple pixels per clock cycle compared to the architecture that adopts the ADSW algorithm (System 4), which fetches one pixel per input image. The frame rate of System 2 is also independent from the support window size and the disparity range, as that system utilizes the edge information not only as directive points, but also as matching primitives, thus exploiting more parallelism. System 2 can compute the aggregated costs for all d_M support windows in a single clock cycle. System 4 requires d_M clock cycles to compute the costs for all disparity levels as that system processes color stereo images and uses grayscale pixels as matching primitives. Thus, it can exploit much less parallelism when assuming a constant amount of FPGA resources; particularly, it processes a support window every clock cycle, but requires d_M clock cycles to compute a disparity value in the case where the processing pixel represents an edge. System 4, however, performs the matching process on a smaller number of edge points, since it is directed by the Canny detector, which achieves larger percentage of data reduction compared to the Sobel detector. With respect to the I/O bandwidth and the image size, the performance of both system architectures decreases as the image size increases and the I/O bandwidth decreases, since in the former case there is more data to be processed, while, in the latter case, the amount of data flowing into the system limits the system throughput.

To evaluate the processing speed of the edge-based architectures, we compare them with their equivalent system architectures that do not integrate the edge detectors (stand-alone architectures). We identify the speedup of the proposed architectures compared to the stand-alone architectures and provide results when increasing the input image size and the disparity range, in order to illustrate the impact of the edge detector in all cases. We used synthetic stereo images as benchmarks from [2, 33] in order to extract the processing speed of the systems mentioned above. The results are given in Tables 5 and 6.

As it can be seen, the processing speed is inversely proportional to the image size in all cases. Another particular

Table 4: Comparison of disparity map quality between stand-alone and edge-directed architectures.

Image pair	Percentage of bad pixels											
	System 1*			System 2			System 3*			System 4		
	nonocc. (%)	all (%)	disc. (%)	nonocc. (%)	all (%)	disc. (%)	nonocc. (%)	all (%)	disc. (%)	nonocc. (%)	all (%)	disc. (%)
Tsukuba	8.45	10.4	29.5	8.78	10.5	27.4	7.95	9.54	28.9	7.91	8.60	26
Venus	6.33	7.95	46.0	12.3	13.7	32.3	4.41	5.53	31.7	8.78	9.66	27.9

*Disparity map quality results for System 1 and System 3 were extracted using Matlab simulation.

Table 5: Frame rate versus image size for Systems 1–4 (disparity range = 64, support window size = 11 × 11).

Image size	*w/o* edge detector*		*w*/edge detector	
	Fixed-support	ADSW	Fixed-support	ADSW
320 × 240	1225	34	2570	174
640 × 480	315	8	645	45
800 × 600	203	5	412	30
1024 × 768	124	3	252	18
1280 × 1024	75	1	150	7

*The frame rates for the systems without the edge detectors are projected frame rates.

Table 6: Frame rate versus disparity range for Systems 1–4 (image size = 800 × 600, support window size = 11 × 11).

Disparity range	*w/o* edge detector*		*w*/edge detector	
	Fixed-support	ADSW	Fixed-support	ADSW
16	203	21	412	102
32	203	10	412	59
64	203	5	412	30
128	203	3	412	16

*The frame rates for the systems without the edge detectors are projected frame rates.

observation is that the architectures based on the ADSW approach are slower even if they operate on higher operating frequencies and have larger memory bandwidth; this is due to the high computational complexity of the ADSW algorithm and due to the limited parallelism that can be exploited by such an algorithm in a resource constraint implementation platform such as an FPGA. The most important observation is that the use of the edge detection offers significant speedups, when compared to the stand-alone architectures for all image sizes. The impact of the edge detector is even more emphasized in the architecture based on the ADSW algorithm, as the data reduction of the Canny detector is larger compared to the Sobel. While the use of the edge detector in System 4 yields a speedup of ∼5, the speedup obtained in System 2 is ∼2. This is because System 2 was designed with more parallelism and provides a disparity value every clock cycle, irrespective of whether or not the pixel being processed is an edge. As a result, the edge detector in System 2 needs to provide multiple edges per clock cycle, so the bottleneck in this case is the Edge Detection Unit. Of course, the EDU in System 2 can further be parallelized, so the real limitation in terms of parallelism is the external memory I/O. On the other hand, System 4 exploits parallelism only on the level of a support window. If the pixel being processed is an edge, it requires d_M cycles to compute the disparity, so the edge detector used in System 4 needs to provide only one edge every clock cycle. Table 6 indicates that the disparity range affects only the ADSW-based system architectures, due to the limited parallelism that can be exploited on the targeted FPGA. This however does not necessarily imply that the architecture is not scalable. The parallelism is limited by the amount of hardware resources. Of course, the ADSW-based architecture could be implemented to process multiple support windows in parallel on a larger FPGA. This would increase the frame rate as well.

Table 7 presents a comparison between existing implementations ([12, 13, 15, 16, 19, 20, 22–31]) and the proposed FPGA prototypes (System 2 and System 4). Performance is provided in frames per second (fps), as well as in points × disparity estimates per second (PDS). The proposed edge-directed architecture that implements the fixed-support algorithm (System 2) achieves a PDS of 23592×10^6 and is the fastest implementation listed in the table, when considering the input image size and the maximum disparity range supported. System 4 achieves a PDS of 922×10^6 for an image size of 800×600 and a disparity range of 64. Such PDS seems sufficient considering that the algorithm implemented by System 4 is much more complex compared to the algorithms implemented by the other implementations in the table; only [30] implements an ADSW algorithm and achieves lower performance rates. Conclusively, performance results indicate that the proposed edge-directed approach for accelerating hardware implementations of stereo correspondence algorithms exhibits high potential for applications with hard real-time constraints. System 2 can meet the real-time requirements of such applications, even for high-resolution images. Moreover, the combination of the edge-directed approach with the ADSW algorithm enables the realization of accurate disparity computations systems that can also satisfy the real-time requirements of many embedded vision applications such as pedestrian and obstacle detection.

5.4. Hardware Overheads. The proposed FPGA systems were evaluated for relevant metrics such as area and operating frequency. Table 8 gives the overall hardware demands of the FPGA prototype that is based on the fixed-support algorithm (System 2). The table also lists the hardware overheads associated with each of the implemented components.

Table 7: Comparison of PDS performance for various systems and methods.

Work	Image size	Disparity Range	Frame rate (fps)	PDS (10^6)	Algorithm	Platform
Yang and Pollefeys [12]	512 × 512	32	N/A	289	Block-matching (SAD)	ATI Radeon 9800 graphics card
Yang et al. [13]	384 × 288	16	12.77	22.2	Hierarchical belief propagation	NVIDIA Geforce 7900 GTX graphics card
Khaleghi et al. [15]	160 × 120	30	20	11.5	Census transform based	BlackFin processor-ADSP-BF561 (600 MHz)
Zinner and Humenberger [16]	450 × 375	60	11.8	119.5	Census transform based	Texas Instruments TMS320C6414 DSP
Hile and Zheng [19]	512 × 480	32	30	235.9	Block-matching (SAD)	N/A
Miyajima and Maruyama [20]	640 × 480	80	26	639	Block-matching (SAD)	ADM-XRC-II (40 MHz)
Arias-Estrada and Xicotencatl [22]	320 × 240	16	71	87.2	Block-matching (SAD)	XCV800HQ240-6 (66 MHz)
Lee et al. [23]	640 × 480	64	30	589	Block-matching (SAD)	XC2V8000 (10 MHz)
Hariyama et al. [24]	64 × 64	64	5063	1327.2	Block-matching (SAD)	APEX20KE (86 MHz)
Georgoulas and Andreadis [25]	800 × 600	80	550	21120	Block-matching (SAD)	EP4SGX290 (511 MHz)
Ambrosch et al. [26]	450 × 375	100	600	10125	Block-matching (SAD)	EP2S130 (110 MHz)
Díaz et al. [27]	1280 × 960	29	52	1885	Phase based	Custom FPGA, Xilinx Virtex-II (65 MHz)
Darabiha et al. [28]	256 × 360	20	30.3	55.2	LWPC (phase correlation)	TM-3A board (Xilinx Virtex-4 2000E FPGA)
Jin et al. [29]	640 × 480	64	230	4522	Census transform	Virtex-5 XC4VLX200-10 FPGA (93.1 MHz)
Chang et al. [30]	352 × 288	64	42	272.5	Minicensus adaptive support weight	UMC 90ns Std. Cell
Ambrosch and Kubinger [31]	750 × 400	60	60	1080	SAD-IGMCT	Altera Stratix I (133 MHz)
Proposed (System 2)	1280 × 1024	120	150	23592	Edge-based fixed support weight block matching (SAD)	ML505 Evaluation Board with Virtex-5 LX110T FPGA (100 MHz)
Proposed (System 4)	800 × 600	64	30	922	Edge-based adaptive support weight block matching (SAD)	ML505 Evaluation Board with Virtex-5 LX110T FPGA (155 MHz)

Table 8: Hardware overheads of the edge-directed fixed-support-based architecture (System 2) (image size = 1280 × 1024, max disparity range = 120, window size = 11 × 11).

Design unit	Slice LUTs (69120)	Slice registers (69120)	Block RAMs (148)	DSP48Es (64)	Frequency (MHz)
Edge Detection Unit (EDU)	2812 (∼4%)	1008 (∼3%)	0	0	134.6
ABDIF and BTA	129 (∼0.2%)	0	0	0	N/A
Tree Comparators	1760 (∼2.5%)	329 (∼0.5%)	0	0	174.3
Disparity Computation Unit (DCU)	47331 (∼68%)	31863 (∼46%)	0	0	109
Microblaze system and external Memory CONTROLLER	7754 (∼11%)	8562 (∼12%)	30 (∼20%)	6 (∼9%)	161.2
Entire system	**66882** (∼**83.8**%)	**41559** (∼**60**%)	**30** (∼**20**%)	**6** (∼**9**%)	**100**

The entire system utilizes ∼83.8% of the FPGA slice LUTs and ∼60% of the FPGA slice registers. The more hardware demanding component is the DCU, as obviously anticipated, due to the large amount of computations performed by that unit. The EDU requires only a small amount of resources, despite the fact that it processes two pixels per input image. This is attributed to the simplicity of the circuits that compute the convolution operation (using shifters instead of multipliers). Table 9 gives the overall hardware demands of the FPGA prototype that is based on the ADSW algorithm (System 4). That prototype utilizes ∼73.1% of the FPGA LUTs and ∼64.8% of the FPGA slice registers and can operate

Table 9: Hardware overheads of the edge-directed ADSW-based architecture (System 4) (image size = 800 × 600, max disparity range = 64, support window size = 11 × 11).

Design unit	Slice LUTs (69120)	Slice registers (69120)	Block RAMs (148)	DSP48Es (64)	Frequency (MHz)
rgb2gray	0	0	3 (∼4.7%)	0	N/A
rgb2yuv	261 (∼0.4%)	0	0	0	N/A
Weight generator	7056 (∼10.2%)	0	0	0	N/A
Cost aggregator	30691 (∼44.4%)	434 (∼0.6%)	0	0	391
WTA	83 (∼0.1%)	61 (∼0.1%)	0	0	429
On-chip MA (left)	228 (∼0.3%)	2304 (∼3.3%)	0	0	632
On-chip MA (right)	644 (∼0.9%)	2574 (∼3.7%)	0	0	582
Canny detector	2060 (∼3%)	947 (∼1.4%)	0	0	165
Microblaze system & external memory Controller	7754 (∼11.2%)	8562 (∼12.4%)	6 (∼9.4%)	30 (∼20.3%)	161
Complete system	**50539** (∼**73.1**%)	**44802** (∼**64.8**%)	**12** (∼**18.8**%)	**30** (∼**20.3**%)	**155**

at 155 MHz. The slice LUTs are dominated by the cost aggregator and the weight generators, which consume ∼44% of the available LUTs. The slice registers are dominated by the on-chip MAs, which consume ∼28% of the available slice registers.

A closer look in Tables 8 and 9 leads us to the following useful observations.

(i) Even though System 2 exploits parallelism aggressively in order to compute the cost values for all support windows in the range $[1\ d_M]$ in a clock cycle, it requires only ∼10% more FPGA LUTs than those required by System 4. This indicates that the edge-based approach for stereo correspondence is much more efficient (in terms of hardware usage) when applied to a fixed-support algorithm, mainly due to the fact that it performs correlation using binary data (edges), thus requiring very simple circuits during the cost computation and aggregation steps.

(ii) While the Sobel detector is computationally less demanding than Canny, it requires ∼1% more FPGA LUTs. This is due to the fact that the EDU is able to process 4 pixels per clock cycle (2 per input image); thus, the hardware resources have been replicated by a factor of 4. The Canny detector, on the other hand, outputs only a pixel per cycle.

(iii) The EDU operates at a slower clock frequency compared to the Canny detector. It could be possible to divide the EDU into more pipeline stages so that to increase its frequency, at the expense of further hardware usage. However, this would be meaningless considering that the DCU works at a slower frequency. Moreover, the frame rates achieved by System 2 are extremely high, and therefore any further increase of the operating frequency to achieve higher frame rates would not justify the extra hardware cost.

6. Conclusion

Stereo correspondence is an important algorithm in several embedded vision applications that require real-time computation of depth information. This paper presented an overview of the use of edge information, as a means to accelerate hardware implementations of stereo correspondence algorithms. The paper analyzed design issues and considerations associated with the edge-directed approach when it was applied to different hardware implementations of both fixed-support and even more complex ADSW algorithms.

Our immediate plans involve the integration of other interpolation methods into the proposed architectures in order to achieve better disparity map accuracy. We also plan to investigate the applicability of the presented edge-directed approach to global stereo correspondence algorithms, which return better quality on the disparity map. Furthermore, the FPGA prototypes will be extended to full-custom ASIC designs in order to evaluate large-scale performance and power.

References

[1] B. Cyganek and J. P. Siebert, *Introduction to 3D Computer Vision Techniques and Algorithms*, Wiley, John & Sons, 2009.

[2] D. Scharstein and R. Szeliski, "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms," *International Journal of Computer Vision*, vol. 47, no. 1–3, pp. 7–42, 2002.

[3] M. Z. Brown, D. Burschka, and G. D. Hager, "Advances in computational stereo," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 25, no. 8, pp. 993–1008, 2003.

[4] K.-J. Yoon and I.-S. Kweon, "Adaptive support-weight approach for correspondence search," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 4, pp. 650–656, 2006.

[5] F. Tombari, S. Mattoccia, and L. Di Stefano, "Segmentation-based adaptive support for accurate stereo correspondence," in *Proceedings of the 2nd Pacific Rim Conference on Advances*

in Image and Video Technology, vol. 4872 of *Lecture Notes in Computer Science*, pp. 427–438, Springer, December 2007.

[6] M. Gerrits and P. Bekaert, "Local stereo matching with segmentation-based outlier rejection," in *Proceedings of the 3rd Canadian Conference on Computer and Robot Vision (CRV '06)*, p. 66, June 2006.

[7] C. Ttofis, S. Hadjitheophanous, A. S. Georghiades, and T. Theocharides, "Edge-directed hardware architecture for real-time disparity map computation," in *Proceedings of the IEEE Transactions on Computers*, 2012.

[8] K. Mühlmann, D. Maier, J. Hesser, and R. Männer, "Calculating dense disparity maps from color stereo images, an efficient implementation," *International Journal of Computer Vision*, vol. 47, no. 1–3, pp. 79–88, 2002.

[9] S. Forstmann, Y. Kanou, J. Ohya, S. Thuering, and A. Schmitt, "Real-time stereo by using dynamic programming," in *Proceedings of the Computer Vision and Pattern Recognition Workshop (CVPRW '04)*, p. 29, June 2004.

[10] L. D. Stefano, M. Marchionni, and S. Mattoccia, "A fast area-based stereo matching algorithm," *Image and Vision Computing*, vol. 22, no. 12, pp. 983–1005, 2004.

[11] H. Hirschmüller, P. R. Innocent, and J. Garibaldi, "Real-time correlation-based stereo vision with reduced border errors," *International Journal of Computer Vision*, vol. 47, no. 1–3, pp. 229–246, 2002.

[12] R. Yang and M. Pollefeys, "A versatile stereo implementation on commodity graphics hardware," *Real-Time Imaging*, vol. 11, no. 1, pp. 7–18, 2005.

[13] Q. Yang, L. Wang, R. Yang, S. Wang, M. Liao, and D. Nister, "Real-time global stereo matching using hierarchical belief propagation," in *Proceedings of the The British Machine Vision Conference*, 2006.

[14] R.-P. M. Berretty, A. K. Riemens, and P. F. Machado, "Real-time embedded system for stereo video processing for multi-view displays," in *Proceedings of the Stereoscopic Displays and Virtual Reality Systems XIV*, Proceeding of SPIE, San Jose, Calif, USA, January 2007.

[15] B. Khaleghi, S. Ahuja, and Q. M. J. Wu, "An improved real-time miniaturized embedded stereo vision system (MESVS-II)," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops, CVPR Workshops*, pp. 1–8, June 2008.

[16] C. Zinner and M. Humenberger, "Distributed real-time stereo matching on smart cameras," in *Proceedings of the 4th ACM/IEEE International Conference on Distributed Smart Cameras (ICDSC '10)*, pp. 182–189, New York, NY, USA, September 2010.

[17] S. Cavanag and M. Manzke, "Real time disparity map estimation on the cell processor," in *Proceedings of the Eurographics Ireland Workshop 2009*, pp. 67–74, Trinity College Dublin, December 2009.

[18] J. Liu, Y. Xu, R. Klette, H. Chen, W. Rong, and T. Vaudrey, "Disparity map computation on a cell processor," in *Proceedings of the IASTED International Conference on Modelling, Simulation, and Identification (MSI '09)*, Beijing, China, October 2009.

[19] H. Hile and C. Zheng, "Stereo video processing for depth map," Tech. Rep., University of Washington, 2004.

[20] Y. Miyajima and T. Maruyama, "A real-time stereo vision system with FPGA," in *Proceedings of the 13th International Conference on Field-Programmable Logic and Applications (FPL '03)*, pp. 448–457, Lisbon, Portugal, 2003.

[21] L. Nalpantidis, G. Sirakoulis, and A. Gasteratos, "Review of stereo matching algorithms for 3D vision," in *Proceedings of the 16th International Symposium on Measurement and Control in Robotics (ISMCR '07)*, pp. 116–124, Warsaw, Poland, June 2007.

[22] M. Arias-Estrada and J. M. Xicotencatl, "Multiple stereo matching using an extended architecture," in *Proceedings of the Field-Programable Logic and Applications*, vol. 2778, pp. 203–212, Springer, 2003.

[23] S. H. Lee, J. Yi, and J. Kim, "Real-time stereo vision on a reconfigurable system," in *Proceedings of the Embedded Computer Systems: Architectures, Modelling and Simulation*, vol. 3553, pp. 299–307, Springer, 2005.

[24] M. Hariyama, Y. Kobayashi, M. Kameyama, and N. Yokoyama, "FPGA implementation of a stereo matching processor based on window-parallel-and-pixel-parallel architecture," in *Proceedings of the IEEE International 48th Midwest Symposium on Circuits and Systems, (MWSCAS '05)*, vol. 2, pp. 1219–1222, Covington, Ky, USA, August 2005.

[25] C. Georgoulas and I. Andreadis, "A real-time occlusion aware hardware structure for disparity map computation," in *Proceedings of the Image Analysis and Process (ICIAP '09)*, vol. 5716, pp. 721–730, 2009.

[26] K. Ambrosch, M. Humenberger, W. Kubinger, and A. Steininger, "A SAD-based stereo matching using FPGAs," in *Proceedings of the Embedded Computer Vision part II*, pp. 121–138, Spinger, 2009.

[27] J. Díaz, E. Ros, R. Carrillo, and A. Prieto, "Real-time system for high-image resolution disparity estimation," *IEEE Transactions on Image Processing*, vol. 16, no. 1, pp. 280–285, 2007.

[28] A. Darabiha, J. MacLean, and J. Rose, "Reconfigurable hardware implementation of a phase-correlation stereoalgorithm," *Machine Vision and Applications*, vol. 17, no. 2, pp. 116–132, 2006.

[29] S. Jin, J. Cho, X. D. Pham et al., "FPGA design and implementation of a real-time stereo vision system," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 20, no. 1, pp. 15–26, 2010.

[30] N. Y. C. Chang, T. H. Tsai, B. H. Hsu, Y. C. Chen, and T. S. Chang, "Algorithm and architecture of disparity estimation with mini-census adaptive support weight," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 20, no. 6, pp. 792–805, 2010.

[31] K. Ambrosch and W. Kubinger, "Accurate hardware-based stereo vision," *Computer Vision and Image Understanding*, vol. 114, no. 11, pp. 1303–1316, 2010.

[32] M. Raman and H. Aggarwal, "Study and comparison of various image edge detection techniques," *The International Journal of Image Processing*, vol. 3, no. 1, pp. 1–12, 2009.

[33] "A Framework for Evaluating Stereo-Based Pedestrian Detection Techniques," http://www.cdvp.dcu.ie/datasets/pedestrian_detection/.

[34] Z. Vasicek and L. Sekanina, "An evolvable hardware system in Xilinx Virtex II Pro FPGA," *International Journal of Innovative Computing and Applications*, vol. 1, no. 1, pp. 63–73, 2007.

Hindawi Publishing Corporation
VLSI Design
Volume 2012, Article ID 209208, 12 pages
doi:10.1155/2012/209208

Research Article

Optimized Architecture Using a Novel Subexpression Elimination on Loeffler Algorithm for DCT-Based Image Compression

Maher Jridi,[1] Ayman Alfalou,[2] and Pramod Kumar Meher[3]

[1] *Vision Department, L@bIsen, ISEN–Brest, CS 42807, 29228 Brest Cedex 2, France*
[2] *Vision Department, L@bIsen, ISEN–Brest, CS 42807, 29228 Brest Cedex2, France*
[3] *Department of Embedded Systems, Institute for Infocomm Research, Singapore 138632*

Correspondence should be addressed to Maher Jridi, maher.jridi@isen.fr

Received 17 November 2011; Revised 17 February 2012; Accepted 3 March 2012

Academic Editor: Muhammad Shafique

The canonical signed digit (CSD) representation of constant coefficients is a unique signed data representation containing the fewest number of nonzero bits. Consequently, for constant multipliers, the number of additions and subtractions is minimized by CSD representation of constant coefficients. This technique is mainly used for finite impulse response (FIR) filter by reducing the number of partial products. In this paper, we use CSD with a novel common subexpression elimination (CSE) scheme on the optimal Loeffler algorithm for the computation of discrete cosine transform (DCT). To meet the challenges of low-power and high-speed processing, we present an optimized image compression scheme based on two-dimensional DCT. Finally, a novel and a simple reconfigurable quantization method combined with DCT computation is presented to effectively save the computational complexity. We present here a new DCT architecture based on the proposed technique. From the experimental results obtained from the FPGA prototype we find that the proposed design has several advantages in terms of power reduction, speed performance, and saving of silicon area along with PSNR improvement over the existing designs as well as the Xilinx core.

1. Introduction

Many applications such as video surveillance and patient monitoring systems require many cameras for effective tracking of living and nonliving objects. To manage the huge amount of data generated by several cameras, we proposed an optical implementation of an image compression based on DCT algorithm in [1]. But this solution suffers from bad image quality and higher material complexity. After this optical implementation, in this paper we propose a digital realization of an optimized VLSI for image compression system. This paper is an extension of our prior work [2–4] with a new compression scheme along with supplementary simulations and FPGA implementation followed by performance analysis.

More recent video encoders such as H.263 [5] and MPEG-4 Part 2 [6] use the DCT-based image compression along with additional algorithms for motion estimation (ME). A simplified block diagram of the encoder is presented in Figure 1. The 2D DCT of 8×8 blocks of the image is performed to decorrelate each block of input pixels. The DCT coefficients are then quantized to represent them in a reduced range of values using a quantization matrix. Finally, the quantized components are scanned in a zigzag order, and the encoder employs run-length encoding (RLE) and Huffman coding/binary arithmetic coding (BAC-) based algorithms for entropy coding.

Since the DCT computation and quantization processes are computation intensive, several algorithms are proposed in literature for computing them efficiently in dedicated hardware. Research in this domain can be classified into three parts. The first part is the earliest and concerns the reduction of the number of arithmetic operators required for DCT computation [7–13]. The second research thematic relates to the computation of DCT using multiple constant multiplication schemes [14–25] for hardware implementation. Some other works on design of architectures for DCT make use of convolution formulation. They are efficient but can be used only for prime-length DCT and not suitable for video processing applications [26, 27]. Finally, the third part is about the optimization of the DCT computation in the

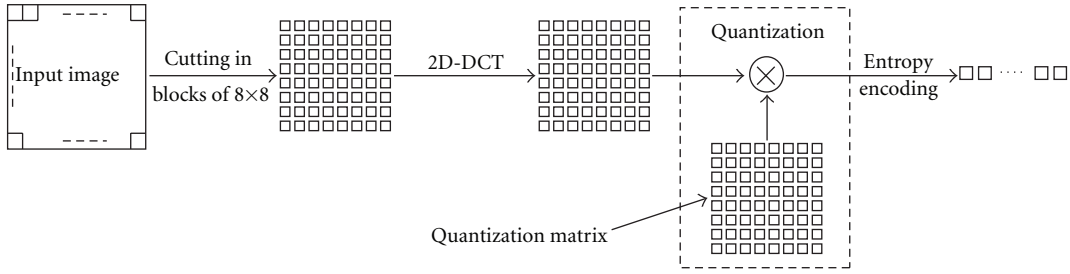

Figure 1: Simplified block diagram of the encoder [2].

context of image and video encoding [28–32]. In this paper, we are interested in the last research thematic.

In this paper, we propose a novel architecture of the DCT based on the canonical signed digit (CSD) encoding [33, 34]. Hartley in [35] has used CSD-based encoding and common subexpression elimination (CSE) for efficient implementation of FIR filter. The use of similar CSD and CSE technique for DCT implementation is not suitable. To improve the efficiency of implementation, we identify multiple subexpression occurrences in intermediate signals (but not in constant coefficients as in [35]) in order to compute DCT outputs. Since the calculation of multiple identical subexpression needs to be implemented only once, the resources necessary for these operations can be shared and the total number of required adders and subtractors can be reduced.

The second contribution of the paper is an introduction of a new schema of image compression where the second stage of 1D DCT (DCT on the columns) is configured for joint optimization of the quantization and the 2D DCT computation. Moreover, tradeoffs between image visual quality, power, silicon area, and computing time are analysed.

The remainder of the paper is organized as follows: an overview of fundamental design issues is given in Section 2. Proposed DCT optimization based on CSD and subexpression sharing is described in Section 3. An algorithm based on joint optimization of quantization and 2D DCT computation is proposed in Section 4. Finally, the experimental results are detailed in the Section 5 before the conclusion.

2. Background

Given an input sequence $\{x(n)\}$, $n \in [0, N-1]$, the N-point DCT is defined as:

$$X(n) = \sqrt{\frac{2}{N}} C(n) \sum_{k=0}^{N-1} x(k) \cos \frac{(2k+1)n\pi}{2N}, \quad (1)$$

where $C(0) = 1/\sqrt{2}$ and $C(n) = 1$ if $n \neq 0$.

As stated in the introduction, we find two main types of algorithms for DCT computation. One class of algorithms is focused on reducing the number of required arithmetic operators, while the other class of algorithms are designed for hardware implementation of DCT. In this Section, we provide a brief review of the major developments of different types of algorithms.

2.1. Fast DCT Algorithm. In literature, many fast DCT algorithms are reported. All of them use the symmetry of the cosine function to reduce the number of multipliers. In [36] a summary of these algorithms is presented. In Table 1, we have listed the number of multipliers and adder involved in different DCT algorithms. In [13], the authors show that the theoretical lower limit of 8-point DCT algorithm is 11 multiplications. Since the number of multiplications of Loeffler's algorithm [12] reaches the theoretical limit, our work is based on this algorithm.

Loeffler et al. in [12] proposed to compute DCT outputs on four stages as shown in Figure 2. The first stage is performed by 4 adders and 4 subtractors while the second one is composed of 2 adders, 2 subtractors, and 2 MultAddSub (multiplier, adder and subtractor) blocks. Each MultAddSub block uses 4 multiplications and can be reduced to 3 multiplications by constant arrangements. The fourth stage uses 2 MultSqrt(2) blocks to perform multiplication by $\sqrt{2}$.

2.2. Multiplierless DCT Architecture. The DCT given by (1) can be expressed in inner product form as:

$$Y = \sum_{k=0}^{N-1} x(k) \cdot c(k), \quad (2)$$

where $c(k)$ for $0 \leq k \leq N-1$ are fixed coefficients and equal to $\cos((2k+1)\pi/2N)$, and $x(k)$ for $0 \leq k \leq N-1$ are the input image pixels.

One possible implementation of the inner product in programmable devices uses embedded multipliers. However, these IPs are not designed for constant multipliers. Consequently, they are not power efficient and consume a larger silicon area. Moreover, such a design is not portable for efficient implementation in FPGAs and ASICs. Many multiplierless architectures have, therefore, been introduced for efficient implementation of constant multiplications for the inner product computation. All those methods can be classified as: the ROM-based design [14], the distributed arithmetic (DA-) based design [15], the New distributed

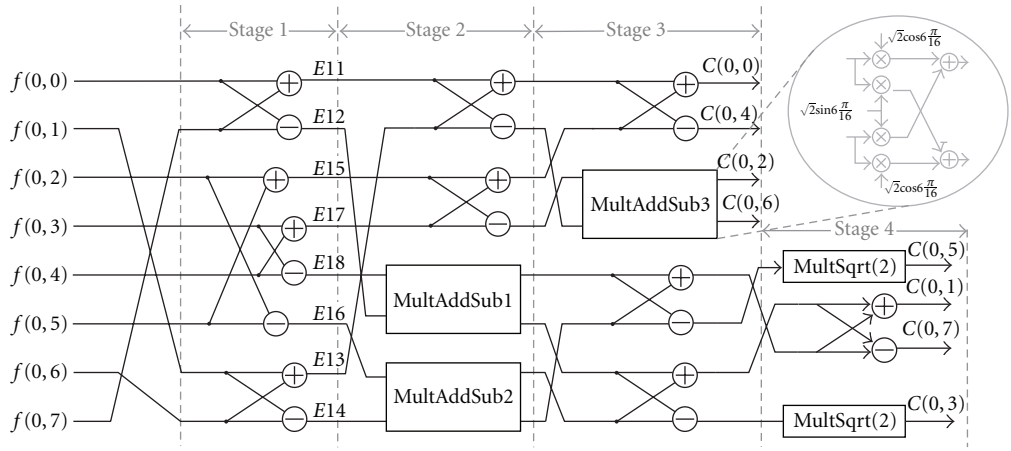

Figure 2: Loeffler architecture of 8-point DCT algorithm.

arithmetic (NEDA-) based design [22], and the CORDIC-based design [23].

2.2.1. ROM Multiplier-Based Implementation. This solution is presented in [14] to design a special-purpose VLSI processor of 8×8 2D DCT/IDCT chip that can be used for high-speed image and video coding. Since the DCT coefficient matrix is fixed, the authors of [14] precompute all possible product values and store them in a ROM rather than computing them by any combinational logic. Since the dynamic range of input pixels is 2^8 for gray scale images, the number of stored values in the ROM is equal to $N = 2^8$. Each value is encoded using 16 bits. For example, for an 8-point inner product, the ROM size is about $8 * 2^8 * 16$ bits which is equivalent to 32.768 kbits. To obtain 8-point DCT, 8-point inner products are required and consequently, the ROM size becomes exorbitant for realization of image compression.

2.2.2. Distributed Arithmetic (DA). Distributed arithmetic (DA) [15] is a well-known technique for computing inner products which outperforms the ROM-based design. Authors of [16–18] use the recursive DCT algorithm to derive a design that requires less area than conventional algorithms. By precomputing all the partial inner products corresponding to all possible bit vectors and storing these values in a ROM, the DA method speeds up the inner product computation over the multiplier-based method. Unfortunately, in this case also the size of ROM grows exponentially with the number of inputs and internal precision. This is inherent to the DA technique where a great amount of redundancy is introduced into the ROM to accommodate all possible combinations of bit patterns in the input signal.

2.2.3. New Distributed Arithmetic (NEDA). The New Distributed Arithmetic (NEDA) is adder-based optimization of DA implementation. The NEDA architecture does not require ROMs and multipliers. It provides reduced complexity solution by sharing of common subexpression of input vector to generate optimal shift-add network for DCT implementation. This results in a low-power, high-throughput architecture for the DCT. Nevertheless, the implementation of NEDA has two main disadvantages, [22]:

(i) the parallel data input leads to higher scanning rate which severely limits the operating frequency of the architecture;

(ii) the assumption of serial data input leads to lower hardware utilization.

2.2.4. CORDIC. COordinate Rotation DIgital Computer (CORDIC) provides a low-cost technique for DCT computation. The CORDIC-based DCT algorithm in [23] utilizes dynamic transformation rather than static ROM addressing. The CORDIC method can be employed in two different modes: the rotation mode and the vectoring mode. Sun et al. in [24] have presented an efficient Loeffler DCT architecture based on the CORDIC algorithm. However, the use of dynamic computation of cosine function in iterative way involves long latency, high-power consumption and involves a costly scale compensation circuit.

2.2.5. CSD. Vinod and lai in [25] have proposed an algorithm to reduce the number of operations by using CSD and CSE techniques, where they have minimized the number of switching events in order to reduce the power consumption. In CSD encoding, the constant multiplications are replaced by additions. Hence, there are two types of additions: interstructural adders to compute the summation terms of the inner product of (2) and intrastructural adder required to replace the constant multipliers. The authors of [25] applied the CSD encoding on the constant multiplier of the conventional DCT computation. Consequently, the number of intrastructural adders is reduced, but the number of interstructural adders is increased to 56 for 8-point DCT. However, as it is reported in Table 1, there are some fast DCT algorithms which use the cosine symmetry to reduce the number of adders (from 26 to 29). Moreover, the authors of [25] have used CSE technique to reduce the number of intrastructural adders. Indeed, since each data (image

Table 1: Complexity of different DCT algorithms.

Reference	[7]	[8]	[9]	[10]	[11]	[12]
Multipliers	16	12	12	12	12	11
Adders	26	29	29	29	29	29

pixel) is multiplied with distinct constant element (cosine coefficient), Vinod and lai have proposed to reformulate the DCT matrix for efficient substitution of CSE. With this optimization, the total number of intrastructural adders substantially reduced.

2.3. Joint Optimization. Recent research on DCT implementation uses the optimization to adapt the implementation of the DCT to the specific compression standards in order to reduce the chip size and power consumption. All these recent works in this area exploit the context in which the DCT is used to reduce the computational complexity. Some of them use the characteristics of input signals and the others simplify the DCT architecture. Xanthopoulos and Chandraksan in [28] have exploited the signal correlation property to design a DCT core with low-power dissipation. Yang and Wang have investigated the joint optimization of the Huffman tables, quantization, and DCT [31]. They have tried to find the performance limit of the JPEG encoder by proposing an iterative algorithm to find the optimal DCT bit width for a given Huffman tables and quantization step sizes.

A prediction algorithm is developed in [32] by Hsu and Cheng to reduce the computation complexity of the DCT and the quantization process of H264 standard. They have built a mathematical model based on the offset of the DCT coefficients to develop a prediction algorithm.

In this paper, we propose a model for combined optimization of DCT and quantization to implement them in the same architecture to save the computational complexity for image and video compression.

3. Proposed Algorithm for DCT Computation

In this Section, we present a new multiplierless DCT based on CSD encoding.

3.1. Principle of CSD. The CSD representation was first introduced by Avizienis in [33] as a signed-digit representation of numbers. This representation was created originally to eliminate the carry propagation chains in arithmetic operations. It is a unique signed-digit representation containing the fewest number of nonzero bits. It is therefore used for the implementation of constant multiplications with the minimum number of additions and subtractions. The CSD representation of any given number c is given by:

$$c = \sum_{i=0}^{N-1} c_i \cdot 2^i, \quad c_i = \{-1, 0, 1\}, \tag{3}$$

CSD numbers have two basic properties:

(i) no two consecutive digits in a CSD number are nonzero;

Table 2: 8-point DCT fixed coefficient representation.

Real value	Decimal	Natural binary	Partial products	CSD	Partial products
$\cos(3\pi/16)$	106	01101010	4	+0−0+0+0	4
$\sin(3\pi/16)$	71	01000111	4	0+00+00−	3
$\cos(\pi/16)$	126	01111110	6	+00000−0	2
$\sin(\pi/16)$	25	00011001	3	00+0−00+	3
$\cos(6\pi/16)$	49	00110001	3	0+0−000+	3
$\sin(6\pi/16)$	118	01110110	5	+000−0−0	3
$\sqrt{(2)}$	181	10110101	5	+0−0−0+0+	5
Total partial products			30		23

(ii) The CSD representation of a number contains the minimum possible number of nonzero bits and thus the name canonic.

The CSD values of the constants used in 8-point DCT by Loeffler algorithm are listed in Table 2. The digits 1,−1 are, respectively, represented by +,−. For 8 bit width, the saving in term of partial products is about 24%. A generalized statistical study about the average number of nonzero elements in N-bit CSD numbers is presented in [33], and it is proved that this number tends asymptotically to $N/3 + 1/9$. Hence, on average, CSD numbers contain about 33% fewer nonzero bits than 2's complement numbers. Consequently, for multiplications by a constant (where the bit pattern is fixed and known a priori), the numbers of partial products are reduced by nearly 33% in average.

3.2. New CSE Technique for DCT Implementation. To minimize the number of adders, subtractors, and shift operators for DCT computation, we can use the common subexpression elimination (CSE) technique over the CSD representation of constants. CSE was introduced in [35] and applied to digital filters in transpose form. Contrary to transpose form FIR filters, constant coefficients of DCT (shown in Table 2) multiply 8 *different input data* since the DCT consists in transforming 8-point input sequence to 8-point output coefficient. For this reason we cannot exploit the redundancy among the constants for subexpression elimination as in case of FIR filter. Moreover, for bit patterns in the same constant, Table 2 shows that only the constant $\sqrt{2}$ presents one common subexpression which is +0− repeated once with an opposite sign. Consequently, we cannot use the conventional CSE technique in the same manner as in the case of multiple constant multiplication in FIR filters.

We have proposed here a new CSE approach for DCT optimization where we do not consider occurrences in CSD coefficients, but we consider the interaction of these codes. On the other hand, according to our compression method (detailed in the next Section) we use only some of the DCT coefficients (1 to 5 among 8). Hence, it is necessary to compute specific outputs separately. To emphasize the advantage of CSE, we take the example of

$X(2)(X(2) = E35 + E37)$. According to Figure 2, we can express $E35$ as follows:

$$\begin{aligned} E35 &= (E25 + E28) \\ &= \left(E18 * \cos\left(\frac{3\pi}{16}\right) + E12 * \sin\left(\frac{3\pi}{16}\right)\right) \\ &\quad + \left(E14 * \cos\left(\frac{\pi}{16}\right) - E16 * \sin\left(\frac{\pi}{16}\right)\right). \end{aligned} \tag{4}$$

Using CSD encoding of Table 2, (7) is equivalent to:

$$\begin{aligned} E35 = {} & E18(2^7 - 2^5 + 2^3 + 2^1) + E12(2^6 + 2^3 - 2^0) \\ & - E16(2^5 - 2^3 + 2^0) + E14(2^7 - 2^1). \end{aligned} \tag{5}$$

After rearrangement (8) is equivalent to:

$$\begin{aligned} E35 = {} & 2^7(E18 + E14) + 2^6E12 - 2^5(E16 + E18) \\ & + 2^3(E12 + E16 + E18) + 2^1(E18 - E14) \\ & - 2^0(E12 + E16). \end{aligned} \tag{6}$$

In the same way, we can determine $E37$:

$$\begin{aligned} E37 = {} & 2^7(E12 + E16) - 2^6E18 + 2^5(E14 - E12) \\ & + 2^3(E12 - E14 - E18) + 2^1(E12 - E16) \\ & + 2^0(E14 + E18). \end{aligned} \tag{7}$$

Equations (10) and (11) give

$$\begin{aligned} X(2) = {} & 2^7\left(\overbrace{(E16 + E18)}^{\text{CS1}} + E12 + E14\right) + 2^6(E12 - E18) \\ & - 2^5\left(\overbrace{(E12 - E14)}^{\text{CS2}} + \overbrace{(E16 + E18)}^{\text{CS1}}\right) \\ & + 2^3\left(\overbrace{(E12 - E14)}^{\text{CS2}} + E12 + E16\right) \\ & + 2^1\left(\overbrace{(E18 - E16)}^{\text{CS3}} + \overbrace{(E12 - E14)}^{\text{CS2}}\right) \\ & + 2^0\left(\overbrace{(E14 - E12)}^{\text{CS2}} + \overbrace{(E18 - E16)}^{\text{CS3}}\right), \end{aligned} \tag{8}$$

where CS1, CS2, and CS3 denote 3 common subexpressions. In fact, the identification of common subexpressions results in significant reduction of hardware and power consumption reductions. For example, CS2 appears 4 times in $X(2)$. This subexpression is implemented only once and resources needed to compute CS2 are shared. An illustration of resources sharing is given in Figure 3.

Table 3: Statistics of X(2) calculation.

Components/methods	Multiplier based	CSD	CSD-CSE
Adders/subtractors	11	23	16
Registers	125	188	119
MULT18x18SIOs	4	0	0
Equivalent no of LUT	305	221	200
Latency	$3T_A + T_M$	$6T_A$	$4T_A$
Maximum frequency[1]	143.451	121.734	165.888

[1]Maximum frequency is measured in MHz.

Symbols $\ll n$ denote left shift operation by n-bit positions. It is important to notice that nonoverbraced terms in (12) are potential common subexpressions which could be shared with other DCT coefficients such as $X(4)$, $X(6)$, and $X(8)$. According to this analysis, $X(2)$ is computed by using 11 adders and 4 embedded multipliers. If CSD encoding, is applied 23 adders/subtractors, are required. The proposed method enables to compute $X(2)$ by using only 16 add/subtract operations. This improvement allows to save silicon area and reduces the power consumption without any decrease in the maximum operating frequency.

To emphasize the common subexpression sharing, a VHDL model of calculation of $X(2)$ is developed using three techniques: embedded multipliers, CSD encoding, and CSE of CSD encoding. It is shown in Table 3 that the CSD encoding uses more adders, subtractors, and registers than the proposed combined CSD-CSE technique to replace the 4 embedded multipliers MULT18x18SIOs. Also, we have included the equivalent number of LUT if $X(2)$ is synthesized on Xilinx FPGA without any arithmetic DSP core (without MULT18x18). Total number of LUTs is found to be 305, 221, and 200, respectively, for the multiplier-based design, the CSD-based design, and the combined CSD-CSE-based design. Moreover, it can be observed that the time required to get $X(2)$ coefficient is equal to $3T_A + T_M$, $6T_A$, and $4T_A$, respectively, for the multiplier-based design, the CSD-based design, and the combined CSD-CSE-based design, where T_A is the addition time and T_M is the multiplication time. Consequently, the area-delay product is decreased by sharing subexpression.

4. Joint Optimization

4.1. Principle. As discussed earlier, the 2D DCT is computed in two stages by row/column decomposition using row-wise 1D DCT of input in stage 1, followed by column-wise 1D DCT of intermediate result in stage 2. If we consider an input block $f(i, j)$ of 8×8 samples, the row-wise transform calculates $X(i, v)$ as 1D DCT of $f(i, j)$, and the column-wise transform gives $Y(u, v)$ which are the 1D DCT coefficients applied to $X(i, v)$ for $i, j, u, v \in [1 : 8]$. Hence, for a given 8×8 block of pixels, we obtain 64 DCT coefficients of different frequencies. Unlike the high-frequency coefficients, the low-frequency coefficients have a greater effect on image reconstruction. Moreover, after quantization process, most of the high-frequency coefficients are likely to be zero as

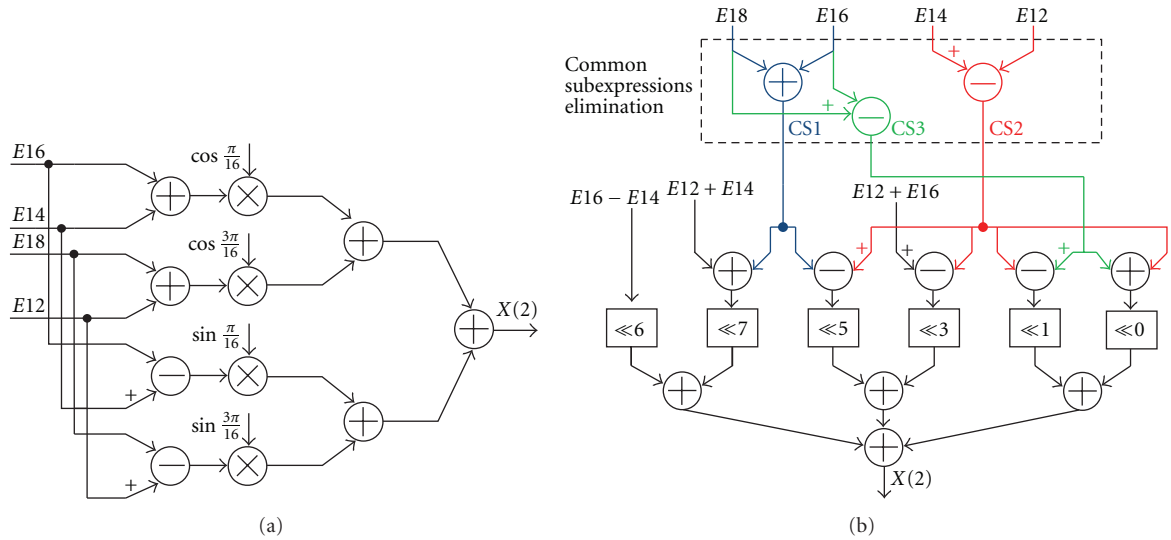

FIGURE 3: $X(2)$ calculation (a) conventional method, (b) shared subexpression using CSD encoding.

shown in Figure 4. Since these coefficients are likely to be zero after quantization, to save computation time and resources we avoid computing these DCT coefficients. In fact, for an 8×8 block of pixels, we compute 64 1D DCT coefficients of the first stage and then we compute only low frequency components for 1D DCT of second stage. With this method, the quantization is done on the fly with the DCT algorithm and consequently we save computational resources. Another advantage of the proposed method is the latency improvement. Since the high-frequency coefficients are to be eventually discarded, the time used for their computation is saved. In fact, for the second 1D DCT algorithm, at least 3 rows do not need to be computed as illustrated in the Figure 5. This gives a saving of at least 12 clock cycles, since the latency of calculation of each row is 4 clock cycles.

4.2. Quantization Levels. For an 8×8 block of pixels, the 1D DCT is calculated for each of the 8 input rows as mentioned in Figure 5. For each row, the first 1D DCT coefficient $X(i, 1)$ is encoded using 11 bits which is the estimated word length without truncation or rounding for $i \in [1 : 8]$. Coefficients $X(i, 2)$ to $X(i, 8)$ are truncated using 8 bits to trade accuracy for compression ratio and computational complexity.

In the second stage, 1D DCT is calculated selectively since the higher frequency components need not be computed. It is known that the lower frequencies tend to spread across either the first row or the first column of the 2D DCT coefficient matrix. However, the computation of an entire row and entire column leads to the computation of all DCT coefficients. For this purpose, we propose an efficient and simple computing scheme by creating 4 DCT zones (shown in Figure 5) where each zone corresponds to a specific compression ratio. For an 8×8 block of pixels, the row-wise transform is applied to compute 64 DCT coefficients while the column-wise transform is applied partially to reduce the computational complexity. Indeed, the proposed quantization zones are chosen to be square in order to avoid redundancy of computation. In Zone 1, only 4 coefficients $Y(1, 1)$, $Y(1, 2)$, $Y(2, 1)$, and $Y(2, 2)$ are calculated by 2 1D DCT operations. The first one is applied to the first column of intermediate result (output of the row-wise transform) which gives $Y(1, 1)$ and $Y(1, 2)$ while the second one is applied to the second column of intermediate result to compute $Y(2, 1)$ and $Y(2, 2)$. For this quantization mode, all the others DCT coefficients are set to zero. Similarly, in Zone 4, 25 DCT coefficients are calculated by 5 1D DCT operations to compute coefficients $Y(u, v)$, $u, v \in [1 : 5]$.

The compression ratio depends on the zone selection. In Zone 1, $Y(1, 1)$ is encoded using 14 bits. Indeed, the first 1D-DCT coefficient is encoded using 11 bits since in Loeffler DCT algorithm (shown in Figure 2), the DC output is obtained by three cascaded adder stages applied to 8-bit image pixels. Since the 11-bit DC output is fed to the second stage of 1D DCT the bit width of the first output of 2D-DCT is equal to 14 bits. This bit width is taken as reference for encoding the AC coefficients which have less influence than DC coefficient on the quality of reconstructed images. To estimate the bit width of AC coefficients, we were referred to image and video quantization tables (Q) in JPEG standard for Luminance image component and in MPEG-4 standard for intraframe video coding. It is found that for image quality of 50%, $Y(2, 2)$ is divided by $Q(2, 2) = 24$. Then, in order to have a unique bit width by quantization zone, AC coefficients of zone 1 are encoded with 5 bits under the DC bit width (i.e., 9 bits). Likewise, AC coefficients of zone 2 need coefficients of zone 1 along with 5 other coefficients $Y(1, 3)$, $Y(2, 3)$, $Y(3, 1)$, $Y(3, 2)$, and $Y(3, 3)$. All these data are encoded using 8 bits. The additional coefficients of zone 3 are encoded using 7 bits. Finally, the remaining coefficients of zone 4 are encoded using 6 bits since $Y(5, 5)$ will be divided by $Q(5, 5) = 136$.

Hence, by selecting zone 1, the total number of bits is equal to $14 + 9 * 3 = 41$ bits and the compression ratio (CR)

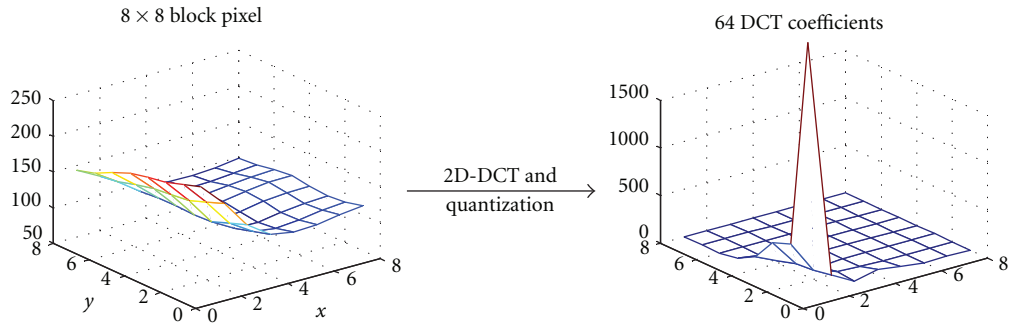

Figure 4: Principle of the DCT and the quantization.

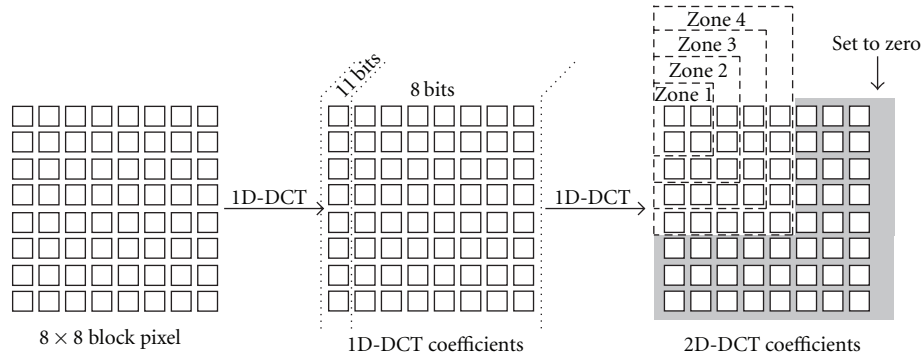

Figure 5: Quantization zones.

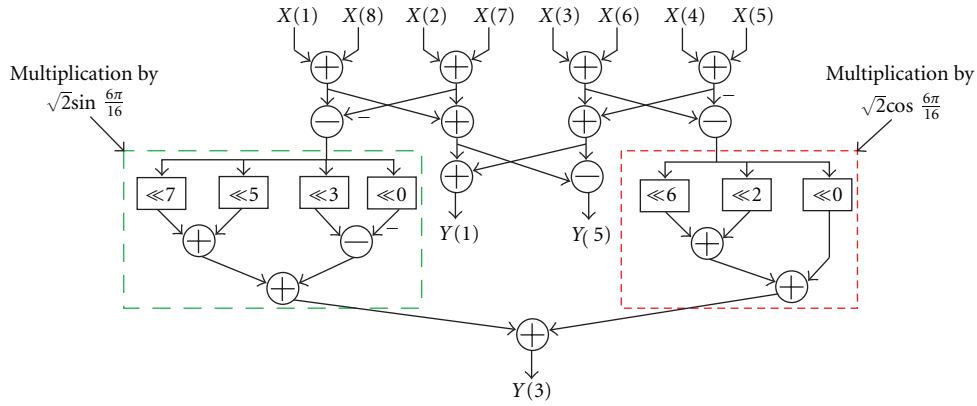

Figure 6: $Y(1,v)$, $Y(3,v)$, and $Y(5,v)$ calculation with CSE and CSD.

which is the ratio of the original image size to the compressed image size is equal to 8 bits $*$ 64/14 bits = 12.48. For the zone 2, the compression ratio is equal to 8 bits $*$ 64/(14 + 9 $*$ 3 + 8 $*$ 5 bits) = 6.32. Similarly, for zone 3 and 4, the CRs are, respectively, equal to 3.95 and 2.78.

4.3. DCT Calculation. An example of the calculation of the 1D DCT coefficient ($X(i,2)$ for $i \in [1:8]$) is given in Section 3.2. To compute the DCT coefficients of the 1D DCT of second stage, we use the same method by replacing the inputs by $X(i,v)$ and the outputs by $Y(u,v)$ for $u,v \in [1:8]$. According to the algorithm illustrated in Figure 2, for a given column, $Y(1,v)$ and $Y(5,v)$ are calculated using adders and subtractors while $Y(3,v)$ uses a multiplicative constant and is given by:

$$Y(3) = \sqrt{2}\cos\left(\frac{6\pi}{16}\right)E24 + \sqrt{2}\sin\left(\frac{6\pi}{16}\right)E22. \quad (9)$$

Intermediate results $E22$ and $E24$ in (9) are shown in Figure 2 for a given column. Now, constants $\sqrt{2}\cos(6\pi/16)$ and $\sqrt{2}\sin(6\pi/16)$ are converted to CSD format and given, respectively, by 0 + 000 + 0+ and 0 + 0 + 0 + 00−. $Y(1,v)$, $Y(3,v)$ and $Y(5,v)$ calculations are given in Figure 6.

For $Y(2,v)$ and $Y(4,v)$ also the CSD-CSE techniques are used. $Y(2,v)$ is calculated as in (8), and the common subexpression of $Y(4,v)$ calculation is determined by increasing

the number of common subexpressions shared between $Y(2,v)$ and $Y(4,v)$.

According to Figure 2, $Y(4,v) = \sqrt{2}(E26 - E27)$ which is equivalent to:

$$\begin{aligned} Y(4,v) = {} & \sqrt{2}\left(E12 * \cos\left(\frac{3\pi}{16}\right) - E18 * \sin\left(\frac{3\pi}{16}\right)\right) \\ & - \sqrt{2}\left(E16 * \cos\left(\frac{\pi}{16}\right) - E14 * \sin\left(\frac{\pi}{16}\right)\right). \end{aligned} \tag{10}$$

Using CSD encoding of the constant coefficient, (10) is equivalent to:

$$\begin{aligned} Y(4,v) = {} & E12(2^7 + 2^5 - 2^3 - 2^0) \\ & - E18(2^7 - 2^5 + 2^2 + 2^0) \\ & - E16(2^8 - 2^6 - 2^4 + 2^1) \\ & + E14(2^5 + 2^2 - 2^0). \end{aligned} \tag{11}$$

After rearrangement (11) is equivalent to:

$$\begin{aligned} Y(4,v) = {} & 2^7\left(-\overbrace{(E16+E18)}^{\text{CS1}} + \overbrace{(E12-E16)}^{\text{CS4}}\right) \\ & + 2^5\left(-\overbrace{(E16+E18)}^{\text{CS1}} + \overbrace{(E12+E14)}^{\text{CS5}} + E14\right) \\ & - 2^3\left(\overbrace{(E12-E16)}^{\text{CS4}}\right) \\ & + 2^2\left(E16 + E14 - \overbrace{(E18-E16)}^{\text{CS3}}\right) \\ & - 2^0\left(\overbrace{(E16+E18)}^{\text{CS1}} + \overbrace{(E12+E14)}^{\text{CS5}} + E16\right), \end{aligned} \tag{12}$$

For $Y(4,v)$ calculation, the common subexpression CS1 and CS3 defined for $Y(2,v)$ calculation is used. Two new subexpressions CS4 and CS5 are introduced to further reduce the arithmetic operators. It is important to mention that the equations listed before are expressed to create several occurrences of common subexpression such as CS1, CS3, and CS4 those are used for $Y(2,v)$ calculation. The Signal flow graphs of $Y(2,v)$ and $Y(4,v)$ are shown in Figure 7.

5. Simulation Results

We have coded the proposed method and the existing competing algorithms in VHDL and synthesized them using Xilinx ISE tool.

Table 4: Macrostatistics of 1D DCT calculation.

Method	DA [16]	DA [18]	NEDA [19]	CSD [25]	Proposed
Adders	136	144	85	123	72

Table 5: Microstatistics of 1D DCT calculation.

Method	NEDA [19]	CORDIC	[37]	Xilinx's core	Proposed
Slices	1031	780	531	369[2]	454

[2] Apart from the number 369 slices Xilinx core uses 4 embedded multipliers.

5.1. Synthesis Results. From high-level synthesis results we obtain the number of adders used for different DA-based 1D DCT design and listed in Table 4. It is found that our design uses fewer adders than the other. The direct realization of DA-based DCT design requires 308 adders. Optimizations presented in [19] reduce the number of adders to 85. Regarding the CSD-based design [25], for 8-bit constant width, we found that design of [25] consumes 123 adders (67 intrastrucutral adders + 56 interstructural adders) while the proposed design involves the DCT with 72 adders. We have listed the number of slices occupied by the 1D DCT of [37] and proposed design in the Table 5. The proposed method is compared favorably with the conventional multiplierless architectures using Xilinx XC2VP50 FPGA, the same device employed in [37].

Note that the Xilinx's core uses the Chen's algorithm [7] and requires 369 Slices along with 4 embedded multipliers 18x18SIOs. Besides, the number of slices required by the Xilinx core is relatively low compared with other designs because, the adder/subtractor module of the Xilinx's design alternatively chooses addition and subtraction by using a toggle flop. However, the slice-delay product of proposed design is significantly less than that of the Xilinx DCT IP core since the later has a maximum usable frequency (MUF) of 101 MHz on Spartan3E device while the proposed design provides MUF of 119 MHz.

Regarding the timing analysis derived from synthesis results obtained by cadence 0.18 μ library, we find that the proposed design involves a delay about 14.4 ns which represents nearly 15% less than the Xilinx core and 60% less than optimized NEDA-based design [20]. We should underline that in [20] an optimized architecture of NEDA-based design [19] where compressor trees are used to decrease the delay.

Moreover, we have used the XPower tool of Xilinx ISE suite to estimate the dynamic power consumption. The power dissipation of the proposed 1D DCT design and Xilinx's core is about 39 mW and 62 mW respectively.

In order to highlight the effect of subexpression sharing, 1D DCT structure of Loeffler algorithm is implemented with different multiplier designs. The power-delay product in nJ is computed as the product of the DCT computation time (ns) and the power dissipation (W) for the proposed design and the Xilinx core. The power-delay product for different number of DCT coefficients is calculated and plotted in Figure 8. The CSD-based design and the proposed design using CSE involve nearly 43% and 33% of power-delay

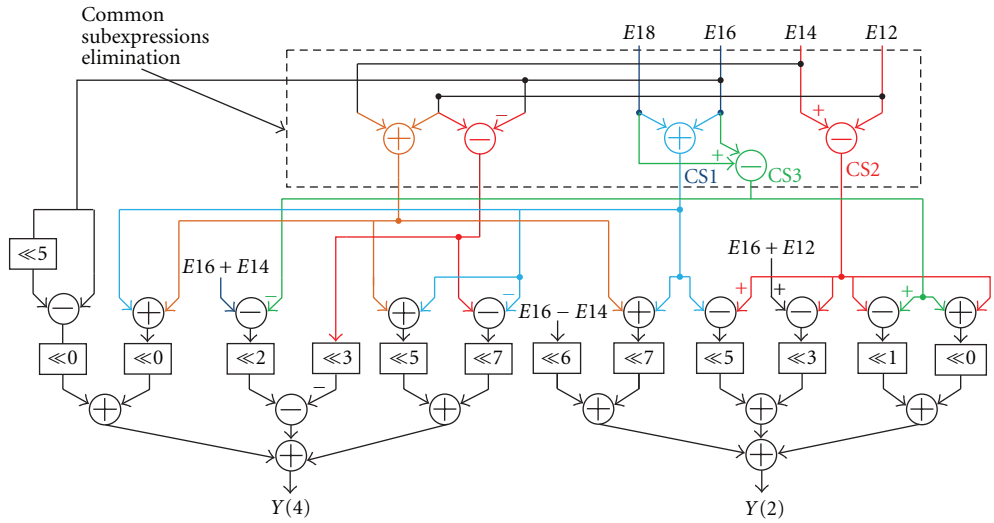

Figure 7: $Y(2, v)$ and $Y(4, v)$ calculation with CSE and CSD.

product of the Xilinx's multiplier-based design, respectively. It can be seen in Figure 8 that the computation of only the first 1D DCT coefficient involves the same power-delay product since this coefficient does not require any multiplier. Note that the computation of 4th DCT coefficient requires nearly the same power-delay product as that for 5th DCT coefficient. Indeed, the computation of the fifth DCT coefficient requires only one more subtrator.

For the 2D DCT architecture using the Loeffler algorithm a performance analysis is presented in Table 6 in order to highlight the effects of CSD coding, subexpression sharing, and quantization. The multiplier-based structures considered in the comparison are the Xilinx's embedded multiplier synthesized as multiplier block IP and in LUTs. It should be indicated that the input bit width is of 8 bits, the DC coefficient bit width of the first and second 1D DCT stages are 11 bits and 14 bits, respectively and the constant cosine coefficient bit width is 8 bits. The implementation of 2D DCT is realized by decomposing the 2D DCT into two 1D DCT computations together with a transpose memory. It can be observed in Table 6 that the area-delay complexity of Xinlinx's multiplier-based 2D DCT design (synthesized in block) is nearly the same as that of the combined CSD-CSE design but has nearly twice the power consumption. On the other hand, when the Xilinx's multipliers are synthesized as LUT, the 2D DCT structure has less power-delay product but involves twice the area compared with the combined CSD-CSE structure.

The average computation time (ACT) is the time interval after which we get a set of 2D DCT coefficients. ACT is the product of the number of clock cycles required for the 2D DCT computation and the duration of a clock cycle. The 2D DCT computation requires 86 cycles, which is comprised of 8 cycles for register inputs, 7 cycles for the first stage 1D DCT, 64 cycles for transpose memory, and 7 cycles for the second stage of 1D DCT.

Finally, we use the energy per output coefficient (EoC) as power metric which amounts to the average of energy

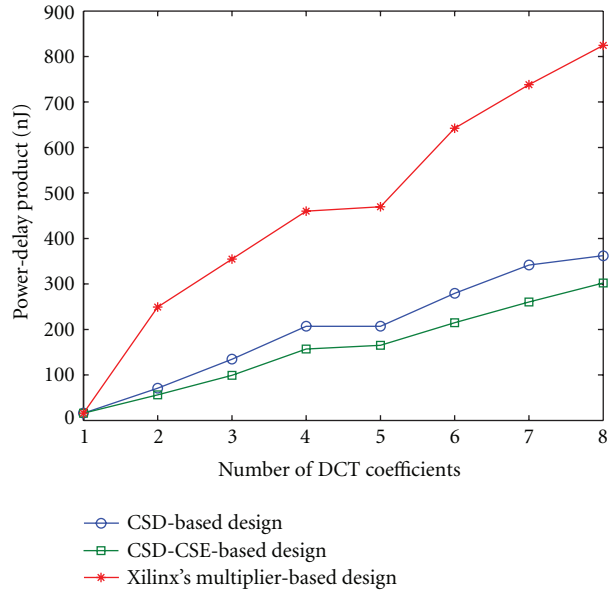

Figure 8: Power-delay product estimation with 1.6 V design.

required to compute one value of 2D DCT output. EoC is calculated by multiplying the ACT by the power consumption and dividing the product by 64. It is shown in Table 6 that the design based on Xilinx's multiplier IP involves more than twice EoC compared with all the other designs. Moreover, the proposed 2D DCT with CSD-CSE technique needs less energy compared with CSD-based design and Xilinx's multiplier-based design. It can be further observed that the proposed CSD-CSE and quantization technique has 46% to 65% of EoC compared to CSD-based design.

5.2. FPGA Implementation of Image Compression. In this subsection, we examine the quality of reconstructed image using an FPGA prototype of the proposed DCT-based image

Table 6: Performance analysis of the 2D DCT design using Loeffler algorithm.

Constant multiplication design	Slice	MULT18x18SIOs	Power dissipation (mW)	Delay (ns)	ACT (μs)	EoC (nJ)
Xilinx-embedded multipliers (Block)	960	20	176	20.58	1.769	311.344
Xilinx-embedded multipliers (LUT)	1836	0	66.4	24.02	2.065	137.116
CSD	1423	0	74	22.39	1.925	142.45
CSD-CSE	1048	0	76	19.94	1.714	130.264
CSD-CSE and quantization	[625, 765]	0	[48, 64]	[15.90, 16.89]	[1.367, 1.452]	[65.616, 92.928]

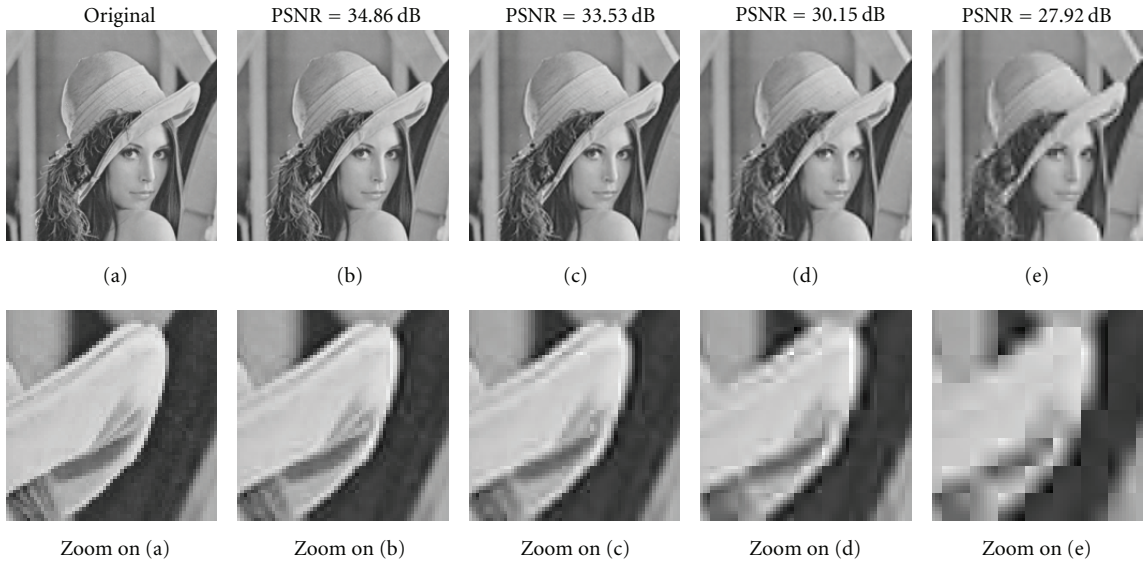

Figure 9: Decoded images after joint optimization. (a) (Original), (b) (quantization zone = 4, bpp = 2.87, PSNR = 33.24 dB), (c) (quantization zone = 3, bpp = 2.0, PSNR = 30.26 dB), (d) (quantization zone = 2, bpp =1.26, PSNR = 28.23 dB), and (e) (quantization zone = 1, bpp=0.64, PSNR = 25.38 dB).

Table 7: PSNR (dB) versus bpp evaluation.

bpp	0.64	1.26	2.0	2.87
Lena	25.38	28.23	30.26	33.24
Mandrill	20.84	25.27	28.20	29.36
Peppers	27.92	30.15	33.53	34.86
Goldhill	27.51	29.22	32.43	33.12

compression unit. The test images are saved in a ROM in order to avoid the transmission time between the PC and the FPGA. It is important to mention that in the final design we need to use a 2-bit word to indicate 4 available compression ratios. To measure the visual quality of the reconstructed image and to validate the proposed DCT design, we use the Xilinx's integrated logic analyzer (ILA). This module works as a digital oscilloscope and enables to trigger on signals in the hardware design.

To reconstruct back the images, a floating point inverse 2D DCT function of Matlab tool is applied to the FPGA output. PSNR of different 255 $\times$ 255 gray scale images are evaluated and listed in Table 7. The bit per pixel (bpp) depends on the quantization zone selection and varies from 0.64 to 2.87 (The compression is due to DCT only. To increase the compression ratio further the quantized DCT output needs to pass through the entropy coding which we have not performed here.). As shown in Table 7, a good or acceptable image visual qualities can be obtained by joint optimization of the quantization and the DCT. Moreover, to underline the adequacy between PSNR results and the user perception, in Figure 9 we have shown the decoded images for different selection of quantization zones. It is found that the higher the PSNR of reconstructed image, the better the quality is.

6. Conclusion

In this paper, we have presented a low-complexity DCT-based image compression. We presented a novel common subexpression sharing of intermediate signals of the DCT computation based on CSD representation of Loeffler's 8-point DCT algorithm. Finally, we have combined the quantization process with the second stage of DCT computation

in order to optimize the bit width of computation of DCT coefficient according to the quantization of different zones.

We would like to point out that a prior detection of zero-quantized coefficients along with the proposed techniques could be used to further reduce the complexity of DCT computations.

References

[1] A. Alkholidi, A. Alfalou, and H. Hamam, "A new approach for optical colored image compression using the JPEG standards," *Signal Processing*, vol. 87, no. 4, pp. 569–583, 2007.

[2] M. Jridi and A. Alfalou, "Joint Optimization of Low-power DCT Architecture and Effcient Quantization Technique for Embedded Image Compression," in *VLSI-SoC: Forward-Looking Trends in IC and System Design*, J. Ayala, A. Alonso, and R. Reis, Eds., pp. 155–181, Springer, Berlin, Germany, 2012.

[3] M. Jridi and A. Alfalou, "A low-power, high-speed DCT architecture for image compression: Principle and implementation," in *18th IEEE/IFIP International Conference on VLSI and System-on-Chip (VLSI-SoC '10)*, pp. 304–309, September 2010.

[4] M. Jridi and A. AlFalou, "A VLSI implementation of a new simultaneous images compression and encryption method," in *IEEE International Conference on Imaging Systems and Techniques (IST '10)*, pp. 75–79, July 2010.

[5] "Video coding for low bit rate communication," (ITU-T Rec. H.263), February 1998.

[6] ISO/IEC DIS 10 918-1, "Coding of audio visual objects: part 2. visual," ISO/IEC 14496-2 (MPEG-4 Part2), January 1999.

[7] W. H. Chen, C. H. Smith, and S. C. Fralick, "A fast computational algorithm for the discrete cosine transform," *IEEE Transactions on Communications*, vol. 25, no. 9, pp. 1004–1009, 1977.

[8] B. G. Lee, "A new algorithm to compute the discrete cosine transform," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 32, no. 6, pp. 1243–1245, 1984.

[9] M. Vetterli and H. J. Nussbaumer, "Simple FFT and DCT algorithms with reduced number of operations," *Signal Processing*, vol. 6, no. 4, pp. 267–278, 1984.

[10] N. Suehiro and M. Hatori, "Fast algorithms for DFT and other sinusoidal tranforms," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 34, pp. 642–664, 1986.

[11] H. Hou, "A fast recursive algorithm for computing the discrete cosine transform," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 35, pp. 1455–1461, 1987.

[12] C. Loeffler, A. Lightenberg, and G. S. Moschytz, "Practical fast 1-D DCT algorithm with 11 multiplications," in *International Conference on Acoustics, Speech, and Signal Processing (ICASSP '89)*, pp. 988–991, May 1989.

[13] P. Duhamel and H. H'mida, "New 2n DCT algorithm suitable for VLSI implementation," in *International Conference on Acoustics, Speech, and Signal Processing (ICASSP '87)*, pp. 1805–1808, November 1987.

[14] D. Slawecki and W. Li, "DCT/IDCT processor design for high data rate image coding," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 2, no. 2, pp. 135–146, 1992.

[15] S. A. White, "Applications of distributed arithmetic to digital signal processing: a tutorial review," *IEEE ASSP Magazine*, vol. 6, no. 3, pp. 4–19, 1989.

[16] A. Madisetti and A. N. Willson, "100 MHz 2-D 8 × 8 DCT/IDCT processor for HDTV applications," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 5, no. 2, pp. 158–165, 1995.

[17] S. Yu and E. E. Swartzlander, "DCT implementation with distributed arithmetic," *IEEE Transactions on Computers*, vol. 50, no. 9, pp. 985–991, 2001.

[18] D. W. Kim, T. W. Kwon, J. M. Seo et al., "A compatible DCT/IDCT architecture using hardwired distributed arithmatic," in *IEEE International Symposium on Circuits and Systems (ISCAS '01)*, pp. 457–460, May 2001.

[19] A. Shams, W. Pan, A. Chidanandan, and M. Bayoumi, "A low power high performance distributed DCT architecture," in *IEEE Computer Society Annual Symposium on VLSI (ISVLSI '02)*, pp. 21–27, 2002.

[20] A. Chidanandan, J. Moder, and M. Bayoumi, "Implementation of NEDA-based DCT architecture using even-odd decomposition of the 8 × 8 DCT matrix," in *49th Midwest Symposium on Circuits and Systems (MWSCAS '06)*, pp. 600–603, August 2007.

[21] P. K. Meher, "Unified systolic-like architecture for DCT and DST using distributed arithmetic," *IEEE Transactions on Circuits and Systems I*, vol. 53, no. 12, pp. 2656–2663, 2006.

[22] M. Alam, W. Badawy, and G. Jullien, "A new time distributed DCT architecture for MPEG-4 hardware reference model," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 15, no. 5, pp. 726–730, 2005.

[23] S. Yu and E. E. Swartzlander, "A scaled DCT architecture with the CORDIC algorithm," *IEEE Transactions on Signal Processing*, vol. 50, no. 1, pp. 160–167, 2002.

[24] C. C. Sun, S. J. Ruan, B. Heyne, and J. Goetze, "Low-power and high-quality Cordic-based Loeffler DCT for signal processing," *IET Circuits, Devices and Systems*, vol. 1, no. 6, pp. 453–461, 2007.

[25] A. P. Vinod and E. M. K. Lai, "Hardware efficient DCT implementation for portable multimedia terminals using subexpression sharing," in *IEEE Region 10 Annual International Conference (TENCON '04)*, pp. A227–A230, November 2004.

[26] C. Cheng and K. K. Parhi, "A novel systolic array structure for DCT," *IEEE Transactions on Circuits and Systems II*, vol. 52, no. 7, pp. 366–369, 2005.

[27] P. K. Meher, "Systolic designs for DCT using a low-complexity concurrent convolutional formulation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 16, no. 9, pp. 1041–1050, 2006.

[28] T. Xanthopoulos and A. P. Chandrakasan, "A low-power dct core using adaptive bitwidth and arithmetic activity exploiting signal correlations and quantization," *IEEE Journal of Solid-State Circuits*, vol. 35, no. 5, pp. 740–750, 2000.

[29] J. Huang and J. Lee, "A self-reconfigurable platform for scalable dct computation using compressed partial bitstreams and blockram prefetching," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 19, no. 11, pp. 1623–1632, 2009.

[30] J. Huang and J. Lee, "Efficient VLSI architecture for video transcoding," *IEEE Transactions on Consumer Electronics*, vol. 55, no. 3, pp. 1462–1470, 2009.

[31] E. H. Yang and L. Wang, "Joint optimization of run-length coding, Huffman coding, and quantization table with complete baseline JPEG decoder compatibility," *IEEE Transactions on Image Processing*, vol. 18, no. 1, pp. 63–74, 2009.

[32] C. L. Hsu and C. H. Cheng, "Reduction of discrete cosine transform/quantisation/inverse quantisation/inverse discrete cosine transform computational complexity in H.264 video encoding by using an efficient prediction algorithm," *IET Image Processing*, vol. 3, no. 4, pp. 177–187, 2009.

[33] A. Avizienis, "Signed-digit number representations for fast parallel arithmetic," *IRE Transaction on Electronic Computers*, vol. 10, pp. 389–400, 1961.

[34] R. H. Seegal, "The canonical signed digit code structure for FIR filters," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 28, no. 5, pp. 590–592, 1980.

[35] R. T. Hartley, "Subexpression sharing in filters using canonic signed digit multipliers," *IEEE Transactions on Circuits and Systems II*, vol. 43, no. 10, pp. 677–688, 1996.

[36] C. Y. Pai, W. E. Lynch, and A. J. Al-Khalili, "Low-power data-dependent 8 × 8 DCT/IDCT for video compression," *IEE Proceedings: Vision, Image and Signal Processing*, vol. 150, no. 4, pp. 245–255, 2003.

[37] B. I. Kim and S. G. Ziavras, "Low-power multiplierless DCT for image/video coders," in *13th International Symposium on Consumer Electronics (ISCE '09)*, pp. 133–136, May 2009.

Hindawi Publishing Corporation
VLSI Design
Volume 2012, Article ID 413747, 14 pages
doi:10.1155/2012/413747

Research Article

An Efficient Multi-Core SIMD Implementation for H.264/AVC Encoder

M. Bariani, P. Lambruschini, and M. Raggio

Department of Biophysical and Electronic Engineering, University of Genova, Via Opera Pia 11 A, 16145 Genova, Italy

Correspondence should be addressed to P. Lambruschini, lambruschini@dibe.unige.it

Received 18 November 2011; Revised 20 February 2012; Accepted 3 March 2012

Academic Editor: Muhammad Shafique

The optimization process of a H.264/AVC encoder on three different architectures is presented. The architectures are multi- and singlecore and SIMD instruction sets have different vector registers size. The need of code optimization is fundamental when addressing HD resolutions with real-time constraints. The encoder is subdivided in functional modules in order to better understand where the optimization is a key factor and to evaluate in details the performance improvement. Common issues in both partitioning a video encoder into parallel architectures and SIMD optimization are described, and author solutions are presented for all the architectures. Besides showing efficient video encoder implementations, one of the main purposes of this paper is to discuss how the characteristics of different architectures and different set of SIMD instructions can impact on the target application performance. Results about the achieved speedup are provided in order to compare the different implementations and evaluate the more suitable solutions for present and next generation video-coding algorithms.

1. Introduction

In the last years the video compression algorithms have played an important role in the enjoying of multimedia contents. The passage from analog to digital world in multimedia environment cannot be performed without compression algorithms. DVDs, Blu-Ray, and Digital TV are typical examples. The compression algorithm used in DVDs is MPEG-2, and Blu-Ray supports VC-1 standardized with the name SMPTE 421M [1], in addition to MPEG-2 and H.264. In the digital television, the compression algorithms are used to reduce the transmission throughput. In DVB-T, the picture format for DVD and Standard Definition TV (SDTV) is 720×576, and this resolution is the most used in digital multimedia contents. The most recent standards for digital television as DVB-T2 and DVB-H support H.264/MPEG-4 AVC for coding video.

The H.264/AVC [2] video compression standard can cope with a large range of applications, reaching compression rate and video quality levels never accomplished by previous algorithms. Even if the initial H.264/AVC standard (completed in May 2003) was primarily focused on "entertainment-quality" video, not dealing with the highest video resolutions, the introduction of a new set of extensions in July 2004 covered this lack. These extensions are known as "fidelity range extensions" (FREext) and produced a set of new profiles, collectively called High Profiles. As described in [3], these profiles support all the Main Profile features and introduce additional characteristics such as adaptive transform block-size and perceptual quantization scaling matrices. Experimental results show that, when restricted to intra-only coding, H.264/AVC High Profile outperforms the state-of-the-art in still-image coding represented by JPEG2000 on a set of monochrome test images by 0.5 dB average PSNR [4].

It results that a H.264 encoder addressing high definition (HD) resolutions needs to support High Profiles in order to be part of an effective video application. On the other hand, the already great complexity of the H.264 algorithm is further increased by supporting FREext. In particular, this leads to implement two new modules: the 8×8 intraprediction and the 8×8 transform.

In case of mobile devices, the H.264 complexity issues together with the constraints of limited power consumption and the typical need of real-time operations in video-based applications draw a difficult scenario for video application developers.

The HD resolution involves a large amount of data, and the compression algorithms are high computational demand applications, often used as benchmark to measure the processor performance. In order to support real-time video encoding and decoding, specific architectures are developed. Multicore architectures have the potential to meet the performance levels required by the real-time coding of HD video resolutions. But in order to exploit multicore architectures, several problems have to be faced. The first issue is the subdivision of an encoder application in modules that can be executed in parallel. In this case, the main difficult is the strong data dependency in video encoder algorithms. Parallel architectures can be more easily exploited using other kind of algorithm like computer graphics, rendering technology or cryptography, where the data dependency is not as strong as in video compression. Once a good partitioning is achieved, the optimization of a video encoder should take advantage of the data level parallelism to increase the performance of each encoder module running on the architecture's processing element. A common approach is to use the SIMD instructions to exploit the data level parallelism during the execution; otherwise, ASIC design can be adopted for critical kernel. SIMD architectures are widely used for their flexibility. SIMD ISAs are added at most market spread processor: Intel's MMX, SSE1, SSE2, SSE3, SSE4; Amd 3DNow!; ARM's NEON; Motorola's AltiVec (also known as Apple's Velocity Engine or IBM's VMX).

In this paper, we will show how the data level parallelism is exploited by SIMD and which instructions are more useful in video processing. Different instruction set architectures (ISAs) will be compared in order to show how the optimization can be driven and how different ISA features can lead to different performance. This paper is intended to be a great help to both software programmers that have to choose for the most suitable SIMD ISA for developing a video-based application and for ISA designers that want to create a generic instruction set being able to give good performance on video applications. In that regard, the authors will select a set of generic SIMD instructions that can speed up video codec applications, detailing the modules that will profit from the introduction of each instruction. Besides describing the optimization methods, the paper indicates a few guidelines that should be followed to partition the encoder in separate modules.

Even though the work focuses on H.264/AVC, most of the proposed solutions will also apply to the earlier mentioned standards as well as to more recent video compression algorithms as scalable video coding (SVC) [5]. Moreover, H.264/AVC tools will have a fundamental role in the emerging high efficiency video coding (HEVC) standardization project [6].

This paper is organized as follows. Section 2 gives an overview of the state-of-the-art SIMD-based architectures, giving particular attention to those targeting video-coding applications. A brief description of the three architectures used for the presented project is given in Section 3. Section 4 describes the H.264 optimized encoder, focusing on module partitioning and SIMD-based implementation. The performance results of both the C-pure implementation and the SIMD version are given in Section 5 together with an explanation about what are the key instructions for optimizing a video codec. Finally, the conclusion is drawn in Section 6.

2. Related Works

The basic concept of SIMD instructions is the possibility of fill vector registers with multiple data in order to execute the same operation on several elements. One of the major bottlenecks in the SIMD approach is the overhead due to the data handling needed to feed the vector registers. Typical required operations are extra memory accesses, packing data, element permutation inside vectors, and conversion from vector to scalar results. All these preliminary operations limit the vector dimension and the performance enhancement achievable with SIMD optimization.

In literature, several studies regarding the SIMD optimization of video-coding applications are available [7–11]. The scope driving these studies is the achievement of the maximum performance, adopting measures in order to reduce the known bottlenecks. Since its standardization, SIMD optimizations targeting the H.264 algorithm have been proposed as can be seen in [12, 13]. However, both the works only address the H.264 decoder and present a MMX optimization starting from the H.264 reference code. Besides addressing a more complex application, our aims were also to discuss how the characteristics of different architectures and different set of SIMD instructions can impact on the encoder performance.

In SIMD processors the memory access has an important impact on performance. The unaligned access is not usually possible in SIMD ISA, and when possible it is discouraged due to additional instruction latency. The programmers usually take care of handling the unaligned load adding further overhead to vector data organization. Moreover, the need of unaligned load is always present in video-coding algorithm especially in motion estimation (ME) and motion compensation (MC), where the pixel blocks selected by motion vectors are frequently at misaligned positions even if the start of a frame is memory aligned. Often, the position of a block we need to access cannot be known in advance, and this leads to unpredictable misalignment in data loaded from memory. In Intel's architectures, starting from SSE2 the support to unaligned load has been added, but the performance is strongly reduced either if the load operation crosses the cache boundary or, with SSE3, if the load instruction needs store-to-load forwarding. In AltiVec, it is necessary to load two adjacent positions and shift data in order to achieve one unaligned load, a usually adopted approach to overcome the misalignment access issue. This problem is common in digital signal processor (DSP) as well. Usually, DSP do not support unaligned loads, but due to the large use of DSP in video application several producers have added the support to this kind of operation. For example, Texas Instruments family TMS320C64x supports unaligned load and store operations of 32 and 64 bit element, but with only one of the two memory ports [14].

The MediaBreeze SIMD processor was proposed to reduce the bottlenecks in SIMD implementations [15]. The Breeze SIMD ISA uses a multidimensional vector able to speed up nested loops but at the cost of a very complicated instruction structure requiring a dedicated instruction memory. In [16], a specific SIMD ISA named VS-ISA was proposed in order to improve performance in video coding. The authors adopted specific solutions for sum of absolute difference (SAD), not aligned load applied to ME, interpolation, DCT-IDCT, and quantization dequantization.

Another typical approach to reduce the SIMD overhead is the usage of multibank vector memory where data is stored interleaved. The drawback is the increase of hardware cost for supporting the addresses generation.

An alternative to SIMD implementation on programmable processor architectures is the hardwired processor. Usually, it is only used when performance and low power consumption are essential requirements [7, 14, 17]. In fact, the lack of flexibility typical of hardwired processors reduces their applicability to a narrow segment of the market, where the programmability is either not required or considerably reduced.

3. SIMD ISA Description

In order to optimize the H.264 encoder, we chose three different ISAs. The adopted architectures are ST240, xSTream, and P2012, all developed by STMicroelectronics. The former is a single-processor architecture, and the others are multicore platforms. In the following, the three architectures will be briefly described, giving special attention to the SIMD instruction set.

We chose these architectures for their novelty and for the possibility to have a complete toolchain (code generation, simulation, profiling, etc.) for developing an application in an optimal way. Each toolchain allowed a complete observability of the system. In this way, it was possible to evaluate the effectiveness of every author's solution. Observability is a very important characteristic when developing/optimizing an application. Using a real system it is not always possible to reach the degree of observability you have using a simulator and a suitable toolchain. Moreover, in an architecture under development as P2012 we had the possibility to contribute to the SIMD instruction set and, more important, to evaluate the contribution of each particular SIMD to the performance of the target video codec application. The three instruction sets present suitable characteristics for our research; they are generic instruction set, but ST240 includes a few video-specific instructions; we can analyse the impact of different vector register sizes; even if xSTream and P2012 share many characteristics, only xSTream supports horizontal SIMD (this is a special feature; e.g., other SIMD extensions as Intel SSE and ARM NEON do not have the same support); in P2012 platform, we were able to define and insert new SIMD instructions.

Besides the type of instructions, the SIMD extensions differ in both size and precision. These differences allow analyzing the impact of different architecture solutions on the global performance.

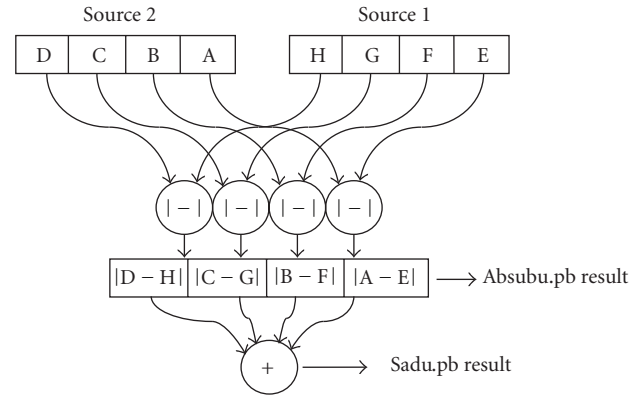

Figure 1: SAD operation.

3.1. ST240. The ST240 is a processor of STMicroelectronics ST200 family based on LX technology jointly developed with Hewlett Packard [18, 19]. The main ST240's features are the following:

(i) 4-issue Very Long Instruction Word (VLIW)

(ii) 64-32-bit general purpose registers

(iii) 32KB D-Cache and 32KB I-Cache

(iv) 450 MHz clock frequency

(v) 8-bit/16-bit arithmetic SIMD.

In the H.264 encoder SIMD optimization, the most significant instructions of the ST240 ISA are the following: the SIMD add.ph and sub.ph which perform, respectively, the packed 16-bit addition or subtraction; the perm.pb instruction which performs byte permutations and the muladdus.pb which multiplies an unsigned byte by a signed byte in each of the byte lanes and then sums across the four lanes to produce a single result. Furthermore, several data manipulation instructions are defined: pack.pb packs 16-bit values to byte elements ignoring the upper half; shuffeve.pb and shuffodd.pb, respectively, perform 8-bit shuffle of even and odd lanes. Two averaging operations (avg4u.pb and avgu.pb) are also defined in the instruction set.

One important operation in video-coding algorithms, the absolute value of the difference, abs (a-b), can be performed with the absubu.pb instruction (Figure 1) which works on each byte lane (treating each byte lane as an unsigned value) and returns the result in the corresponding byte lane of the destination register. The sadu.pb (Figure 1) performs the same operation and then sums the byte lanes value and returns the result.

3.2. xSTream. xSTream is a multiprocessor dataflow architecture for high-performance embedded multimedia streaming applications designed at STMicroelectronics [20, 21].

xSTream is constituted by a parallel distributed and shared memory architecture. It is an array of processing elements connected by a Network on Chip (NoC) with specific hardware for management of communication [22], as depicted in Figure 2.

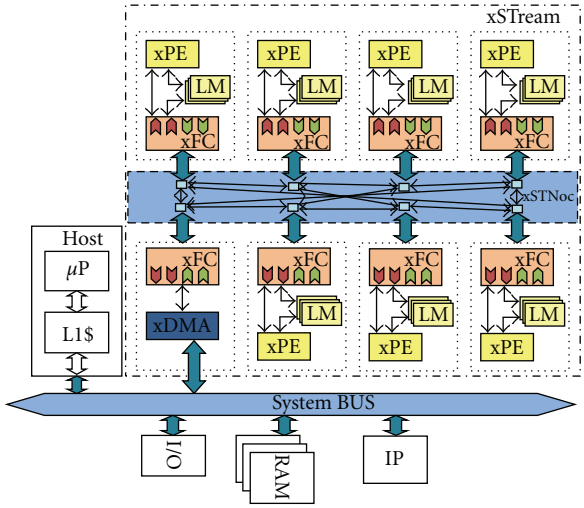

Figure 2: xSTream architecture.

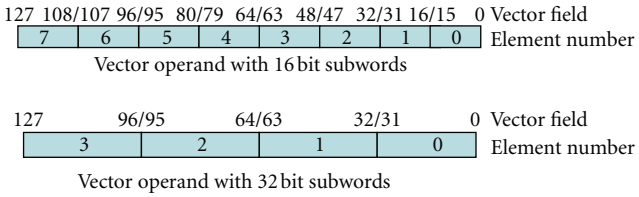

Figure 3: Vector operand.

The main elements in Figure 2 are the general purpose engine, the xSTreaming Processing Engines (XPEs) and the NoC interconnecting all components.

The XPEs are based on ST231 VLIW processors [22] of ST200 STMicroelectronics family [18, 19]. The main features can be resumed as

(i) 2-issue VLIW,

(ii) 128-bit vector registers,

(iii) up to 512 KB local memory cache,

(iv) up to 1 GHz clock frequency, and

(v) 16-bit/32-bit arithmetic SIMD.

In order to achieve excellent performance, the XPE core tries to exploit available parallelism at various levels. It supports a plethora of SIMD instructions to exploit available data-level parallelism. These instructions concurrently execute up to four operations on 32-bit operands or eight operations on 16-bit operands. The core supports wide 128-bit load/store.

The xSTream architecture handles scalar and vector operands.

Vector operands are 128-bit wide and consist of either eight 16-bit half-words or four 32-bit words, as shown in Figure 3.

In the xSTream ISA each SIMD instruction has an additional operand allowing permuting the result's element positions or replicating any element in the other positions.

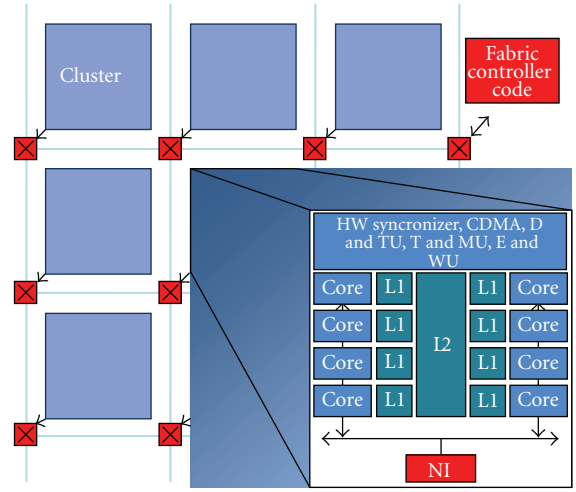

Figure 4: P2012 scheme.

This feature considerably increases the SIMD flexibility because the results have often to be reordered for further elaboration. This is especially true for video-coding algorithm with operations performed on several steps where the input of next step is usually the output of previous one. The permutation operand allows this with the cost of only one additional cycle. This leads to reduced costs to perform all the operation needed for data reordering.

The XPE supports horizontal SIMD as well. This kind of SIMD allows operations among elements in the same vector, and it is a key feature for speeding up execution in several H.264 functional units, as we will see in next sections.

3.3. Platform 2012 (P2012). Platform 2012 is a high-performance programmable architecture for high computational demanding embedded multimedia applications, currently under joint development by STMicroelectronics and Commissariat à l'énergie atomique et aux énergies alternatives (CEA) [23]. The goal of P2012 platform is to be reference architecture for next generation of multimedia product.

The P2012 architecture (Figure 4) is constituted by a large number of decoupled clusters of STxP70 processors interconnected by a Network on Chip (NoC). Each cluster can contain a number of computational elements ranging from 1 to 16. The main features of the STxP70 processor element are as follows:

(i) 32-bit RISC processor (up to 2 instructions per cycle),

(ii) 128-bit vector registers,

(iii) 256 KB of memory shared by all the processors (per cluster),

(iv) 600 MHz clock frequency, and

(v) 16-bit/32-bit arithmetic SIMD.

The P2012 basic modules can be easily replicated to provide scalability [24]. Each module is constituted by a computing cluster with cache memory hierarchy and a communication engine. The STxP70 is dual issue application-specific

instruction-set processor (ASIP) [25] with domain-specific parameterized vector extension named VECx. STxP70 SIMD instructions are used to exploit available data level parallelism [26]. These instructions execute in parallel up to four operations on 32-bit operands or eight operations on 16-bit operands, while 128-bit load/store is supported.

Vector operands are 128-bit wide and consist of either eight 16-bit half-words or four 32-bit words. In order to increase the SIMD flexibility, instructions able to permute data positions inside the vector operands are defined in the instruction set. The support to horizontal SIMD is limited at operation involving only two adjacent elements inside a vector, but its presence is fundamental for typical video-coding operation like sum of absolute difference (SAD).

3.4. SIMD Instruction-Set Evaluation. Whatever platform we choose, we will have a limited number of SIMD instructions because of hardware constraints. For this reason, besides precision and size, one of the key issues while choosing a SIMD extension is generality versus application-specific instructions. The former can show good speedups for a large variety of applications. The latter can reach greater performance, but limited to a particular family of applications. Of course, there are a lot of solutions that lay in the middle.

The vector register size impacts performance, hardware reliability, and costs. The choice of the optimal size and precision of SIMD instructions is a key factor for reaching the desired performance for the target application. The axiom larger SIMD equal to better performance may be valid for applications having no constraints and data dependencies in either spatial or temporal field. It is not the H.264 encoder condition. In general, algorithms with a heavy control flow are very difficult to vectorize, and the SIMD optimization does not always lead to the desired performance enhancement.

The application developers should choose the dimension that best fit their needs, as well as ISA designers should take into account the requirements of the application families they are targeting. As stated in [7], in a processor designed to handle video-coding standards for which the theoretical worst-case video sequence will consist of a large number of 4×4 blocks, four-way SIMD parallelism makes full use of data paths. In this case, increasing the size will lead to little performance improvement. In contrast, if we focus on the H.264's fidelity range extensions, with their 8×8 transform and 8×8 intraprediction, an ISA with eight-way SIMD parallelism will yield to better performance. Next generation video-coding standards like HEVC will use wider ranges of block sizes for both prediction and transformation processes, making the choice of the optimal vector register size even more complicated.

4. H.264 Encoder Implementation

4.1. Software Partitioning. In order to support real-time video encoding addressing HD resolutions, multiprocessor architectures seem to be an optimal solution, as earlier explained. Moreover, we would like to test the multicore architectures with an application of high interest but not so suitable for these kind of architectures in order to stress the architecture design and to evaluate possible issues and finding solutions that could be also useful for other applications.

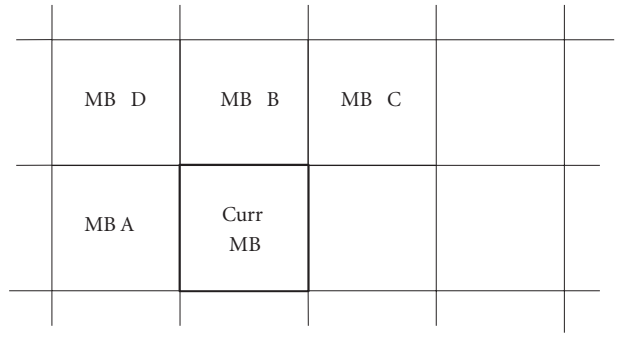

Figure 5: MB neighbours.

The first programmer's task dealing with this type of platforms is the subdivision of the encoder application in modules that can execute in parallel. The H.264 encoder partitioning plays a fundamental role in multicore architectures as xSTream and P2012, where each functional block has to meet the resources of processor elements, and the interconnection system must fulfil the memory bandwidth needed to feed the modules. The designer choice becomes more complex when some modules can run in parallel avoiding stalls in pipeline [26].

Even if a detailed description of the encoder partitioning is beyond the scope of this paper, we can here depict some issues we faced approaching this process and the solutions we adopted.

First of all, it is worth to take into account the data dependency inside the H.264 encoder. Temporal data dependency is implicit in the Motion Estimation mechanism; the coding of the current frame always depends on the previously encoded frame(s) that are used as reference. Thus, there is always a temporal data dependency, except if the current frame is an I picture. Anyway, the encoding process also shows a spatial data dependency between macroblocks, that is, the basic encoding block comprising 16×16 pixel elements. While coding the current macroblock (MB), we need data from the previously encoded MBs belonging to the same frame, or, to be more precise, to the same slice (a sequence of MBs in which the frame can be segmented). Figure 5 shows the current MB together with the already reconstructed neighbours that are needed for its prediction. Specifically, MB A, B, C, and D are required for intraprediction, motion vector prediction, and spatial direct prediction (in the SVC-compatible version). Furthermore, MB A and B are used to check the skip mode in P frames.

Spatial data dependency can even occur inside a MB. The prediction of a 4×4 block may depend on the results of already-predicted neighbouring blocks. e.g., this occurs in Intra 4×4 or in the deblocking filter.

In this scenario, we cannot encode two frames in parallel, because of the temporal data dependency, and we cannot concurrently process different MBs, because of the spatial

Table 1: Encoder data flow.

Module	Input	Output	Input from local buffers
MV prediction	MV A	MV pred	MV B, C, D
Motion estimation	Search window, original MB, MV predictor	MV, cost, best intermode, MB predictor	
Intraprediction	Original MB, reconstructed MB A	Cost, best intramode, MB predictor	Reconstructed MB B, C, D
Residual coding	Original MB, MB predictor	Residual signal, coded MB parameters	
IDCT-DeQuant and reconstruction	Residual signal	Reconstructed MB	
Deblocking filter	Reconstructed MB A	Decoded MB	Reconstructed MB B
Entropy coding	Coded MB parameters	Output stream	

data dependency, unless the MBs belong to different slices. Thus, one opportunity is to concurrently process every slice, but this solution has two drawbacks; it strongly depends on the particular encoder configuration, and it requests to implement the whole encoder on every processor element. Therefore, the only chance to partition the encoder is during the MB processing. This does not mean to separately process 8×8 or 4×4 blocks, but to separately execute the encoder functional units at MB level.

The encoder partitioning should now derive from an evaluation of the functional units that can be concurrently computed, taking into account the amount of data that needs to be exchanged between the different cores.

If we suppose that each module will run on a different core, we must consider both the chunk of data each core needs to exchange with the interconnected cores and the frequency of such communications. Therefore, for an optimal module partitioning, it is important to analyse the encoder data flow. Basically, this analysis should result in a list of selected modules with a set of input and output data for every list's entry, as shown in Table 1. The Figure 5's notation is used to indicate the neighbouring MBs. This table allows identifying the dependencies between modules as well as the data flow, from which we can obtain the requested bandwidth for the communication mechanism between processor elements. This preliminary analysis also produces the partition diagram, shown in Figure 6.

Each module will keep local memory buffers containing the data required to process the current MB. For example, the Intraprediction module needs to store a row of reconstructed MBs plus one MB (the left MB) in order to be able to predict the current MB. The deblocking filter will need to store the same number of reconstructed MBs as well. These local storages are filled by producer modules as soon as they complete the respective tasks. In the previous example, "IDCT-DeQuant & Reconstruction" is the producer for intraprediction; when the MB reconstruction has completed for MB_n, the intraprediction of MB_{n+1} can start. It is worth noting that the intraprediction of MB_{n+1} can be concurrently executed with the motion estimation of MB_{n+1} and the deblocking filter of MB_n.

For the sake of simplicity we did not put into Figure 6 scheme all the project components. The buffer mechanism for passing reference-frame data to the ME and the decoded

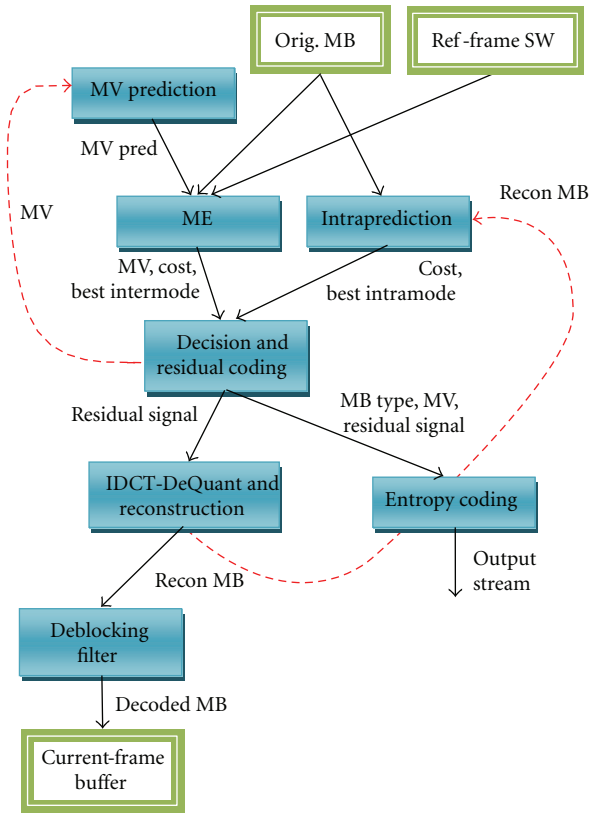

Figure 6: Encoder partition diagram.

picture buffer are not described. We preferred to focus on the encoder data flow in order to highlight the chances for module parallelisation. Moreover, the buffering mechanisms strongly depend on the architecture design implementation.

The here described partitioning seems to both fulfil the data dependency constraints and exploit the few opportunities of parallel execution available in a H.264 encoder. Moreover, the computational weight of the encoder components is quite well distributed among the different cores. The only exception is the ME, which is the most time-consuming module. In our encoder we utilize the SLIMH264 ME algorithm [27]. SLIMH264 is divided into two different stages: the first phase is common to all the partitions and

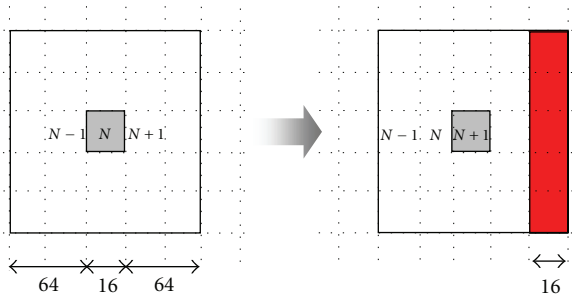

Figure 7: Search window update.

performs a fast search; the second step utilizes the coarse results coming from the first phase to refine the search for every MB partition. The second step can be executed in parallel for every MB partition. This leaves the designer the freedom to subsequently split the module to eventually avoid stalls in pipeline in the likely case the ME requires more cycles than intraprediction.

Among the issues the designer should take into account, there is still the memory bandwidth needed to feed the modules. From Table 1, we can notice that the ME module requests the largest amount of data. Besides coding parameters, the ME should receive data belonging to two frames: current and reference frame. For each MB, the data passed to the ME consists of the original MB luma values and the portion of reference frame enclosed by the search window (SW). Supposing one byte per luma sample and a SW set to 64×64 pixels (a suitable value for HD formats), we will get a width of $(64 + 16 + 64)$ pixels leading to 20736 bytes. Thus, we had 20736 bytes plus the original-image MB 16×16 bytes to send to the ME module for every MB. This leads to a very large memory bandwidth. Anyway, as could be noticed in Figure 7, not all of the SW must be resent every time a new MB is coded. Since MBs are coded in raster-scan order and search window of neighbouring MB overlaps, just a 16-byte-wide column update can be sent after the first complete window, as described in [28, 29]. Figure 7 shows the SW for the MB_N (left side) and the SW for the next MB (right side). The amount of data sent to the ME module for coding MB_{N+1} is shown as a red rectangle. When reaching the end of the row, MB_{N+1} does not need the update because this will be over the image border. Nevertheless, a SW update is written to the array, and it will be part of the SW of the first MB in the next row.

4.2. Modules Optimization. The H.264 encoder modules work on a block basis. Even though the basic block of the coding process is the macroblock, consisting of 16×16 pixel elements, the basic block of each module's computation can vary from 4×4 to 16×16. A number of experiments carried out at STMicroelectronics's Advanced System Technology Laboratories showed that, addressing HD resolutions, it is possible to disable interprediction modes involving the 8×8 blocks subpartitions without significant effects on video-quality and -coding efficiency. The same experiments also showed that fidelity range extensions are needed to improve video quality at high resolutions, as one could expect. For this reason, we choose both to disable ME on partitions 4×8, 8×4, and 4×4 and to add intra 8×8 and transform 8×8. In this scenario, most of the encoder modules work on 8×8 blocks of 8-bit samples. The 4×4 blocks are still used in Intra-4×4 and DCT/Q/IQ/IDCT 4×4. The Intra-16×16 prediction works on the whole MB, whereas the correspondent transformations just iterate the 4×4 procedures.

Usually, inside each module the computations require 16-bit precision for intermediate results. Thus, a typical situation is as follows:

(i) load 8-bit samples from memory;

(ii) switch to 16-bit precision and compute the results;

(iii) store the results to memory as 8-bit samples.

Some of the modules, or at least some parts of them, require a 32-bit precision. Among them, it is worth noting a few computations for pixels interpolation and the Quantization and Inverse-Quantization process.

In order to evaluate the different performance achievable with the three different ISAs, we have inserted the SIMD instructions in an already optimized ANSI C code which is used as reference to evaluate the achieved speedup. For a better understanding of the presented work, the comparison is not only carried out at global level, but for every H.264 functional unit.

In the following, the implementation detail of the sum of absolute difference (SAD) and the Hadamard filter will be shown for all the three addressed ISAs. Among all the several modules implementations, we have chosen to describe these particular operations for different reasons: the SAD is one of the most time-consuming operations in video-compression algorithms; the implementation of Hadamard filter is a good example for describing how an ANSI C implementation can be rewritten to best fit the available SIMD ISA. The access to data stored in memory will be discussed as well because it is a typical issue in optimizing video compression algorithms using SIMD instructions. A complete description of the encoder SIMD implementation on the ST240 processor can be found in [30].

4.2.1. SAD Operation. The sum of absolute differences is a key operation for a large variety of video-coding algorithms. The number of times this operation is executed during a coding process can vary depending on the encoder implementation and it strongly depends on the motion estimation module, that it is not covered by the H.264 standard definition. Anyway, independently of specific implementations, this operation is a key factor for the whole-encoder performance.

Here, we will show three different SAD implementations using SIMD instructions, and we will compare them with an optimized ANSI C code.

Given the essential role the SAD plays in video coding algorithms, some instruction sets include specific instructions to speed up such operation. Here, we will compare SIMD instruction sets having different size and different degree of specializations.

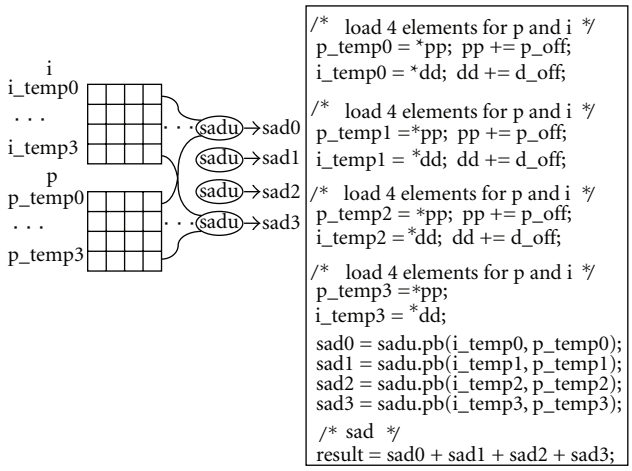

Figure 8: SAD implementation.

Table 2: SAD performance.

	Cycles	Operations	Load	Store
ANSI C version	36	134	8	0
SIMD version	14	30	8	0

Using the ST240 32-bit wide SIMD extension, the optimization of the SAD computation has been quite straightforward thanks to the SIMD instruction *sadu.pb*.

The SAD finds the "distance" between two 4×4 blocks, generally between a prediction block and the original image; given the two blocks in the left side of Figure 8, the pseudocode computing the SAD can be viewed in the right side of the same figure. Besides loading the input data, it basically consists of four calls to the *sadu.pb* instruction.

The achieved speedup is shown in Table 2.

The xSTream and P2012 architectures support 128-bit-wide vector registers, and they can perform 8-bit, 16-bit, or 32-bit arithmetic SIMD operations. Usually, SAD is performed using 8-bit precision, allowing for each SIMD calculation a capability to handle sixteen elements. Using vertical SIMD instructions, it is easy to achieve the absolute difference among several elements stored in two vectors, but the addition of the elements stored in a single vector is onerous because usually it requires several vertical SIMD inefficiently utilized. Both P2012 and xSTream ISAs have horizontal addition of SIMD instructions, but with different capability. In xSTream, it is allowed adding all the elements stored in the same vector, producing a scalar result. In P2012, VECx horizontal addition is limited to only add two adjacent elements inside a vector; in this way, four SIMD instructions must be used in order to achieve the scalar result of the SAD operation. This difference significantly impacts the encoder optimization. For example, when the SAD is calculated to evaluate the predictor cost in Intra 16×16, only two SIMDs are used with xSTream against the six used with P2012. This is schematized in Figure 9.

Even if in P2012 ISA the lacking of a horizontal SIMD for addition partially wastes the obtained great gain, we still complete the SAD operation using six VECx instructions and one scalar instruction, as shown in Figure 9, versus the 48 scalar instructions used in the ANSI C implementation (16 subtractions, 16 absolute values, and 16 additions).

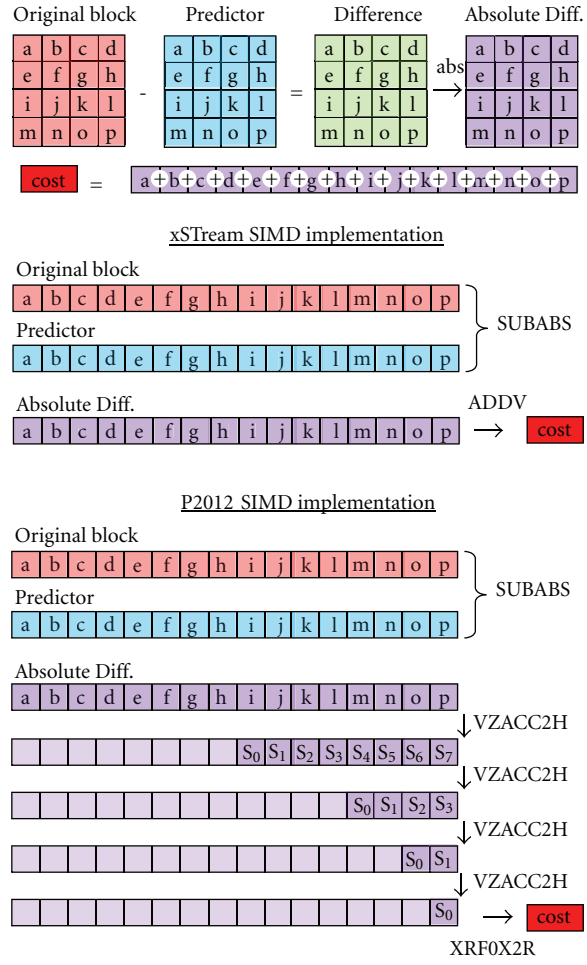

Figure 9: Predictor cost calculation.

4.2.2. Hadamard. We consider very interesting the Hadamard SIMD optimization because it involves a large number of instructions and can be considered a typical case study.

Although the Hadamard transform it is not currently used in the rest of the encoder, the intraprediction module utilizes such transform to find the best 16×16 intraprediction mode. The intramodule divides the predicted MB into sixteen 4×4 blocks. Each block is compared to the correspondent original-image's block, and sixteen differences are calculated. These sixteen values are filtered through the Hadamard transform before computing the SAD of the whole MB.

In the ST240 code, the optimization has started considering that Hadamard can be subdivided into two different phases: horizontal and vertical. The horizontal phase can be subdivided into 4 rows as well as the vertical phase into 4 columns, as shown in the portion of pseudocode in Table 3.

Table 3: Hadamard phases.

Horizontal phase	Vertical phase
/* first row */	/* first column */
$m0 = d0 + d3 + d1 + d2;$	$w0 = m0 + m12 + m4 + m8;$
$m1 = d0 + d3 - d1 - d2;$	$w1 = m0 + m12 - m4 - m8;$
$m2 = d0 - d3 + d1 - d2;$	$w2 = m0 - m12 + m4 - m8;$
$m3 = d0 - d3 - d1 + d2;$	$w3 = m0 - m12 - m4 + m8;$
/* second row */	/* second column */
$m4 = d4 + d7 + d5 + d6;$	$w4 = m2 + m14 + m6 + m10;$
$m5 = d4 + d7 - d5 - d6;$	$w5 = m2 + m14 - m6 - m10;$
$m6 = d4 - d7 + d5 - d6;$	$w6 = m2 - m14 + m6 - m10;$
$m7 = d4 - d7 - d5 + d6;$	$w7 = m2 - m14 - m6 + m10;$
/* third row */	/* third column */
$m8 = d8 + d11 + d9 + d10;$	$w8 = m1 + m13 + m5 + m9;$
$m9 = d8 + d11 - d9 - d10;$	$w9 = m1 + m13 - m5 - m9;$
$m10 = d8 - d11 + d9 - d10;$	$w10 = m1 - m13 + m5 - m9;$
$m11 = d8 - d11 - d9 + d10;$	$w11 = m1 - m13 - m5 + m9;$
/* fourth row */	/* fourth column */
$m12 = d12 + d15 + d13 + d14;$	$w12 = m3 + m15 + m7 + m11;$
$m13 = d12 + d15 - d13 - d14;$	$w13 = m3 + m15 - m7 - m11;$
$m14 = d12 - d15 + d13 - d14;$	$w14 = m3 - m15 + m7 - m11;$
$m15 = d12 - d15 - d13 + d14;$	$w15 = m3 - m15 - m7 + m11;$

In the pseudocode, d_i are the differences and m_i the intermediate values of the transform.

Once we have all the differences contained in packed 16-bit values subdivided into even and odd pairs, we can rewrite the first row of the horizontal Hadamard transform as

$$\begin{aligned} m0 &= (d0 + d1) + (d2 + d3), \\ m1 &= (d0 - d1) - (d2 - d3), \\ m2 &= (d0 + d1) - (d2 + d3), \\ m3 &= (d0 - d1) + (d2 - d3). \end{aligned} \tag{1}$$

In such a way, we can exploit the packed 16-bit addition and subtraction to obtain the high and low halves of the m_i coefficients. As can be noted, the low and high halves of $m0$ and $m2$ are the same, but while the $m0$'s value is achievable by adding its halves, to compute the value of $m2$ we have to subtract its high half from the lower one. Similar considerations can be applied to the odd elements $m1$ and $m3$.

Anyway, since the vertical phase of Hadamard is yet to come, there is no need to compute such values at this point. In fact, we can rewrite the m_i coefficient as functions of their own halves as follows:

$$\begin{aligned} m0 &= m0L + m0H, \\ m1 &= m1L - m1H, \\ m2 &= m2L - m2H, \\ m3 &= m3L + m3H. \end{aligned} \tag{2}$$

and utilize this notation to rewrite the vertical phase of the Hadamard transform as described below

$$\begin{aligned} w0 &= (m0 + m4) + (m8 + m12) \\ &= (m0L + m4L) + (m0H + m4H) \\ &\quad + (m8L + m12L) + (m8H + m12H) \\ &= (m0L + m4L) + (m8L + m12L) \\ &\quad + (m0H + m4H) + (m8H + m12H) \\ &= w0L + w0H. \end{aligned} \tag{3}$$

We can use the low and high halves of the intermediate coefficients to compute the low and high halves of the final coefficients w_i as illustrated in Figure 10.

The Hadamard optimization with ST240 SIMD is quite complex. Due to shortness of SIMD, the standard algorithm has been modified in order to better match the SIMD ISA features.

Using the xSTream and P2012 architectures, we followed a different approach. Our goal was the exploitation of 128-bit-wide SIMD minimizing the data reordering. Considering that the Hadamard transform can be defined as

$$H_n = \begin{bmatrix} H_{n-1} & H_{n-1} \\ H_{n-1} & -H_{n-1} \end{bmatrix} \qquad H_0 = 1, \tag{4}$$

the Hadamard matrices are composed of ± 1 and are a special case of discrete fourier transform (DFT). For this reason, the calculation can exploit the FFT algorithm, usually known as Fast Walsh-Hadamard transform [31].

The only issue is obtaining a good implementation of the FFT butterfly with SIMD, avoiding wasting all the gain achieved using fast algorithm with the data reordering needed to implement the calculation. Our approach consists of a modified butterfly that allows using always the same butterfly structure for every level, even if we have to reorder data between stages (Figure 11).

The output values coming from every butterfly can be calculated for 16 samples at a time using two SIMD instructions, one calculating the additions and one calculating the differences. In this way, we have the advantage of computing the output of each level using a simple SIMD implementation, at the cost of swapping intermediate results between different levels.

Even if xSTream and P2012 share this implementation mechanism, we have measured different performance. In this case, the difference depends on the different types of data manipulation instructions. The xSTream ISA having the third operand allowing the permutation of results inside vectors is more flexible and can implement the above algorithm with a reduced number of instructions respect to VECx P2012 ISA. Algorithm 1 shows the xSTream SIMD implementation.

4.2.3. Memory Access Issues. As previous exposed, a key factor to achieve a good performance improvement with

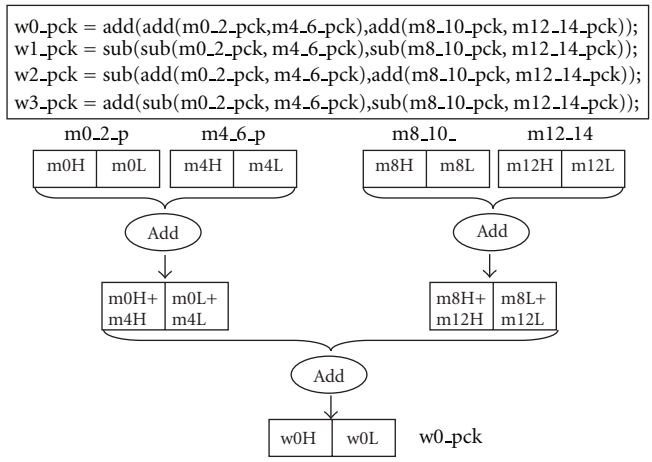

Figure 10: Hadamard vertical phase with ST240 SIMD.

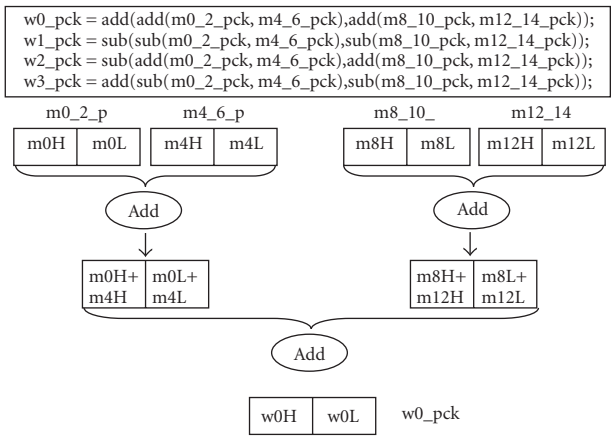

Figure 11: Hadamard modified butterfly.

SIMD optimization is the efficient handling of unaligned load operations. In general, programmers should structure the application data in order to avoid or minimize misaligned memory accesses. In video compression algorithm, the motion compensation is surely a case where it is not possible avoid unaligned memory accesses because it is impossible to predict motion vectors and consequently align data.

None of the three addressed architectures support unaligned load instructions. Therefore, it is important to efficiently use aligned accesses to load misaligned data from memory. The three ISAs support instructions to concatenate two vectors. This allows a solution consisting in two steps: first, we use two aligned load instructions for loading data in two vector registers, and, then, we concatenate and shift their elements in order to extract a single vector containing the needed data, as shown in Figure 12.

```
/*first level: one 16 samples butterfly*/
/*(s0 ÷ s7)+(s8 ÷ s15)*/
vaddh out_low = in_low, in_high
/*(s0 ÷ s7)−(s8 ÷ s15)*/
vsubh out_high = in_low, in_high

/*data reordering*/
/*0 1 2 3 8 9 10 11*/
vmrgbl in_low = out_low, out_high, perm
/*4 5 6 7 12 13 14 15*/
vmrgbu in_high = out_low, out_high, perm

/*second level: two 8 samples butterfly*/
vaddh out_low = in_low, in_high
vsubh out_high = in_low, in_high

/*data reordering*/
/*0 1 8 9 4 5 12 13*/
vmrge in_low = out_low, out_high
/*2 3 10 11 6 7 14 15*/
vmrgo in_high = out_low, out_high

/*third level: four 4 samples butterfly*/
vaddh out_low = in_low, in_high
vsubh out_high = in_low, in_high

/*data reordering*/
/*0 8 2 10 4 12 6 14*/
vmrgeh in_low = out_low, out_high
/*1 9 3 11 5 13 7 15*/
vmrgoh in_high = out_low, out_high

/*fourth level: eight 2 samples butterfly*/
vaddh out_low = in_low, in_high
vsubh out_high = in_low, in_high
```

Algorithm 1: Hadamard transform xSTream SIMD implementation.

```
uint32 AddressAt128;
vector_16b_sw Va, Vb, Vout;

AddressAt128b = ((uint32) (mref_ptr)) & (~0xF);
Offset = ((uint32) (mref_ptr)) & (0xF);
Va = ldq(AddressAt128, 0);
Vb = ldq(AddressAt128, 16);
Vout = wrot(Va, Vb, Offset);
```

Algorithm 2: Unaligned load SIMD implementation with concatenate instruction.

```
ui32_t PackCurr0 = *(orig_line);
ui32_t PackCurr1 = *(orig_line+1);
/* Pack to 128 bits */
TmpVectArray[0] = PackCurr0;
TmpVectArray[1] = PackCurr1;
Pack128In = ldqi(Pack128In, TmpVectArray,0);
/* Reorganize pixels */
Va = vmrgbeh(Va,Pack128In,VZero,permute0);
Vb = vmrgboh(Vb,Pack128In,VZero, permute1);
VPackCurr = vaddh(VPackCurr,Va,Vb,0);
```

Algorithm 3: Unaligned load SIMD implementation without concatenate instruction.

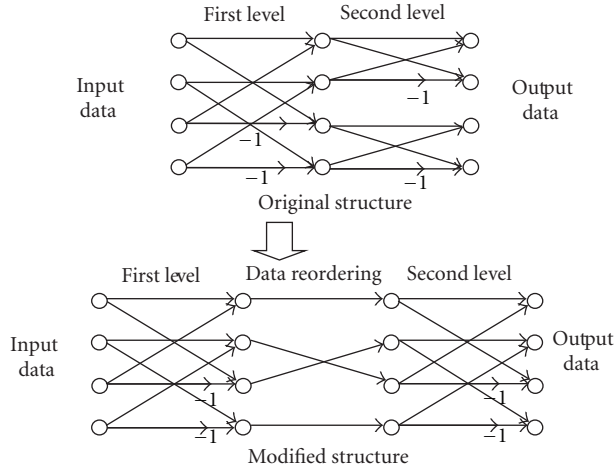

Figure 12: Unaligned load.

If an ISA does not define a SIMD performing this type of concatenate operation, then the unaligned load will be implemented with an extra cost due to the use of additional instructions for merging data between the two vectors.

Algorithm 2 shows the implementation of an unaligned load using xSTream. This solution can be compared to the same operation carried out without a concatenate instruction shown in Algorithm 3, in which we should add three instructions to reorganize the data for composing the required not-aligned vector.

It is very important that these concatenate instructions can take the offset argument not as a constant value but as a variable value; otherwise, modules such as motion compensation would not get any benefit from using them. For example, the Intel SSSE3 "palignr" instruction concatenates two operands and shift right the composite vector by an offset for extracting an aligned results, but the offset must be a compile-time constant value. This is a big issue for a module as motion compensation, in which it is impossible to know in advance the offset of a misaligned address.

5. Results

In the H.264 encoder, the most cycle-demanding modules have been optimized using SIMD instructions: motion estimation and compensation, DCT, Intraprediction, and so forth. The best way to compare different instruction sets in order to judge the effectiveness of both SIMD extensions and code optimizations is to measure the speedup obtained with the SIMD-based implementation versus the ANSI C version of the same source code. In order to separate the effect of SIMD performance improvement from ANSI C optimizations, we have inserted SIMD instructions in previously optimized ANSI C modules.

The results are provided in terms of average cycles spent to process one macroblock. The xSTream and P2012 architectures share the same modules subdivision. For the single-core DSP ST240, the subdivision is less fine, and related modules are joined together. In the reported tests, the presence of the ST240 processor is important because it allows comparing the single-processor elements of the multicore platforms to a single-core architecture. Tests are performed on a set of video sequences addressing different

Table 4: SIMD instructions for video coding.

Instruction description	Affected modules	Notes
Horizontal add: adds all the elements inside a vector register and produces a scalar result	ME, intraprediction	Speeds up SAD
Horizontal permute: rearranges elements inside a vector register	Intraprediction, DCT/Q/IQ/IDCT	Allows zig-zag scan and speeds up intra diagonal modes
Concatenate: concatenates two vector registers into an intermediate composite, shifts the composite to the right by a variable offset	Motion estimation and compensation	Allows software implementation of unaligned load
Promotion/demotion precision: an efficient support for promoting element precision while loading data from memory, and demoting the precision (with saturation) while storing data to memory	All the main modules	It will speed up the load and store operations for several modules
Absolute subtraction: for every element "a" in the first vector and every element "b" in the second vector performs the following operation: $\lvert a - b \rvert$	ME, intraprediction, deblocking filter	Speeds up SAD in conjunction with horizontal add; used in deblocking filter
Shift with round: performs the following operation for every element "a" in the vector operand: $(a + 2^{n-1}) >> n$, where n is a scalar value	IDCT, deblocking filter, motion compensation	Speeds up 1/2 pixel interpolation
Average: for every element "a" in the first vector and every element "b" in the second vector performs the following operation: $(a + b + 1) >> 2$	Intraprediction, deblocking filter, motion compensation	Speeds up 1/4 pixel interpolation

Table 5: Cycles/MB spent in each module for each ISA.

	xSTream			P2012			ST240		
	ANSI C	SIMD	Gain factor	ANSI C	SIMD	Gain factor	ANSI C	SIMD	Gain factor
Luma motion compensation	4788	2257	2.1x	8286	3965	2.1x			
Croma motion compensation	3064	658	4.7x	3626	1282	2.8x	265559	200380	1.3x
Motion estimation	303769	84342	3.6x	603182	114776	5.3x			
Intra 4×4	24366	10076	2.4x	38234	15760	2.4x	32013	19182	1.6x
Intra 8×8	15396	4997	3.0x	26972	9455	2,9x			
DCT/Q/IQ/IDCT 4×4	14994	7616	2.0x	20473	9088	2.3x	32013	19182	1.7x
DCT/Q/IQ/IDCT 8×8	18660	3498	5.3x	24486	11636	2.1x			

resolutions, and average results are resumed in Table 5. The results in Table 5 and Figure 13 show that the ST240, exploiting the instruction level parallelism (ILP) with a 4-issue VLIW architecture, achieves the best performance for the ANSI C implementation. All the SIMD implementations improve performance for every encoder module, but the ST240 with the shortest SIMD size obtains the lowest speedup factor. P2012 and xSTream with their wider SIMD can better exploit the data-level parallelism. In terms of pure number of cycles spent to encode one macroblock, the xSTream ISA achieves the best performance.

It is worth analyzing in detail these results to understand how different instruction sets lead to different performance. The xSTream processor elements take advantage from the "horizontal add" instruction that allow an efficient computation of the SAD operations: it is evident in the ME module, where xSTream spends about 25% fewer cycles than P2012 (84,342 versus 114,776 cycles/MB). The higher speedup obtained by P2012 is mainly due to the less-efficient ANSI C code generated by the P2012 compiler. We already described as the ST240 can exploit a specific instruction for the SAD operation. In fact, its result is not far from the architectures having 128-bit-wide vector registers (the 200, 380 cycles/MB also include motion compensation). From these results, we can state that the support for horizontal SIMD will not only give a great performance improvement for the SAD operation, but it significantly impacts the whole ME module.

As earlier said, data manipulation instructions are a key factor to fully exploit SIMD implementations because operations such as transposing matrices or data reordering become frequent in this type of optimizations. An experimental result confirming this consideration can be seen in the DCT/Q/IQ/IDCT 8×8 module, covering all the toolchain performing the residual coding and decoding. This module involves several data-reordering operations, ranging from matrix transposition to zig-zag reorder. Both ST240 and xSTream instruction sets support the permutation of elements inside a vector in a very efficient way, as described in Sections 3.1 and 3.2. The P2012 SIMD extension includes a series of instructions for interleaving and merging elements between two vector operands. The great speed up the

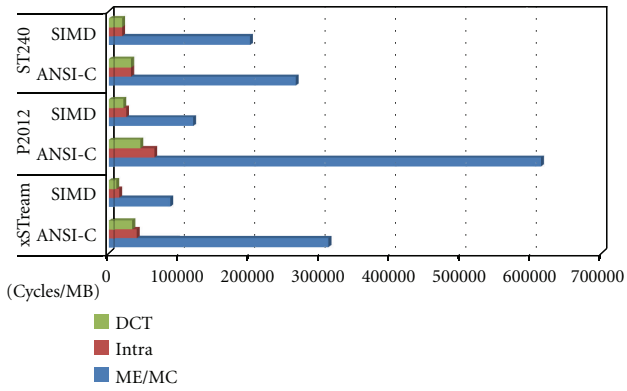

Figure 13: ISA comparison.

xSTream architecture gathers in comparison with P2012 is mainly due to the possibility to permute elements using a single instruction, in a sort of horizontal permute. The effect is emphasized in the 8×8 transform where the data reordering process is stressed more than in the 4×4 case. In our experience, we saw that if such instruction is available, then the zig-zag reordering can be effectively implemented with SIMD instructions; otherwise, we are forced to use the scalar implementation involving look-up tables to perform the reordering.

Intraprediction can exploit the horizontal permute instruction as well; the intraprediction modes involving diagonal directions require the permutation of elements inside the resulting vectors. For similar reasons, ST240 achieves great speedup factors in DCT and intramodules (resp., 1.7 and 1.6), considering that a 32-bit-wide SIMD can only perform two 16-bit-arithmetic operations.

There are other several SIMD instructions that in our opinion are to be considered as key instructions for optimizing video codec applications. Here, we assume an instruction set will already include SIMD for all the common arithmetic operations, compare, select, shift, and memory operations.

In previous sections, we already discussed about the impact of the unaligned memory access to the video codec performance. All the encoder modules are affected by the performance of the unaligned memory operations, but it becomes a keyfactor for motion estimation and compensation. An instruction concatenating two vectors and producing a vector at the desired offset is fundamental to implement an unaligned load instruction. As stated in Section 4.2.3, the capability to support variable offsets is a key factor for the instruction usability because the offset could not be known in advance.

Inside most of the modules, the computations require a 16-bit precision for intermediate results, but the input and output data contained into the noncompressed YUV images are 8-bit values. Thus, a typical operation at the beginning of a module is to load 8-bit input values and extend them to 16-bit precision. At the end, the output data precision is usually demoted down to 8-bit saturating the values before storing the results. Therefore, even if the support to 8-bit operations is not required, it would be very useful that an instruction set will include SIMD instructions for promoting and demoting precision in a fast way. An optimal solution will also combine promotion with load operations and demotion with store instructions.

Usually, the video codec algorithms try to avoid the division operations because of its computational cost. When needed, divisors are power of two, and the division is substituted with a shift right with rounding as follows:

$$\frac{a}{2^n} \Longleftrightarrow (a + 2^{n-1}) \gg n. \tag{5}$$

Therefore, even if most instruction sets already include this type of instruction, it is important to remind its utility. Often, the shift right with rounding is used for averaging two or more values, as in the intraprediction and deblocking filter. In our implementation, one of the reasons the ST240 achieves a good speedup in the intraprediction module is the presence of an average SIMD instruction in the instruction set.

Table 4 summarizes our conclusions based on the presented work. The proposed instructions are described in the first column. For each instruction, the table indicates the H.264 modules that will be mainly affected by the introduction of the instruction, as well as a few notes about specific contributions to basic video coding operations.

6. Conclusions

This paper presents efficient implementations of the H.264/AVC encoder on three different ISAs. The optimization process exploits the SIMD extensions of the three architectures for improving the performance of the most time-consuming encoder modules. For each addressed architecture, experimental results are presented in order to both compare the different implementations and evaluate the speedup versus the optimized ANSI C code.

The paper discusses how SIMD size and different instruction sets can impact the achievable performance. Several issues affecting video-coding SIMD optimization are discussed, and authors' solutions are presented for all the architectures.

Most instruction sets have specific SIMD instructions for video coding. Even though these instructions can lead to great performance improvements, they could be useless for other application families. In this paper, we identify a set of generic SIMD instructions that can significantly improve the performance of video applications.

Besides presenting the SIMD optimization for the most time-demanding modules, the paper describes how a complex application as the H.264/AVC encoder can be partitioned to a multicore architecture.

Acknowledgments

The authors would like to thank STMicroelectronics's Advanced System Technology Laboratories for their support. This work is supported by the European Commission in the context of the FP7 HEAP project (#247615).

References

[1] "VC-1 Compressed Video Bitstream Format and Decoding Process," SMPTE 421M-2006, SMPTE Standard, 2006.

[2] T. Wiegand, G. J. Sullivan, G. Bjøntegaard, and A. Luthra, "Overview of the H.264/AVC video coding standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 560–576, 2003.

[3] G. J. Sullivan, P. Topiwala, and A. Luthra, "The H.264/AVC Advanced Video Coding Standard: Overview and Introduction to the Fidelity Range Extensions," in *Applications of Digital Image Processing XXVII*, Proceedings of SPIE, August, 2004.

[4] D. Marpe, T. Wiegand, and S. Gordon, "H.264/MPEG4-AVC fidelity range extensions: tools, profiles, performance, and application areas," in *IEEE International Conference on Image Processing (ICIP '05)*, pp. 593–596, September 2005.

[5] H. Schwarz, D. Marpe, and T. Wiegand, "Overview of the scalable video coding extension of the H.264/AVC standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 17, no. 9, pp. 1103–1120, 2007.

[6] Joint Collaborative Team on Video Coding (JCT-VC), "WD4: Working Draft 4 of High-Efficiency Video Coding," 6th Meeting, Torino, Italy, July, 2011.

[7] J. Probell, "Architecture considerations for multi-format programmable video processors," *Journal of Signal Processing Systems*, vol. 50, no. 1, pp. 33–39, 2008.

[8] M. Koziri, D. Zacharis, I. Katsavounidis, and N. Bellas, "Implementation of the AVS video decoder on a heterogeneous dual-core SIMD processor," *IEEE Transactions on Consumer Electronics*, vol. 57, no. 2, pp. 673–681, 2011.

[9] M. Sayed, W. Badawy, and G. Jullien, "Towards an H.264/AVC HW/SW integrated solution: an efficient VBSME architecture," *IEEE Transactions on Circuits and Systems II*, vol. 55, no. 9, pp. 912–916, 2008.

[10] T. Rintaluoma and O. Silvén, "SIMD performance in software based mobile video coding," in *10th International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (IC-SAMOS '10)*, pp. 79–85, July 2010.

[11] H. Lv, L. Ma, and H. Liu, "Analysis and optimization of the UMHexagons algorithm in H.264 based on SIMD," in *2nd International Conference on Communication Systems, Networks and Applications (ICCSNA '10)*, pp. 239–244, July 2010.

[12] X. Zhou, E. Q. Li, and Y.-K. Chen, "Implementation of H.264 decoder on general-purpose processors with media instructions," in *Image and Video Communications and Processing*, Santa Clara, Calif, USA, January 2003.

[13] J. Lee, S. Moon, and W. Sung, "H.264 decoder optimization exploiting SIMD instructions," in *IEEE Asia-Pacific Conference on Circuits and Systems (APCCAS '04)*, pp. 1149–1152, December 2004.

[14] W. Lo, D. Lun, W. Siu, W. Wang, and J. Song, "Improved SIMD architecture for high performance video processors," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 21, no. 12, pp. 1769–1783, 2011.

[15] D. Talla, L. K. John, and D. Burger, "Bottlenecks in multimedia processing with SIMD style extensions and architectural enhancements," *IEEE Transactions on Computers*, vol. 52, no. 8, pp. 1015–1031, 2003.

[16] Z. Shen, H. He, Y. Zhang, and Y. Sun, "A Video Specific Instruction Set Architecture for ASIP design," *VLSI Design*, vol. 2007, Article ID 58431, 7 pages, 2007.

[17] M. Shafique, L. Bauer, and J. Henkel, "Optimizing the H.264/AVC video encoder application structure for reconfigurable and application-specific platforms," *Journal of Signal Processing Systems*, vol. 60, no. 2, pp. 183–210, 2010.

[18] P. Faraboschi, G. Brown, J. A. Fisher, G. Desoli, and F. Homewood, "Lx: a technology platform for customizable VLIW embedded processing," in *27th Annual International Symposium on Computer Architecture (ISCA '00)*, pp. 203–213, June 2000.

[19] J. Fisher, P. Faraboschi, and C. Young, "VLIW processors: from blue sky to best buy," *IEEE Solid-State Circuits Magazine*, vol. 1, no. 2, pp. 10–17, 2009.

[20] N. Coste, H. Garavel, H. Hermanns, F. Lang, R. Mateescu, and W. Serwe, "Ten Years of Performance Evaluation for Concurrent Systems using CADP," in *4th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation ISoLA*, Heraklion, Greece, 2010.

[21] D. Pandini, G. Desoli, and A. Cremonesi, "Computing and design for software and silicon manufacturing," in *IFIP International Conference on Very Large Scale Integration (VLSI '07)*, pp. 122–127, October 2007.

[22] G. Desoli and E. Filippi, "An outlook on the evolution of mobile terminals: from monolithic to modular multi-radio, multi-application platforms," *IEEE Circuits and Systems Magazine*, vol. 6, no. 2, pp. 17–29, 2006.

[23] L. Benini, "P2012: a many-core platform for 10Gops/mm2 multimedia computing," in *21st IEEE International Symposium on Rapid System Prototyping*, Fairfax, Va, USA, June 2010.

[24] C. Silvano, W. Fornaciari, S. Crespi Reghizzi et al., "2PARMA: parallel paradigms and run-time management techniques for many-core architectures," in *IEEE Annual Symposium on VLSI*, pp. 494–499, July 2010.

[25] C. Mucci, L. Vanzolini, I. Mirimin et al., "Implementation of parallel LFSR-based applications on an adaptive DSP featuring a Pipelined Configurable Gate Array," in *Design, Automation and Test in Europe (DATE '08)*, pp. 1444–1449, March 2008.

[26] P. Paulin, "Programming challenges & solutions for multiprocessor SoCs: An industrial perspective," in *Design Automation Conference (DAC '11)*, June 2011.

[27] A. Kumar, D. Alfonso, L. Pezzoni, and G. Olmo, "A complexity scalable H.264/AVC encoder for mobile terminals," in *European Signal Processing Conference (EUSIPCO '08)*, Lausanne, Switzerland, August 2008.

[28] C. Y. Chen, C. T. Huang, Y. H. Chen, and L. G. Chen, "Level C+ data reuse scheme for motion estimation with corresponding coding orders," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 16, no. 4, pp. 553–558, 2006.

[29] B. Zatt, M. Shafique, F. Sampaio, L. Agostini, S. Bampi, and J. Henkel, "Run-Time Adaptive Energy-Aware Motion and Disparity Estimation in Multiview Video Coding," in *48th Design Automation Conference (DAC '11)*, pp. 1026–1031, San Diego, Calif, USA, June 2011.

[30] M. Bariani, I. Barbieri, D. Brizzolara, and M. Raggio, "H.264 implementation on SIMD VLIW cores," STreaming Day 2007, Genova, Italy.

[31] C. S. Lubobya, M. E. Dlodlo, G. de Jager, and K. L. Ferguson, "SIMD implementation of integer DCT and hadamard transforms in H.264/AVC encoder," in *Proceedings of the IEEE AFRICON*, pp. 1–5, September 2011.

Hindawi Publishing Corporation
VLSI Design
Volume 2012, Article ID 748019, 20 pages
doi:10.1155/2012/748019

Research Article

Low-Complexity Hierarchical Mode Decision Algorithms Targeting VLSI Architecture Design for the H.264/AVC Video Encoder

Guilherme Corrêa,[1] Daniel Palomino,[2] Cláudio Diniz,[1] Sergio Bampi,[1] and Luciano Agostini[2]

[1] *Microelectronics Group (GME), Federal University of Rio Grande do Sul (UFRGS), Av. Bento Gonçalves, 9500, P.O. Box 15064, 91501-970 Porto Alegre, RS, Brazil*
[2] *Group of Architectures and Integrated Circuits (GACI), Federal University of Pelotas (UFPel), Campus Universitário s/n, P.O. Box 354, 96010-900 Pelotas, RS, Brazil*

Correspondence should be addressed to Guilherme Corrêa, guilherme.correa@co.it.pt

Received 18 December 2011; Accepted 1 March 2012

Academic Editor: Maurizio Martina

In H.264/AVC, the encoding process can occur according to one of the 13 intraframe coding modes or according to one of the 8 available interframes block sizes, besides the *SKIP* mode. In the Joint Model reference software, the choice of the best mode is performed through exhaustive executions of the entire encoding process, which significantly increases the encoder's computational complexity and sometimes even forbids its use in real-time applications. Considering this context, this work proposes a set of heuristic algorithms targeting hardware architectures that lead to earlier selection of one encoding mode. The amount of repetitions of the encoding process is reduced by 47 times, at the cost of a relatively small cost in compression performance. When compared to other works, the fast hierarchical mode decision results are expressively more satisfactory in terms of computational complexity reduction, quality, and bit rate. The low-complexity mode decision architecture proposed is thus a very good option for real-time coding of high-resolution videos. The solution is especially interesting for embedded and mobile applications with support to multimedia systems, since it yields good compression rates and image quality with a very high reduction in the encoder complexity.

1. Introduction

The latest technological advances in multimedia systems have brought us a wide range of multimedia-capable devices, such as mobile phones, personal digital assistants (PDAs), portable computers, and digital televisions. Recently, high-resolution video applications are much more common even in battery-operated portable devices in order to provide more and more visual quality for consumers. In addition, video streaming and digital television applications introduce the requirement of real-time processing. This scenario implies in a very high amount of data to be processed, stored, or transmitted.

Video coding standards, such as H.264/AVC [1], include a series of coding tools to achieve high compression rates while preserving the video visual quality. The prediction steps of H.264/AVC (composed by intra-frame prediction and inter-frames prediction) are responsible for the main contribution of compression provided by this standard, which results, for example, in 50% fewer bits to represent a video when compared to MPEG-2 [2]. This result was achieved through the insertion of a large number of coding modes in the prediction steps, which must be evaluated and compared to each other (for this reason they are also called candidate modes) so that the best one is selected to encode each macroblock (MB).

Rate-distortion optimization (RDO) [3, 4] is a well-known technique used in video encoding to achieve the best coding efficiency by measuring and comparing the distortion and the bit rate of an encoded MB for all

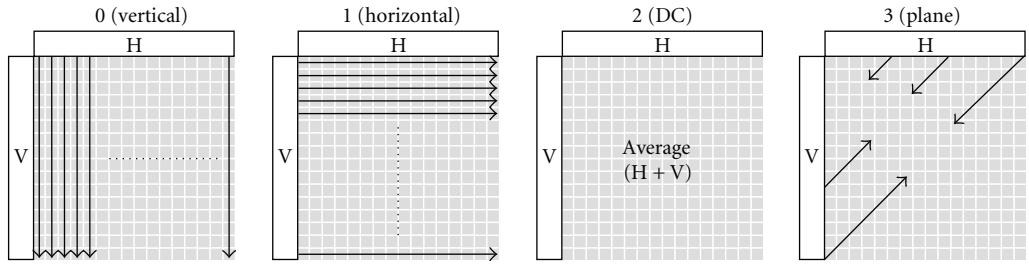

Figure 1: I16 MB prediction modes.

available prediction modes. However, the computation of a large number of prediction modes provided by H.264/AVC standard is extremely expensive in terms of computational complexity. For example, considering an HD 1080 p (1920 × 1080 pixels) video sequence with a group of pictures (GOP) composed by one intraframe followed by five interframes (IPPPPP), more than 8 million iterations composed by prediction, transform, quantization, inverse transform, inverse quantization, and entropy coding (called in this work as *encoding loop*) are needed for each video frame (148 iterations for each intramacroblock and 168 iterations for each intermacroblock). This large number of iterations leads to high energy consumption levels, which is critical in portable battery-operated multimedia devices. Besides, performance is also seriously affected, sometimes even forbidding the encoder to manipulate high-resolution video sequences in real time.

Recently, several works have proposed fast mode decision algorithms and VLSI designs for the H.264/AVC encoding process [5–12]. Even though all these works apply some heuristics at a certain point of the mode decision process, all of them use the RDO technique in some stage to select prediction modes and block sizes, limiting the reduction in terms of computational complexity and energy consumption of a fully RDO-based decision. The method presented in this paper is different. We propose a set of heuristics based on raw video data (which is available in the beginning of encoding process) to select the intra- and interframe coding modes *before* the encoding process itself, completely eliminating the use of RDO. Our technique was developed targeting a VLSI implementation into an H.264/AVC video encoder. The approach yields a computational complexity reduction of 47 times (in terms of encoding loop iterations) when compared to the original RDO-based solution. An average increase of 6.84% in bit rate and an average decrease of 0.25 dB in peak signal-to-noise ratio (PSNR) are noticed in worst-case decisions, but these drawbacks are still small when compared to the significant complexity reduction which satisfies real-time requirements of recent high-resolution video applications running into portable devices. The implemented VLSI design for intra- and interframe mode decisions allowed quantifying such gains provided by the heuristics.

The rest of this paper is organized as follows. Section 2 shows an overview of H.264/AVC coding modes and RDO-based mode decision. Section 3 presents some previous, related works found in the literature. Section 4 exposes the proposed method for fast mode decision, and Section 5 presents the VLSI design of each mode decision module. Section 6 presents rate-distortion performance results, a computational complexity analysis for the proposed method and the architectural synthesis results. Section 7 presents comparisons with related works, and Section 8 concludes this work indicating some future research directions.

2. Mode Decision in H.264/AVC

Intra-frame prediction explores the spatial redundancy in the current frame based on spatial neighbor samples, whilst inter-frame prediction, formed by motion estimation (ME) and motion compensation (MC), explores the temporal redundancy between frames.

Two intra-frame prediction types are defined for luminance (Y) samples: intra frame 16 × 16 (I16 MB) and intra frame 4 × 4 (I4 MB). For I16 MB, one mode among four prediction modes (shown in Figure 1) is chosen. Each mode generates a predicted MB using the 32 neighbor samples at the top and left boundaries of the original MB. For I4 MB, the MB is broken into sixteen 4 × 4 blocks that are individually predicted through nine prediction modes using up to 13 neighbor samples at the boundary of each 4×4 block (shown in Figure 2). For chrominance samples, the prediction is always applied over 8 × 8 blocks and uses the same I16MB modes, although it uses less neighbor samples and another order to identify the modes. The same prediction is applied for both blue and red chrominance channels [13].

In inter-frames prediction, the ME module searches in the reference frames the block which is the most similar to the current block. When the best match occurs, a motion vector (MV) is generated. One of the new features of H.264/AVC is the variable block-size motion estimation (VBSME), where one MB can be divided according to four partition types: P16 × 16, P16 × 8, P8 × 16, and P8 × 8, as shown in Figure 3. Each P8×8 partition (called sub-macroblock) could be further subpartitioned according to four subpartition types: SMB8 × 8, SMB8 × 4, SMB4 × 8, or SMB4 × 4, as shown in Figure 4. Each partition or subpartition is predicted separately and has its own MV to point to a different block of the reference frame [13].

Besides these partition and subpartition sizes, the inter-frames prediction can decide to use the *SKIP* mode, which is a special block which does not send any information to the decoder. This case happens when the rate-distortion cost of sending MB information is higher than the cost of not sending any information regarding this MB [13].

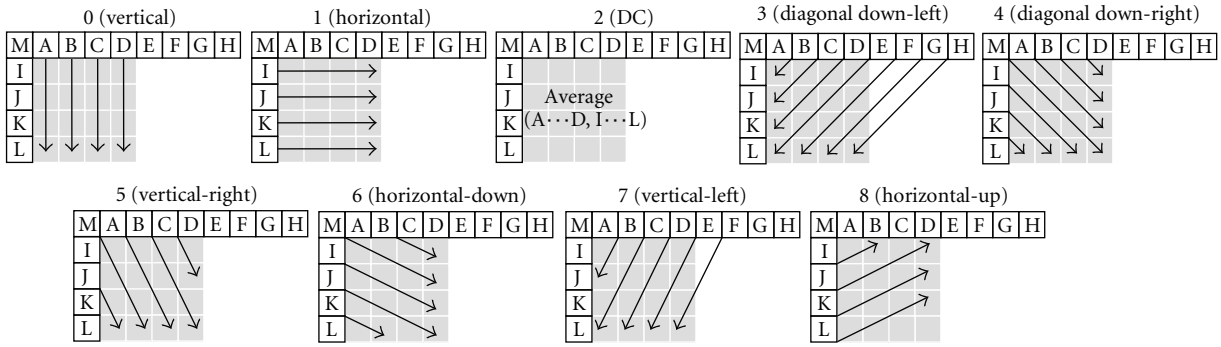

Figure 2: I4 MB prediction modes.

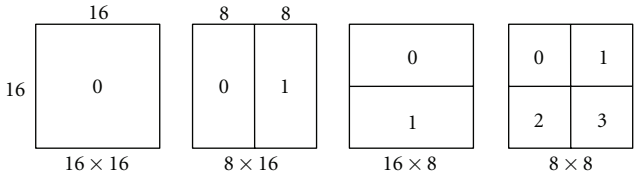

Figure 3: Partition types of a macroblock.

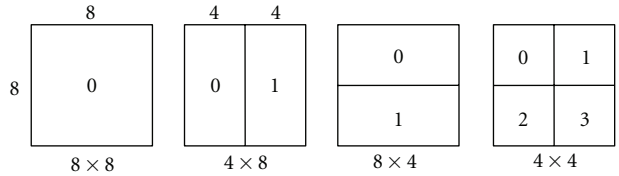

Figure 4: Subpartitions types of a P8 × 8 sub-macroblock.

As previously mentioned, the rate-distortion optimization (RDO) technique is suggested by the H.264/AVC standard to select, among all the candidate modes, which is the most efficient one in terms of compression and image quality. Equation (1) shows the Lagrangian rate-distortion cost formula, in which D denotes the distortion (measured in PSNR), R denotes the bit rate, λ denotes the Lagrangian multiplier, and J is the final rate-distortion cost

$$J = D + \lambda \cdot R. \tag{1}$$

Equation (1) is applied to each prediction mode for all MB inside a frame. Figure 5 shows the sequential execution flow of the H.264/AVC encoder in the JM reference software [14]. The first part, represented by intra-/inter-prediction, performs the intra- or interframe prediction considering one possible encoding mode. In the case of intra-frame prediction, this stage determines the predicted samples for one of all I16 MB and I4 MB modes presented in Figure 1 and Figure 2 and also for the chrominance blocks. In the case of inter-frame prediction, the most similar block is searched in the reference frames considering one of all possible partition and sub-partition sizes (P16 × 16, P16 × 8, P8 × 16, P8 × 8, SMB8 × 8, SMB8 × 4, SMB4 × 8, SMB4 × 4). After the intra-/interprediction, residual generation, transforms (T), quantization (Q), and entropy coding are applied, the bit rate for the prediction mode is calculated. The distortion is evaluated after the inverse transform (IT), the inverse quantization (IQ), and the addition of the residue to the reconstructed prediction samples for each mode. All grey blocks in Figure 5 are executed once for each possible intra- and intermodes, whilst the mode decision block (in white) receives all prediction modes' bit rates and distortions (dashed lines) generated in each iteration of the encoding loop, calculates the rate-distortion cost according to (1), and finally selects the mode with the lowest cost to be used in the encoding process.

It is possible to perceive that the RDO technique is extremely complex, making its use in encoders for real-time systems and high-resolution videos difficult, especially when portable devices are considered (since energy consumption is directly proportional to computational complexity). The use of fast algorithms and hardware architectures is thus of essential importance to reduce the mode decision complexity, increasing the encoder performance with negligible PSNR drop and small bit rate increase.

3. Previous Works

Recently, several works have proposed fast mode decision algorithms for the H.264/AVC encoding process. All of them aim at the reduction of the decision process complexity by decreasing the number of possible encoding modes considered in the RDO process [5–12].

In [5, 7, 9, 11] some heuristics are proposed to discard intraprediction modes for RDO calculation based on the transformed coefficients. The method proposed in [9] uses the low-frequency transform coefficients to detect image homogeneity, selects the best intrablock size, and then performs the selection of intramodes based on the RDO technique. In [11], the authors propose a SATD-based intra-frame decision in which the modes with high sum of absolute transformed differences (SATDs) values are not considered in the RDO process. In [5] the transformed coefficients of each block are analyzed in order to determine which modes would be evaluated by the RDO process. The work claims that each transform block tends to assume a formation pattern according to the encoding mode applied in the intra-prediction process. In [7], the SATD value is also used

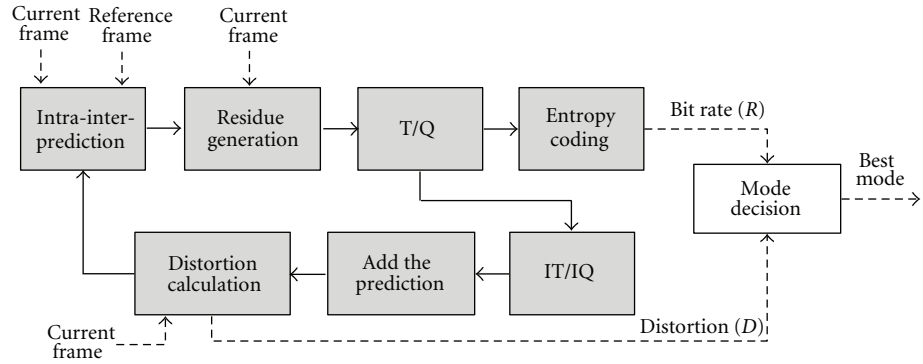

Figure 5: Diagram of the RDO-based encoding process.

to discard the less probable modes, as in [11]. However, the algorithm applies a bit rate estimation technique based on the number of coefficients quantized to zero after the quantization step. In [9], the low-frequency components of the transformed blocks are analyzed in order to identify the homogeneity of the original image, as proposed in this work. However, [9] is limited to decide only the best block size, so that the best 4×4 and the best 16×16 intraframe modes are still chosen through RDO iterations. As in [7, 11], the work [9] also eliminates probable modes by the analysis of SATD values.

The mode decision proposed in [6] chooses the best intra-frame mode based on the analysis of the directional gradient of each block. The two modes with the lowest gradient levels are selected and the RDO technique is applied to both of them and the DC mode. In [10], an intra-frame fast mode decision is performed based on the analysis of the motion field continuity. The Sobel operator is used in that work in order to detect borders and predict the spatial continuity of the objects.

Other works proposed heuristics for inter-frame mode decision. In [12], a set of heuristics for intra-frame and inter-frames mode decisions are proposed. The method is composed by stationarity detection through sum of absolute differences (SAD) calculation and border detection using the Sobel operator. When a specified condition is achieved, the RDO process is finalized and the encoding process occurs according to the best RDcost found so far. If no condition is achieved, all possible modes are still checked. The Sobel operator is also used in [10] to select larger or smaller partition blocks for inter-frames prediction. In [8], a fast mode decision algorithm based on the occurrence frequency of each prediction mode in P frames is proposed. The algorithm evaluates the performance of the most frequent modes before the less frequent modes. The method also applies a conditional evaluation of intra-frame modes, which happens only if the spatial correlation level is higher than the temporal correlation level.

Architectural designs based on heuristics have been also proposed in some works aiming at decreasing the encoding time for one MB [15–17]. In [15], a hardware design which applies a method similar to that in [6, 10] is proposed for the intramode decision. A filtering technique is used to perform the edge detection so that the number of modes is reduced from nine to four in I4 MB blocks. For I16 MB and chrominance block decisions, only the detected mode and the DC mode are selected to be evaluated by the RDO technique. In [16], a low-complexity intra mode decision algorithm based on a cost function composed by distortion calculation and bit rate estimation is implemented in a hardware architecture. In [17], a modified three-step algorithm performs the intra mode decision decreasing the number of I4 MB candidate modes (from nine to seven) to be evaluated by the RDO technique. Regarding the inter mode decision, to the best of the authors knowledge there are no published works yet which propose a hardware architecture for the inter mode decision independently from the implementation of a motion estimation module.

Even though all these works propose the use of some heuristics, they apply the RDO method in some stage to select prediction modes and block sizes and do not allow the achievement of significant reductions in terms of computational complexity compared with the fully RDO-based decision.

4. Fast Hierarchical Mode Decision

In this work, a fast hierarchical mode decision for the H.264/AVC standard targeting high-resolution video encoding is proposed. The goal of the method is to perform a heuristic decision which uses only the original frame and the reference frames information instead of performing the complete RDO process. The heuristic solutions and their respective architectures were all developed for low-complexity video encoding systems and are thus based on the baseline profile of H.264/AVC. They can be, however, easily incorporated into other profiles.

The hierarchical mode decision was divided into two steps: intra-prediction decision and inter-prediction decision. To reduce the computational complexity, we have considered that P slices support only P blocks (I blocks are not supported, as explained in the next subsections), so that the two proposed decision modules are independent from one another. This way, the first module is used only for I slices (generated only by intra prediction) whilst the second one is used only for P slices (generated only by inter prediction).

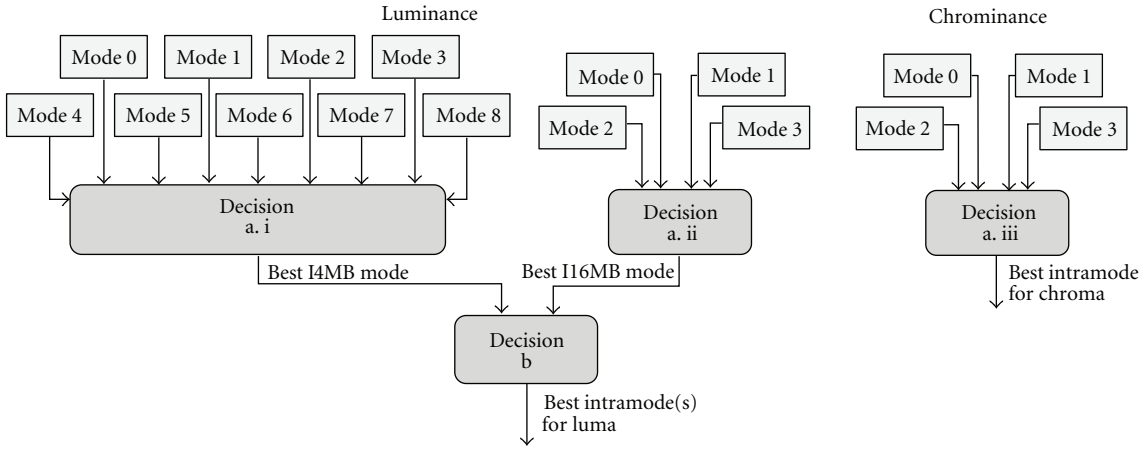

Figure 6: Hierarchical intra mode decision flow.

Even though B (bipredictive) slices are not supported in our work, the techniques proposed for P slices could be easily extended for B slices.

4.1. Fast Intramode Decision. The intra-frame mode decision was divided in a hierarchical way aiming at computational complexity decrease and modularization of the whole process. The steps which comprise intra-frame decision are as follows:

(a) decision for equal block size:

(i) decision of the best I4 MB mode for each 4×4 luminance block,

(ii) decision of the best I16 MB mode for the luminance channel of the macroblock,

(iii) decision of the best 8×8 mode for the chrominance channels of the macroblock,

(b) block-size decision for the luminance MB (I4 MB or I16 MB).

Each decision for equal block size (step (a)) is independent from one another and can be performed in parallel. The luminance block-size decision (step (b)) is performed after all decisions in step (a) are complete. Figure 6 presents the flow of decisions for the intra-frame prediction. As stressed by the orientation of the arrows in the flow, the decision process starts at the bottom of the hierarchy trees, which means that the best luminance I4 MB, I16 MB and chrominance 8×8 modes are decided before the final decision of the best luminance block size.

4.1.1. Intra Mode Decision for Equal Block Sizes. A distortion-based method was used to define which are the best modes for 4×4 luma block (I4 MB), the best mode for 16×16 luma block (I16 MB) and the best mode for 8×8 chroma block. These three steps were presented as *Decision* a.i, *Decision* a.ii, and *Decision* a.iii in the flow presented in Figure 6. The method consists simply in comparing the block prediction error of all candidate modes and choosing the mode with the lowest value as the best one.

Table 1: Comparison among SAD, SSD, and SATD.

SATD *versus* SAD			SSD *versus* SAD		
PSNR (dB)	Bit rate (%)	Number of Sums (%)	PSNR (dB)	Bit rate (%)	Number of Sums (%)
+0.008	−0.63	+361.29	+0.079	−1.13	+348.39

The block prediction error may be computed through the use of several distortion metrics. The use of sum of absolute differences (SADs), sum of absolute transformed differences (SATD) and sum of squared differences (SSDs) was evaluated [13]. Table 1 presents a comparison between the uses of the three distortion metrics in the mode decision module. All results are presented as relative increase or decrease in comparison to the use of SAD.

Even though SATD and SSD present slightly better rate-distortion performance results (around 1% bit rate decrease), the computational complexity of these metrics is extremely larger than the computational complexity of SAD (an increase around 350% for both metrics). As the main goal of this work is to design a faster and low-complexity intra mode decision heuristic, the SAD metric was used in the method proposed in this work. Besides that, a hardware architecture design for SAD calculators is much simpler than a hardware architecture for SATD (which includes a 4×4 Hadamard transform) and for SSD (which includes a multiplier and a square root).

4.1.2. Intramode Decision for Different Block Sizes. As previously explained, the second step of the intra mode decision process, shown as *Decision b* in Figure 6, consists in deciding the best macroblock size (I4 MB or I16 MB). In the proposed method, the choice between the use of 16 4×4 predicted blocks (I4 MB) or just one 16×16 predicted block (I16 MB) is made using the SAD results calculated in the first step of the mode decision.

The proposed algorithm picks the total distortion of the 16 4×4 blocks (I4 MB), which had all their best modes individually decided by *Decision* a.i (Figure 6) and compares it with the distortion of the unique 16×16 block (I16 MB), which had its best mode decided by *Decision* a.ii. In most cases, I4 MB would present the smallest distortion value, since a finer prediction granularity was used in it and more encoding modes are allowed for 4×4 blocks than for 16×16 blocks (Figures 1 and 2). This way, a simple SAD comparison cannot be used between them. On the other hand, when the differences between these two distortion values in RDO-based mode decision are analyzed, a good correlation between it and the best block size is perceived.

A set of simulations were performed with seven video sequences typically used by the video coding community (*Station 2, Sunflower, Tractor, Man in Car, River Bed, Rolling Tomatoes, Rush Hour*) using the *Joint Model* (JM) H.264/AVC reference software set in full-RDO-based decision and intra-only MB modes [18]. It was possible to notice that in most cases when I16 MB had been chosen, the distortion values of I4 MB and I16 MB were very close. This happens because in such cases most of the 16 4×4 blocks inside an I4 MB presented the same prediction modes, and thus only one 16×16 block for the whole MB can be used (i.e., I16 MB is chosen), since it generates less encoding information. On the other hand, when an I4 MB is chosen, the distortion values of I4 MB and I16 MB were very different. This happens because most of the 16 4×4 blocks inside a I4 MB presented different modes, and then choosing only one 16×16 mode would generate a lot of residual information.

To better evaluate the relation presented in the previous paragraph, (2) was defined. This equation shows the difference of distortion (DD) calculation, where $\mathrm{SAD}_{\mathrm{I4MB}}$ represents the sum of all residual generated by the 16 best 4×4 modes and $\mathrm{SAD}_{\mathrm{I16\,MB}}$ is the residual generated by the best 16×16 mode

$$\mathrm{DD} = \mathrm{SAD}_{\mathrm{I4\,MB}} - \mathrm{SAD}_{\mathrm{I16\,MB}}. \quad (2)$$

Figure 7 shows a graph in which the occurrence frequency of each block size chosen by the RDO-based mode decision is plotted with the corresponding DD calculated as in (2). It is possible to perceive that in most cases when I16 MB is selected (grey line) the DD value is very small. Around 97% of the macroblocks encoded as I16 MB presented a DD smaller than 600. On the other hand, when I4 MB is chosen by the RDO-based mode decision, the DD value is generally very large. Around 84% of the macroblocks presented a DD value larger than 600. This way, a threshold value for the DD value can be used to choose the best block-size for intra prediction with a small error regarding the decision.

The best threshold value was experimentally chosen aiming at the best results in terms of PSNR and bit rate, as follows. Firstly, all video sequences used in the simulations were encoded using the RDO-based mode decision for different block sizes. Meanwhile, distortion values and chosen block sizes were saved for future analysis. Then, the DD values were calculated and compared with the chosen block size, as in Figure 7. Simulations were performed with a threshold ranging from 0 to 1000, adjusting it to improve bit rate and video quality. The threshold value set to 600 generated the best results in terms of bit rate and video quality for the evaluated sequences.

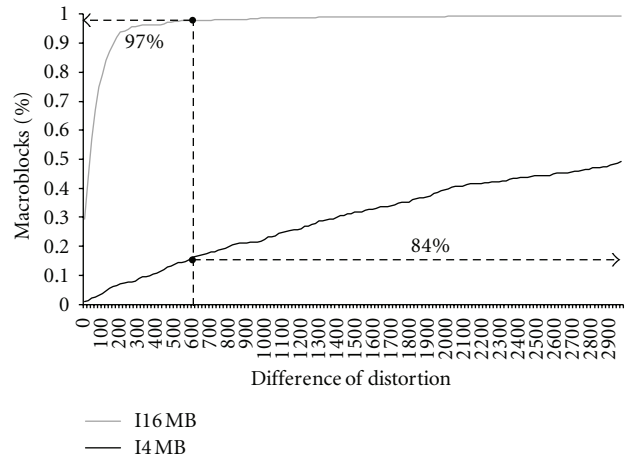

Figure 7: Occurrence of I4 MB and I16 MB and corresponding difference of distortion (DD) values.

4.2. Fast Inter Mode Decision. As in the intra-frame mode decision process, the inter-frames decision was hierarchically divided, aiming at the simplification of the problem in simpler and independent steps. As previously mentioned, the first decision to reduce the inter prediction decision complexity was the elimination of I blocks support into P slices. This way, only P blocks generated through the inter-frames prediction are allowed in such slices. The impacts of this simplification were measured through tests using the reference software and the previously cited video sequences. A reduction of only 0.01 dB in video quality and an increase of only 0.39% in bit rate were noticed when P slices were restricted to inter predicted blocks.

Previous analyses of the occurrence frequency for each mode in P slices were performed in order to identify which mode should be evaluated first. The same eight video sequences related above were used in these analyses. The obtained results show that *SKIP* and P 16×16 macroblocks represent almost 60% of the whole set of used modes in 1080 p (1920×1080 pixels) video sequences, while the use of sub-macroblock partitions smaller than 8×8 represents about 6% of occurrence, as shown in Table 2. Other important result of Table 2 is that intra modes are rarely used (3.8%) and, as cited before, the elimination of these modes in P slices causes a very small reduction in rate-distortion performance. Similar tests were performed for 720 p (1280×720 pixels) video sequences, which presented a small increase in the occurrence of small block sizes (4×4, 4×8 and 8×4). As this work is focused on high-resolution video sequences, only results for 1080p sequences are presented in Table 2.

It is possible to conclude from Table 2 that the partition/subpartition occurrence is higher when the block size

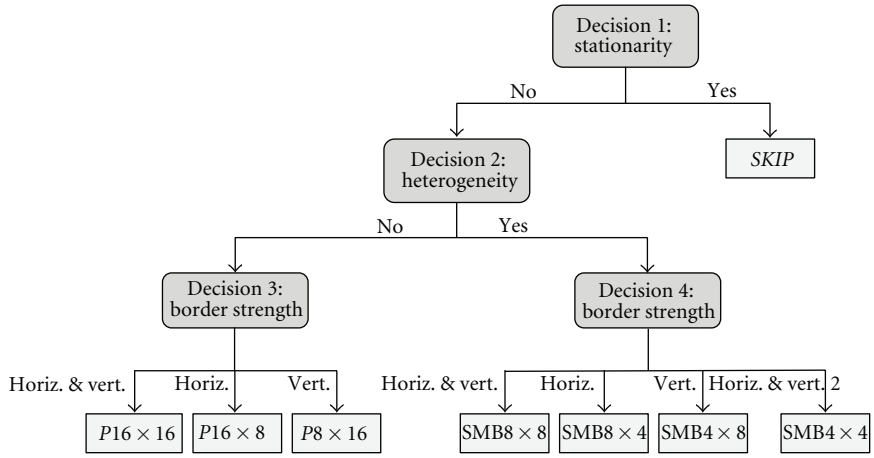

Figure 8: Hierarchical mode decision flow for inter-prediction decisions.

Table 2: Average occurrence and frequency for each mode in P slices.

Mode	Occurrence	Frequency
SKIP	83,645	29.1%
*P*16 × 16	83,651	29.1%
*P*16 × 8	30,364	10.6%
*P*8 × 16	33,187	11.5%
*SMB*8 × 8	27,251	9.5%
*SMB*8 × 4	7,700	2.7%
*SMB*4 × 8	8,472	2.9%
*SMB*4 × 4	2,331	0.8%
Intramodes	10,895	3.8%

is larger especially for large video resolutions. These results confirm the high-definition digital video characteristics of containing a large amount of homogeneous and stationary areas. Based on this occurrence order, a sequence of steps was developed for fast inter mode decision. This sequence prioritizes the identification of the most frequent modes before the least frequent, thus decreasing the average number of performed decision steps.

Figure 8 shows the inter-frames hierarchical mode decision flow proposed in this work. The first performed inter mode decision step is at the top of the tree, as indicated by the arrows direction. In the first step, an analysis of the original and the reference macroblocks is performed in order to allow stationarity detection between two neighbor frames. The second step, based on heterogeneity detection, decides whether or not subpartitioning is necessary. The third and the fourth decisions are quite similar and are based on gradient calculation to detect object borders. The next subsections describe each step in details.

4.2.1. Stationarity-Based SKIP Mode Decision. Natural video sequences present the characteristic of being composed by constant motion and by temporally stationary regions. This characteristic is highly noticeable especially in background images [12]. In the JM reference software, when the RDO-based decision is used, the *SKIP* mode is selected when the cost of not encoding residual information is smaller than the costs of all the other possible modes. Otherwise, a set of four conditions must be satisfied for a macroblock to be encoded with the *SKIP* mode. In [19], the authors affirm through experimental results that the satisfaction of one of them is already enough for a good indication of *SKIP* mode usage. Such condition dictates that the transform coefficients of the *SKIP* mode residue block are all quantized to zero.

Stationarity, also called stillness, is a simple and efficient way of identifying the similarity level of each frame and its neighbor frame in a video. In this work, the simple distortion calculation between a macroblock and its colocated macroblock is proposed to detect stationarity. If the SAD between the two macroblocks is smaller than a pre-defined threshold, the image region is considered temporally stationary, and this region is thus encoded as *SKIP*.

A series of tests were performed in order to verify the threshold SAD value between neighbor frames when *SKIP* and non-*SKIP* macroblocks were selected by the RDO-based decision. Average values obtained for both cases were used in tests which varied the SAD threshold for *SKIP* decision from 200 to 700. The best results in terms of rate-distortion performance were obtained when the value 500 was used. This means that the proposed *SKIP* decision consists in computing the SAD between co-located macroblocks in the reference and the current frame and then comparing this value with the threshold (500). If the SAD is smaller than it, the macroblock is considered stationary and it is encoded as *SKIP*. Otherwise, the macroblock is not considered stationary and it is encoded with another mode.

As the heuristic is a suboptimal solution of the RDO decision, it may select more or less *SKIP* macroblocks, depending on the video sequence. The average results have shown that, in general, more *SKIP* macroblocks were selected than in the RDO process. This caused an increase in the bit rate and also in the image quality when this heuristic was inserted in the reference software. An average increase of 0.04 dB was noticed in the PSNR when this heuristic

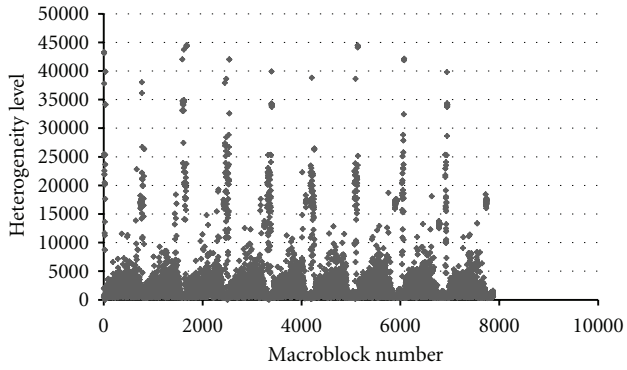

Figure 9: Heterogeneity level (H) for each macroblock encoded without sub-partitions in the sequence *Man in Car* (RDO-based decision).

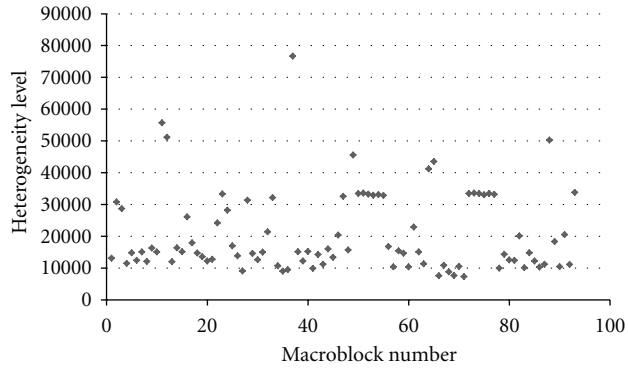

Figure 10: Heterogeneity level (H) for each macroblock encoded with sub-partitions in the sequence *Man in Car* (RDO-based decision).

was applied. On the other hand, the bit rate presented an increase of 1.75% when compared to the RDO-based decision.

4.2.2. Heterogeneity-Based SubPartition Detection. Natural video sequences are also composed by a large amount of homogeneous regions. According to [20], these regions generally present similar directional motion, while heterogeneous regions present disordered movements. This happens because homogeneous areas tend to belong to stationary regions of an image or tend to belong to regions of the same object of an image, which is composed by parts that move together and continuously in the same direction. This way, homogeneous regions are, in general, encoded using large block sizes, such as 16×16, 16×8, and 8×16, while heterogeneous regions are encoded using smaller block sizes.

We have shown in a previous work that the heterogeneity of a macroblock can be detected through the analysis of some coefficients generated after the application of a forward transform, such as the Hadamard and the DCT [21, 22]. The heterogeneity H of a macroblock is thus calculated from a set of low-frequency coefficients from the transformed block. In [21], we propose to use only the first line and the first column of the transformed coefficients in the heterogeneity calculation, as shown in (3), where Y is the transformed macroblock and u and v are the macroblock lines and columns, respectively. The higher is the value of H, the higher is the heterogeneity of the macroblock

$$H = \sum_{u=1}^{15} |Y(u,0)| + \sum_{v=1}^{15} |Y(0,v)|. \quad (3)$$

A set of experimental tests led to a threshold value which can be used to determine if subpartitions are used or not. If the heterogeneity of the macroblock is high (i.e., the H value is higher than the threshold), sub-partitions should be used. Otherwise, only large partitions should be used. Figures 9 and 10 show graphs in which each point represents a macroblock of the *Man in Car* video sequence. In Figure 9, each point is a macroblock encoded as P16 $\times$ 16, P16 $\times$ 8 or P8 $\times$ 16 (i.e., without sub-partitions) and in Figure 10 each point is a macroblock encoded with sub-partitions. The decisions were all performed by the RDO process. It is possible to notice that most of the macroblocks encoded without sub-partitions present an H value smaller than 10,000, while most of the macroblocks encoded with sub-partitions present an H value larger than 10,000. As this behavior is presented in most of the analyzed videos, the value 10,000 was defined to be used in this heuristic as the threshold between the use of large partitions and the use of sub-partitions. Experimental results have shown that an increase of only 0.62% in the bit rate and a negligible image quality loss (around 0.01 dB) were perceived when this threshold was used in such decision step.

4.2.3. Border Detection for Partition Decision. The second decision step presented in the last sub-section decides whether the third or the fourth decision step takes place. The third step decides which partition size is used, and it is based on the difference between pixels that compose the lines and the columns of a macroblock, which is called the gradient. As previously explained, in natural video sequences all the parts of an object tend to move together in the same direction and they are probably grouped inside the same partition. This way, the detection of object edges around possible partition edges allows the definition of which is the best partition size for the macroblock. In the proposed decision method, different objects are considered to have different pixel values, whilst different regions of the same object present a higher probability of being composed by similar pixel values. Based on this, different objects can be distinguished with simple comparisons between neighboring pixels. Even though this is a generalization which incurs in some decision error, bit rate and image quality are not so strongly affected, as shown below.

The proposed edge detection heuristic is based on simple subtractions between neighboring pixels located around possible partition borders. In Figure 11, vertical and horizontal partition borders are shown as bold lines and the pixels used in the border calculation are shown in gray. Eight pixels are used from each line (or column) in the gradient calculation. Equation (4) Defines the summation performed

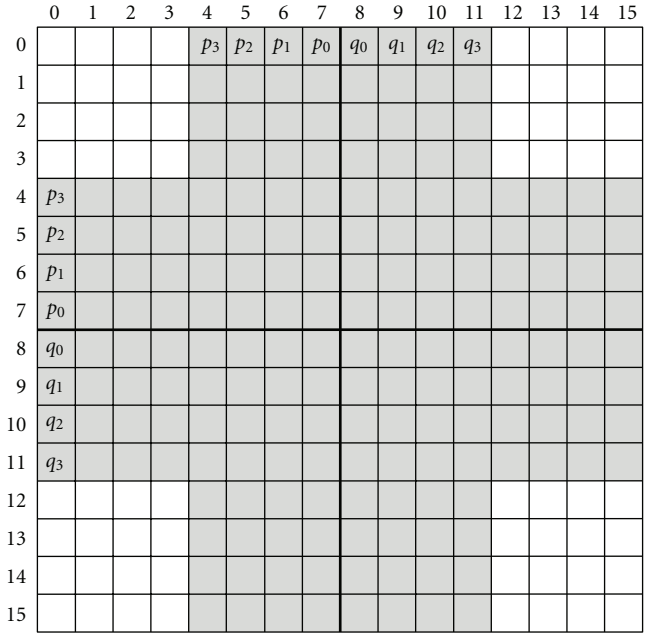

Figure 11: Pixels used in the gradient calculation for partition size decision.

to determine the vertical border strength (VB) and (5) defines the horizontal border strength calculation (HB), where O is the pixel matrix from the original macroblock, i is the macroblock line index, and j is the macroblock column index. As shown in the equations, pixels are compared two by two (i.e., *p3* is compared with *q3*, *p2* is compared with *q2,* and so on), which allows that both abrupt and gradual pixel changes are found

$$\mathrm{VB} = \sum_{i=0}^{15} \sum_{j=4}^{7} \left(O_{i,j} - O_{i,(15-j)} \right), \tag{4}$$

$$\mathrm{HB} = \sum_{j=0}^{15} \sum_{i=4}^{7} \left(O_{i,j} - O_{(15-i),j} \right). \tag{5}$$

After computing the horizontal border (HB) and the vertical border (VB) values, a set of comparisons is performed to determine the best macroblock partition. A set of tests has shown that a threshold value set in 80 for the difference between horizontal and vertical borders leads to the best results in terms of rate-distortion performance. This way, Algorithm 1 was used to decide the best partition size.

Experimental results have shown that a bit rate increase of 3.59% and a PSNR increase of 0.02 dB were noticed when this method was used instead of the RDO-based decision. As previously explained, this is a sub-optimal solution of the RDO method and therefore it eventually performs different decisions, which leads to a different configuration of partitions (and thus a larger or smaller bit rate and PSNR difference) depending on the video sequence.

4.2.4. Border Detection for Subpartition Decision. The fourth decision step is similar to the third one, though it analyzes more than two borders. Figure 12 presents the structure of a macroblock divided in four smaller structures, the 8 × 8 sub-macroblocks. For each sub-macroblock, the vertical and horizontal border calculations are performed and its results are compared in order to decide the sub-partition type. Differently from the border calculation for partitions, sub-partition decision calculations are performed over four samples instead of eight. Besides, in order to allow identifying 4×4 partitions, each border is calculated in two parts: the top (or left) subborder and the bottom (or right) sub-border.

Equations (6)–(11) present the calculations for vertical and horizontal sub-partition borders of the first sub-macroblock. The vertical and the horizontal sub-partition borders are calculated as in (6) and (7), where VPB is the vertical sub-partition border, HPB is the horizontal sub-partition border, and VSB1, VSB2, HSB1, and HSB2 are the halves of each border (called sub-borders). Equations (8), (9), (10), and Equation (11) present the calculation of each vertical and horizontal sub-border, respectively, where O is the pixel matrix from the original macroblock, i is the macroblock line index, j is the macroblock column index, VSB1 *and* VSB2 are the first and second halves of the vertical sub-border, respectively, and HSB1 and HSB2 are the first and second halves of the horizontal sub-border, respectively

$$\mathrm{VPB} = \mathrm{VSB1} + \mathrm{VSB2}, \tag{6}$$

$$\mathrm{HPB} = \mathrm{HSB1} + \mathrm{HSB2}, \tag{7}$$

$$\mathrm{VSB1} = \sum_{i=0}^{3} \sum_{j=2}^{3} \left(O_{i,j} - O_{i,(7-j)} \right), \tag{8}$$

$$\mathrm{VSB2} = \sum_{i=4}^{7} \sum_{j=2}^{3} \left(O_{i,j} - O_{i,(7-j)} \right), \tag{9}$$

$$\mathrm{HSB1} = \sum_{j=0}^{3} \sum_{i=2}^{3} \left(O_{i,j} - O_{(7-i),j} \right), \tag{10}$$

$$\mathrm{HSB2} = \sum_{j=4}^{7} \sum_{i=2}^{3} \left(O_{i,j} - O_{(7-i),j} \right). \tag{11}$$

Border values are compared to previously defined thresholds, according to Algorithm 2. As in all previously shown heuristics, the thresholds were obtained through exhaustive experimental tests which took into consideration the rate-distortion performance. The proposed sub-partition decision method led to an average bit rate increase of only 0.34% and a PSNR increase of 0.02 dB when the thresholds 40 and 20 were used for sub-partition border and sub-partition sub-border, respectively.

```
If |HB-VB| ≤ 80 Then
  best_INTER_mode = P16×16
Else If HB−VB > 80 Then
  best_INTER_mode = P16×8
  // (macroblock is divided into two 16×8 partitions)
Else If VB−HB > 80 Then
  best_INTER_mode = P8×16
  // (macroblock is divided into two 8×16 partitions)
```

Algorithm 1: Algorithm used to define the partition size.

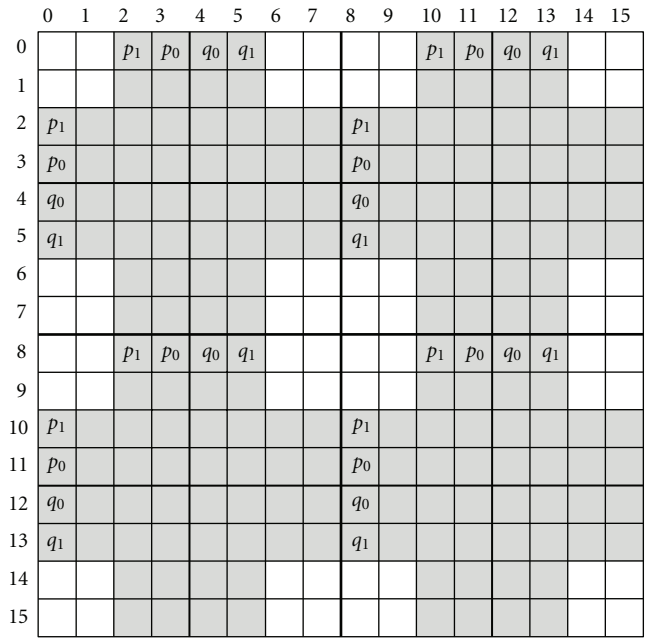

Figure 12: Pixels used in the gradient calculation for sub-partition size decision.

```
If |HPB-VPB| ≤ 40 Then
  best_INTER_mode = SMB8×8
  // (sub-macroblock is not divided into sub-partitions)
Else If HPB−VPB > 40 Then
    If VSB1 > 20 OR VSB2 > 20 Then
        best_INTER_mode = SMB4×4
      // (sub-MB is divided into four 4×4 sub-partitions)
    Else
        best_INTER_mode = SMB8×4
      // (sub-MB is divided into two 8x4 sub-partitions)
Else If VPB − HPB > 40 Then
    If HSB1 > 20 OR HSB2 > 20 Then
        best_INTER_mode = SMB4×4
      // (sub-MB is divided into four 4×4 sub-partitions)
    Else
        best_INTER_mode = SMB4×8
      // (sub-MB is divided into two 4×8 blocks)
```

Algorithm 2: Algorithm used to define the sub-partition size.

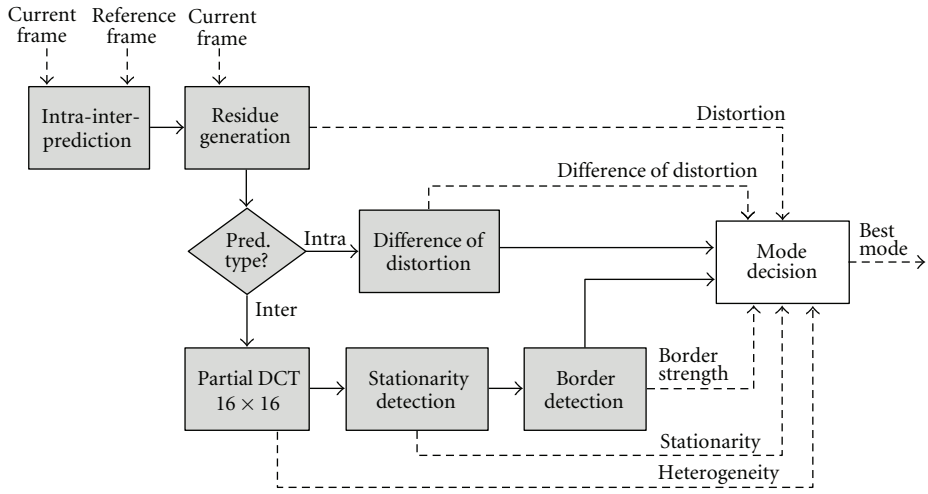

Figure 13: Block diagram of the proposed fast hierarchical mode decision for H.264/AVC.

4.3. Final Results of the Proposed Heuristics. The fast decision heuristics were evaluated together and compared with the RDO-based mode decision. This comparison considered the computational cost reduction and the PSNR and compression rates losses. The main results are presented in the next sub-sections.

4.3.1. Computational Complexity Results. As presented in Figure 5, the mode decision in the RDO-based encoding process is finished only after the execution of all possible intra- and inter-prediction candidate modes (i.e., after successive iterations of the encoding loop). Figure 13 presents a block diagram of our method in which all operations to perform mode decision are shown. The loop from Figure 5 is completely eliminated, significantly simplifying the decision process. Instead of performing the whole encoding process for all possible candidate modes, our method performs the mode decision right after the prediction residual generation and the difference of distortion calculation (when intra prediction is used) or after the prediction residual generation, the partial 16×16 DCT calculation, the stationarity detection, and the border detection (when inter prediction is used).

Considering only luminance mode decision, an analysis of the complexity reduction is done in the next paragraphs. When the RDO technique is applied for intra-frame mode decision, four 16×16 and nine 4×4 intra modes must be compared. This means that the decision of the best mode for a macroblock in an I slice occurs only after the evaluation of all the four prediction modes for the macroblock and all the nine prediction modes for 4×4 blocks (i.e., for each one of the 16 blocks which compose the macroblock). In terms of computational complexity, it means that the loop presented in Figure 5 is repeated four times for 16×16 intra-frame prediction and 144 times for 4×4 intra-frame prediction, totalizing 148 iterations in the encoding flow.

The inter-frames mode decision in P slices is performed after the evaluation of the five partition modes (*SKIP*, $P16 \times 16$, $P16 \times 8$, $P8 \times 16$ and $P8 \times 8$) for each macroblock and the four sub-partition modes for each one of the four 8×8 sub-macroblocks which compose a macroblock. In other words, 5 iterations of the loop presented in Figure 5 are executed for the macroblock modes and 16 iterations are executed for the sub-macroblock modes, totalizing 21 iterations. Besides, as the H.264/AVC standard defines that P slices can also be composed by intra-frame macroblocks, all the 148 iterations for intra-frame modes are also performed, totalizing 168 encoding repetitions per macroblock.

Considering the proposed fast hierarchical mode decision and its execution flow presented in Figure 13, the best mode decision of an I frame is performed right after the prediction residual generation and the difference of distortion calculation. It means that if the difference of distortion detects that the best macroblock partitioning is I16 MB, the encoding flow is executed only once considering the chosen 16×16 mode. Otherwise, the loop is executed 16 times, once for each 4×4 block in I4 MB. For inter-frames macroblocks in P slices, the best mode decision happens after the heterogeneity evaluation, the stationarity analysis, and the border detection calculations. The remaining encoding steps which are now outside the encoding loop (transform, quantization, inverse transform, inverse quantization, entropy coding, filtering) are then executed only once, considering the selected partition and the selected sub-partitions, if necessary.

Based on this complexity analysis and taking into consideration, a group of pictures (GOP) composed by one I frame and five P frames (IPPPPP) in an HD 1080p resolution video (8160 macroblocks per frame), the RDO-based encoding process would perform 8,062,080 iterations in the loop presented in Figure 5. On the other hand, the fast hierarchical mode decision performs 171,360 executions of the encoding flow to complete the encoding process considering the same GOP. This means a reduction of 47 times in the number of encoder iterations, which is a very high gain and justifies the small bit rate increase generated by these techniques, which is discussed in the next sub-section.

Table 3: Bit rate and PSNR for fast intra-frame mode decision and RDO-based mode decision.

Video Sequence	RDO decision		Fast intra frame mode decision		RDO × fast intra frame mode decision	
	PSNR (dB)	Number of Bits (Kbits)	PSNR (dB)	Number of Bits (Kbits)	ΔPSNR (dB)	Δbits (%)
Station 2	39.64	34,022	39.33	35,729	−0.30	+5.02
Sunflower	42.77	33,030	42.41	35,391	−0.36	+7.15
Tractor	39.40	58,208	39.04	60,939	−0.37	+4.69
Man In car	42.61	8,818	42.52	9,018	−0.09	+2.27
River Bed	38.65	56,735	38.28	58,962	−0.37	+3.93
Rolling Tomatoes	40.48	12,170	40.39	12,537	−0.09	+3.02
Rush Hour	41.65	20,013	41.48	21,306	−0.17	+6.46
Average	**40.74**	**31,857**	**40.49**	**33,412**	**−0.25**	**+4.88**

Table 4: Bit rate and PSNR for fast inter-frames mode decision and RDO-based mode decision.

Video sequence	RDO decision		Fast inter frame mode Decision		RDO × Fast Inter frame mode Decision	
	PSNR (dB)	Number of Bits (Kbits)	PSNR (dB)	Number of Bits (Kbits)	ΔPSNR (dB)	Δbits (%)
Station 2	39.64	4407	39.60	4630	−0.04	+5.06
Sunflower	42.58	7105	42.59	7636	+0.01	+7.47
Tractor	38.92	11344	38.87	11897	−0.05	+4.87
Man in car	42.43	2138	42.40	2171	−0.03	+1.54
River bed	40.36	3427	40.32	3740	−0.04	+9.13
Rolling tomatoes	41.47	3711	41.40	4066	−0.07	+9.57
Rush hour	40.44	6933	40.38	7596	−0.06	+9.56
Average	**40.83**	**5581**	**40.79**	**5962**	**−0.04**	**+6.84**

4.3.2. Rate-Distortion Performance Results. Evaluations were performed considering all the heuristics proposed in this work, which were integrated to the H.264/AVC reference software. All evaluations were performed for HD1080p (1920×1080 pixels) video sequences with groups of pictures (GOP) composed by one I frame followed by three P frames (IPPP), encoded with QPs 22, 27, 32, and 37. Tables 3 and 4 present average results for the proposed intra-frame mode decision and inter-frames mode decision, respectively. The first columns of each table present the results obtained through the application of the fully RDO-based decision, the central columns present the results obtained through the proposed method, and the final columns present a comparison between the two approaches in terms of image quality losses (ΔPSNR, in dB) and bit rate increase (Δbits).

An average increase of 4.88% in bit rate and an average PSNR loss of 0.25 dB were noticed when the proposed intra-frame mode decision is used, in comparison to the RDO-based mode decision. When the proposed inter-frames mode decision is used, the average bit rate increase is around 6.84% and the average PSNR drop is under 0.04 dB. Even though such rate-distortion performance losses are not desirable, they are not significant when the computational complexity reductions provided by the fast decision method are taken into consideration.

5. Hardware Design for the Proposed Mode Decision Heuristics

Hardware architectures were designed considering the heuristics presented in the previous section. As previously stated, the main goal of this work is to design hardware architectures for an H.264/AVC complete mode decision module which is able to process high-resolution video (such as HD1080p) in real time (30 frames per second).

5.1. Intra-Frame Mode Decision Architecture. The architecture for the intra-frame mode decision module was designed according to the hierarchical algorithm presented in Section 4.1. The architecture was designed considering the same hierarchical flow used in the algorithm itself. Two sub-modules are thus used in the architecture: (a) decision for equal block size and (b) decision for different block sizes. As previously explained, the decision for equal block sizes was designed using the SAD metric. The architecture designed to calculate the SAD distortion metric is presented in Figure 14, where O represents the original block and P represents the predicted block.

The SAD architecture performs the calculation of eight samples per clock cycle (two lines of a 4×4 block). Registers were used to decrease the critical path of the SAD calculator by generating a two-stage pipeline. The output register is responsible to accumulate partial SAD calculations, since only eight samples are processed per clock cycle. This way, a new complete SAD value is available at the output register after each two clock cycles. As the SAD calculation was used in all algorithms proposed in this work, the architecture presented in Figure 14 is the basic module in all presented architectures that use SAD. Figure 15 shows the block diagram of the whole intra-frame mode decision architecture.

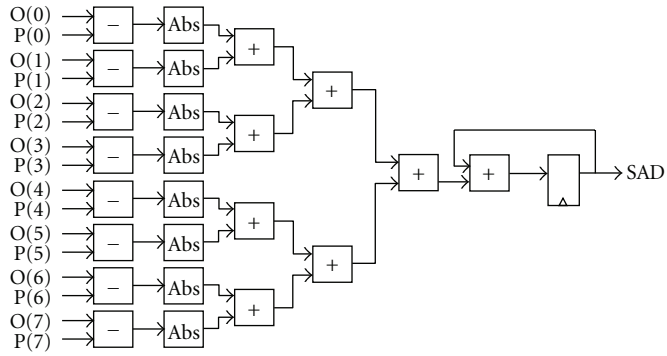

Figure 14: RTL diagram of the SAD calculator.

The architecture presented in Figure 15 is composed by 17 SAD calculators (nine for I4 MB mode decision, four for I16 MB mode decision, and four for chrominance block mode decision), three comparators which perform the decision between modes with same block sizes and the difference of distortion decision.

Distortion calculation for I4 MB mode decision is performed for the nine candidate modes (*SAD 0* to *SAD 8* in *I4 MB Decision* module in Figure 15). To allow the further decision between different block sizes (4×4 or 8×8), all the nine computed SAD values of one block are compared and the smallest one is summed up in the I4 MB *Accumulator* module. After the 16 smallest 4×4 SAD values belonging to one macroblock are computed, they are accumulated and stored before the second intra-frame decision step starts. In order to decide the best I16 MB mode, the four SAD values (*SAD 0* to *SAD* 3 in the I16 MB *Decision* module) are compared and the smallest one is chosen by the comparator. No accumulator is necessary in this case, since only one 16×16 block composes the I16 MB. The chrominance mode decision selects the smallest SAD among the four chrominance prediction modes. The final decision of which is the best mode is performed in just one clock cycle and no other operation is necessary.

The *difference of distortion* module calculates the difference between the best accumulated I4 MB results and the best I16 MB result. The difference is then compared with the threshold value defined in Section 4.1 in order to perform the final intra-frame mode decision.

The intra-frame mode decision operation schedule is shown in Figure 16, where the number of clock cycles spent by each module is presented. Nonlabeled spaces correspond to periods in which the modules are not operational or their output results are not used.

As mentioned before, the SAD calculator was implemented with a two-stage pipeline and with eight input samples (two lines of a 4×4 block). This way, the architecture takes three clock cycles to deliver the first valid SAD considering a complete 4×4 block. When the pipeline is filled, a valid SAD result is available at the output at every two clock cycles. The architecture takes, therefore, 34 clock cycles to evaluate all nine I4 MB candidate modes and one more cycle to accumulate the last SAD result. Considering I16 MB mode decision, the SAD values are accumulated in the SAD calculator itself (the output register in Figure 14). This way, the architecture takes 33 clock cycles (one less than the I4 MB mode decision) to accumulate the SAD for the four modes and one more cycle to compare them. The chrominance decision is similar to the I16 MB decision. However, as the block is composed only of 8×8 samples, the module takes only 10 clock cycles to perform the decision. Finally, when the distortion values for the best I4 MB modes and the best I16 MB mode are ready, the difference of distortion module evaluates these results in one clock cycle to decide which block size must be used. Then, the complete intra-frame decision architecture, in the worst case, uses 36 clock cycles to define which are the best prediction modes and the best block size.

5.2. Inter-Frames Mode Decision Architecture. The architecture for inter-frames mode decision was designed according to the hierarchical algorithm presented in Section 4.2. As the algorithm steps are not dependent on one another, all operations can be executed in parallel when a hardware design is considered. This way, in order to increase the architecture performance, the stationarity calculation for *SKIP* mode decision, the heterogeneity detection, the partition size decision and the sub-partition size decision are calculated concurrently in the proposed hardware architecture. It is important to notice that although all the decisions steps are always executed, some results may not be used since the inter-frames decision algorithm is hierarchical. For example, if the first decision step decides that the macroblock should be encoded as a *SKIP* macroblock, then all the other performed calculations are unnecessary and the results are discarded.

The inter-frames mode decision architecture is divided into the following modules: (1) stationarity-based *SKIP* mode decision, (2) partition/sub-partition detection, and (3) partition and sub-partition sizes decision.

5.2.1. Stationarity-Based SKIP Mode Decision Architecture. The *SKIP* mode decision is performed comparing the 16×16 macroblock in the current frame with the co-located macroblock in the reference frame. This comparison is performed using the SAD metric (calculated between current and co-located macroblock). This way, an instance of the architecture used for SAD calculations (see Figure 14) was also used to perform the *SKIP* mode decision. The SAD

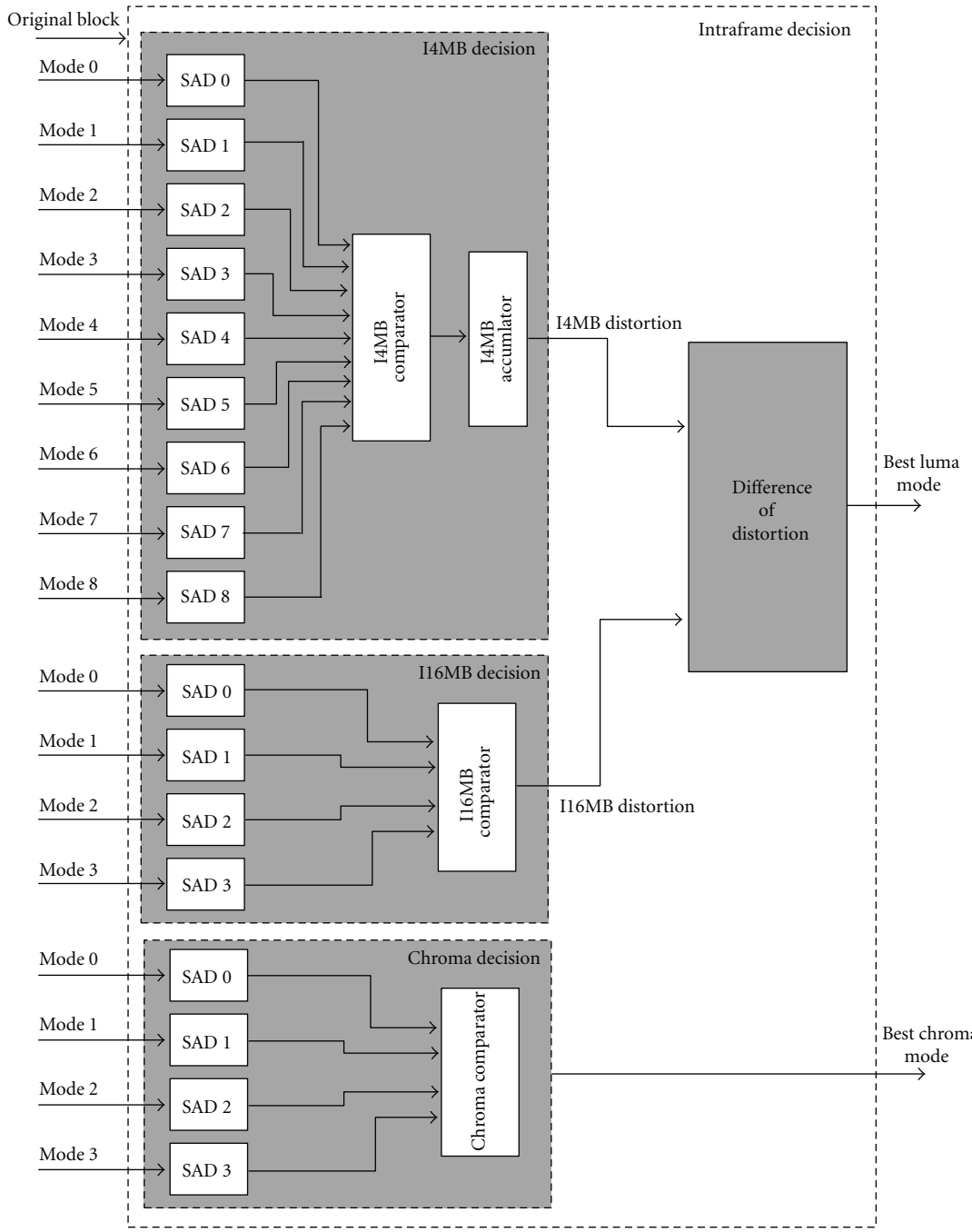

Figure 15: Block diagram of the intra-frame mode decision architecture.

calculation for a 16×16 macroblock consumes 32 clock cycles, since eight samples are processed in parallel (two lines of samples of a 4×4 block). One extra cycle is necessary to compare the SAD result with the threshold defined in Section 4.2.1 in order to define whether or not the macroblock should be encoded as *SKIP*. In the proposed architecture, this extra cycle is spent in the last step of the global inter-frames decision.

5.2.2. Heterogeneity-Based Subpartition Detection Architecture. As previously explained, the sub-partition detection is performed through a heterogeneity metric, which is calculated using a DCT transform over the original 16×16 block (the current macroblock). As only some coefficients are needed to detect the heterogeneity level (top line and left column of coefficients), a partial 16×16 DCT architecture was designed in which unnecessary coefficients, not in the top and left column, are ignored. Besides the reduction in the number of coefficients, the DCT matrix was simplified to work with power-of-two operations. This is a very useful adaptation when hardware architectures are considered, since multiplications and divisions can be all replaced by

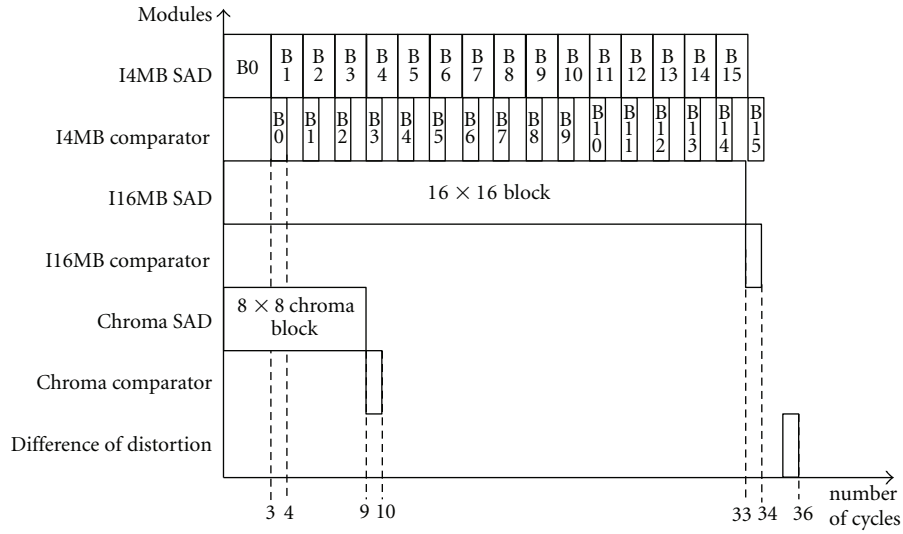

Figure 16: Schedule diagram of the intra-frame decision architecture.

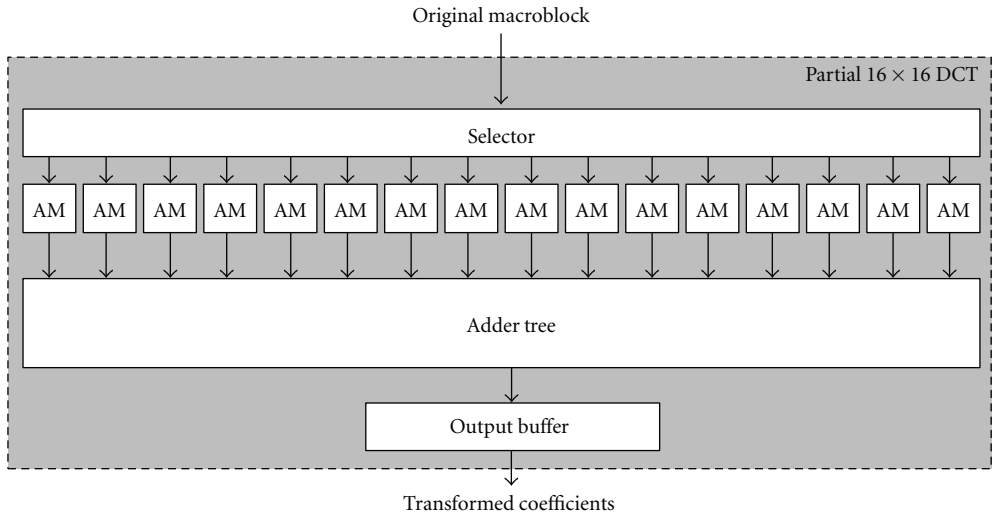

Figure 17: Block diagram of the partial 16 × 16 DCT.

simple shift operations (which introduce no hardware cost). Then, the DCT architecture implemented in this work is composed only by adders, subtractors, and shifters. Some coefficient calculations are very similar, so that it was also possible to reuse several operands and partial results. Figure 17 shows the block diagram of the architecture which generates all coefficients used by the heterogeneity metric.

The architecture was designed with a parallelism of 256 samples (i.e., the complete 16 × 16 original block must be available in the architecture input). However, as memory bandwidth is a common issue in such kind of systems, an input buffer was designed to group all samples before the architecture processing. In Figure 17, the *Selector* module is responsible for reordering the input samples and for delivering them to the *Adder Modules* (AM) input according to the calculation pattern. The AM modules perform all partial calculations which can be used in the calculation of more than one coefficient. Each AM calculator was implemented with four pipeline stages in order to achieve the throughput requirements for real-time processing. The *adder tree* module performs the final calculations for each coefficient summing up the combined results from the AM modules and delivers the final coefficient results to the *output buffer*. Four clock cycles are needed to deliver one valid result from the *adder tree* to the *output buffer*. Finally, the buffer reorders the computed coefficients and delivers them to the final decision module.

5.2.3. Border Detection for Partition/Sub-Partition Decision Architecture. The architecture designed to calculate the border strength and to determine the partition and sub-partition sizes are similar to the SAD calculator used before. However, samples from the original block are used instead of the prediction residual. Figure 18 presents the architecture

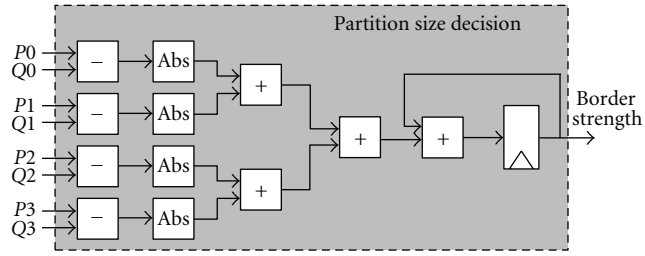

Figure 18: Architecture for partition border strength calculation.

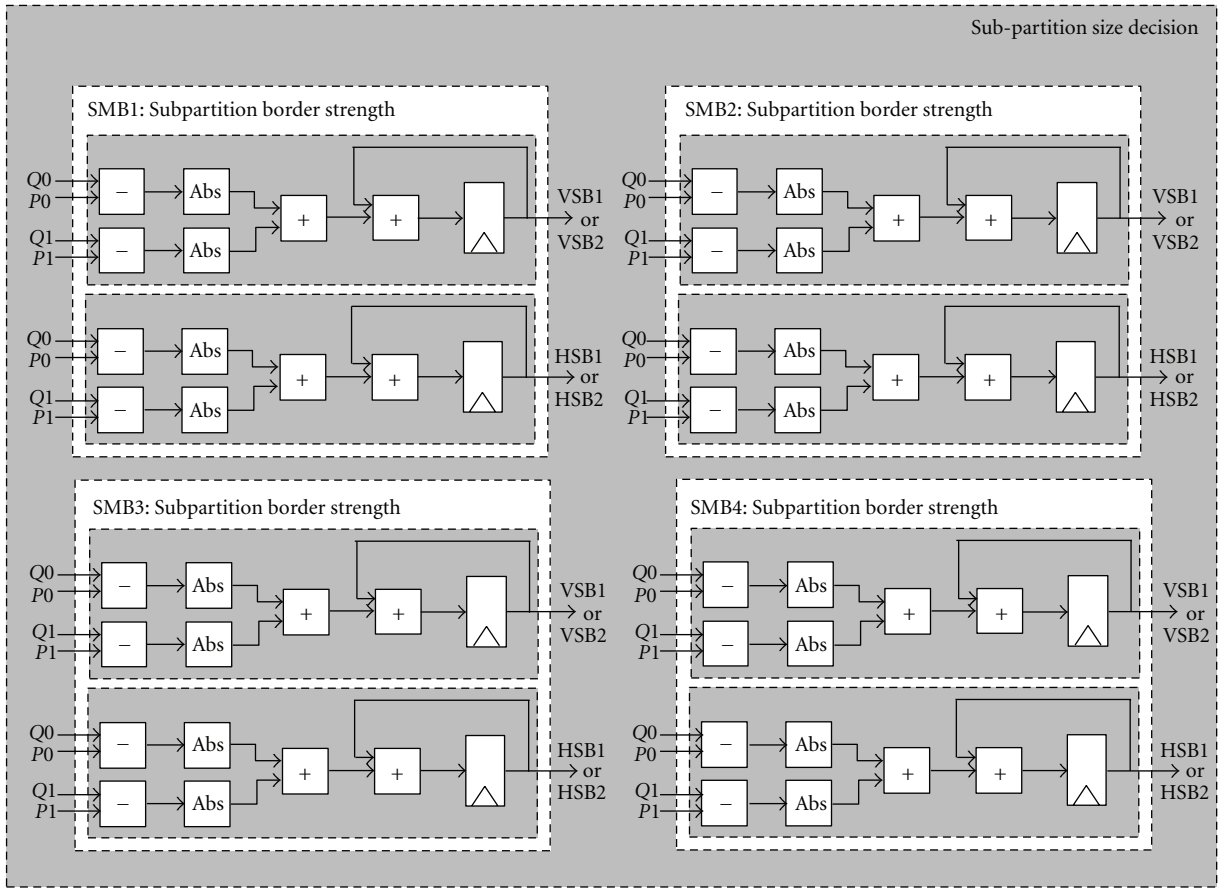

Figure 19: Architecture for sub-partition border strength calculation.

used to detect the partition border strength, where P3, P2, P1, and P0 represent four samples on the left side of the possible partition border and Q3, Q2, Q1, and Q0 represent samples on the right side of the possible partition border (for more details, see Figure 11). One clock cycle per 4×4 block line is required to compute the partition border strength, in a total of 16 clock cycles to perform the calculation of a complete border. After that, the border strength is used in the decision between 16×16, 16×8 and 8×16 partition sizes, as explained in Section 4.2.3.

The architecture for sub-partition border strength calculation used to decide between 8×8, 4×8, 8×4, and 4×4 sub-partitions is similar to that presented in Figure 18. However, fewer samples are used (only P0, P1, Q0, and Q1). Figure 19 presents this architecture. Each submodule computes the subborder strengths corresponding to each sub-macroblock belonging to a macroblock (SMB1, SMB2, SMB3, and SMB4 in the figure). The inputs P0, P1, Q0, and Q1 in each sub-module represent the pixels p_0, p_1, p_2, and p_3 belonging to each sub-macroblock, as previously presented in Figure 12. As explained in Section 4.2.4, the border strength calculation for sub-partition decision is performed in two steps. Four clock cycles are used to perform the calculation of each sub-border strength (defined as VSB1, VSB2, HSB1, and HSB2 in Section 4.2.4), totalizing 8 clock cycles to compute all sub-partition strengths. Two sub-border strengths per sub-macroblock are thus computed after each four clock cycles. After the first four clock cycles, VSB1 and HSB1 are the

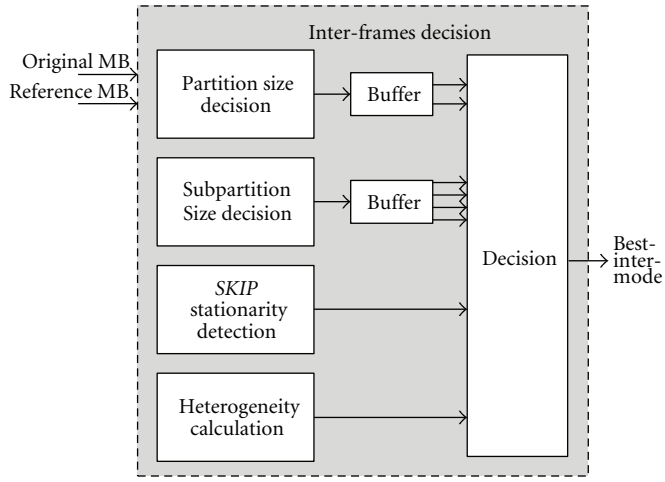

Figure 20: Block diagram of the interframes mode decision architecture.

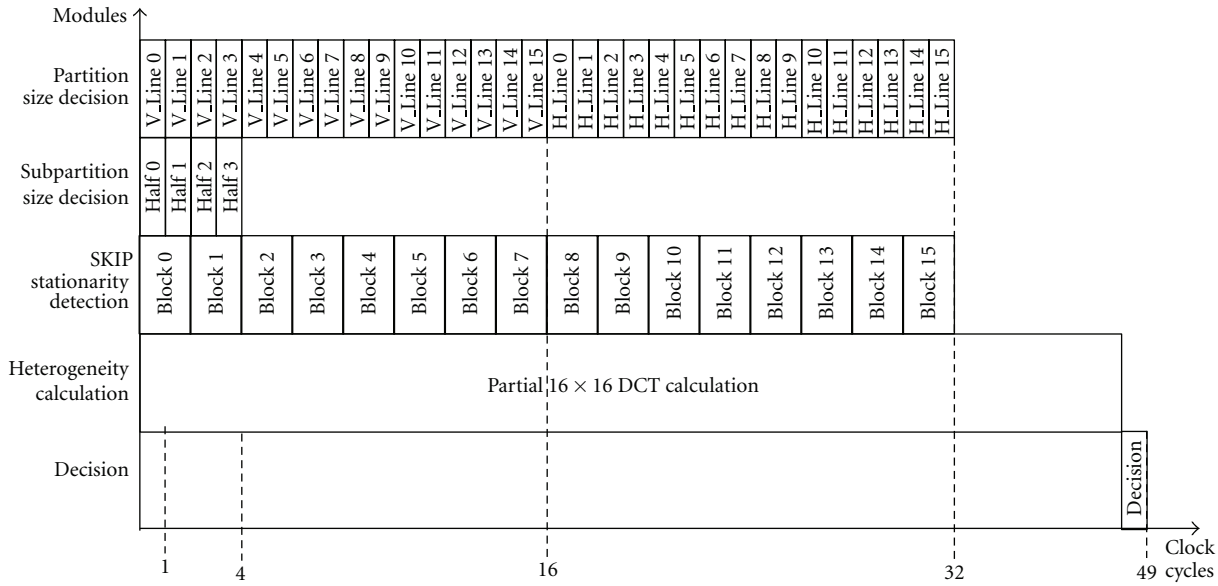

Figure 21: Schedule diagram of the inter-frames mode decision architecture.

available values at the output of each sub-module. At the end of the following four clock cycles, VSB2 and HSB2 are available at the output of the same sub-modules. Once all strengths are computed, they are finally compared to their respective threshold values and the final sub-partition size is chosen.

5.2.4. Complete Inter-Frames Decision Architecture. All architectures presented in the previous sections were integrated to generate the inter-frames decision module, as shown in Figure 20. The *partition size decision* module is responsible to calculate the horizontal and vertical border strengths, the *subpartition size decision* module performs sub-border strength calculations, the *stationarity detection* compares co-located blocks for *SKIP* mode decision, and the *heterogeneity calculation* performs the partial DCT calculation to decide whether or not sub-partitions are used. As previously mentioned, all these operations are performed in parallel, since there is no data dependency between these operations. The results generated by these modules are then evaluated by the *Decision Module*, according to the hierarchical decision method proposed in Figure 8.

In order to provide a better understanding of the architecture working flow, Figure 21 shows the temporal diagram of the inter-frames mode decision. The partition size decision is able to process one block line per clock cycle (e.g., P0–P3 and Q0–Q3 of Figure 11). It means that 16 clock cycles are needed to process each complete border (vertical and horizontal), totalizing 32 clock cycles to generate both border strengths. The sub-partition size decision generates four sub-border strengths per clock cycle. This way, only four clock cycles are needed to generate the 16 border strengths for the whole macroblock. The *SKIP* stationarity detection module is able to perform the SAD calculation between the

Table 5: Synthesis results for the intraframe mode decision architecture.

FPGA Altera Stratix II			TSMC 0.18 μm standard cells	
Number of ALUTs	Number of DLRs	Frequency (MHz)	Area (Number of gates)	Frequency (MHz)
3,267	2,312	98.43	28,518	129.10

Table 6: Synthesis results for the interframes mode decision architecture.

FPGA Altera Stratix II			TSMC 0.18 μm Standard Cells	
Number of ALUTs	Number of DLRs	Max. frequency(MHz)	Area (number of gates)	Max. Frequency(MHz)
5,796	2,859	146.56	85,877	142.40

original and the reference macroblock in 32 clock cycles, since it takes 2 clock cycles to calculate the SAD for one 4×4 block. Finally, the heterogeneity calculation takes 48 clock cycles to generate all coefficients used in the sub-partition detection, and it is thus the critical path of the inter-frames mode decision architecture. The best mode is chosen in one final clock cycle, totalizing 49 cycles to complete the inter-frames mode decision process.

6. Architectural Synthesis Results

Both intra-frame and inter-frames mode decision architectures were synthesized targeting two different technologies in order to allow their use in different applications: (1) Altera Stratix II Field-Programmable Gate Array (FPGA) [23] (using Altera Quartus II) and (2) TSMC 0.18 μm standard cells library [24] (using Mentor Graphics Leonardo Spectrum [25]).

The synthesis results for intra-frame mode decision architecture are presented in Table 5. This architecture used 3,267 altera look-up tables (ALUTs) and 2,312 dedicated logic registers (DLRs) from the specified FPGA, which represent only 4% of the overall device resources. When targeting FPGA prototyping, the architecture reached an operation frequency higher than 98 MHz. The synthesis results for standard cells used less than 29 Kgates and it is able to run at 129.1 MHz.

Considering the highest operation frequency reached by the intra-frame decision architecture and the number of clock cycles used to process a complete macroblock in an HD1080p video sequence (discussed in Section 5.1), the architecture processing rate capability was calculated in frames per second (fps). The intra-frame mode decision is capable of processing up to 468 fps (HD1080p videos), which is much higher than the minimum requirements for real-time video coding. This way, in order to achieve a minimum frame rate of 30 frames per second, the architecture operation frequency can be downscaled to 8.5 MHz, significantly decreasing its energy consumption.

The inter-frames mode decision architecture synthesis results are presented in Table 6. This architecture used 5,796 ALUTs and 2,859 DLRs from the specified FPGA, reaching an operation frequency of 146.6 MHz. The standard cells synthesis reached an operation frequency of 142.4 MHz, using less than 86 Kgates. Considering the standard cells results of frequency and the number of clock cycles necessary to process each macroblock in P slices, the architecture is capable of processing up to 369 fps (HD1080p videos), which is more than enough for real-time applications. To achieve the minimum rate of 30 frames per second, the architecture frequency can be downscaled to 11.9 MHz.

Table 7: Comparison between the proposed mode decision and other related works.

Work	ΔPSNR (dB)	Δbit rate (%)	Δcomplexity (times)	Decision type
Lee et al. [11]	−0.14	+0.69	−4.7	Intra
Fengqin et al. [5]	−0.02	+3.00	−5.9	Intra
Lee and jeon [8]	−0.09	+0.09	−4.0	Intra
Jeon et al. [6]	−0.15	+4.00	−6.0	Intra
Proposed intra	**−0.25**	**+4.88**	**−13.0**	**Intra**
Wu et al. [12]	+0.03	+0.60	−6.0	Inter
Liquan et al. [10]	−0.07	+0.27	0.0	Inter
Proposed inter	**−0.04**	**+6.84**	**−33.0**	**Inter**

7. Comparisons with Related Works

This section presents a comparison between the proposed method and other related works found in the literature (presented in Section 3). Table 7 presents results in terms of PSNR drop, bit rate increase, and mode decision computational complexity decrease, which is measured in number of iterations of the *encoding loop* previously defined. Complexity reduction results (Δcomplexity) for all works considered the worst case decisions and did not consider the complete GOP, so that intra-frame and inter-frames mode decision can be separately compared to the other corresponding intra-only or inter-only decision methods.

From Table 7, it is possible to perceive that the computational complexity reduction provided by the method proposed in this work is much larger than the reduction noticed in other related works, which allowed complexity reductions varying from 0 to 6 times in comparison to the RDO-based mode decision. This work provides a complexity reduction of 33 times when inter-frames mode decision is considered and a reduction of 13 times in computational

Table 8: Comparison between the proposed intra mode decision architecture and other works.

Work	Technology	Hardware resources	Cycles per MB	Max. frequency	HD1080 p fps	Min. frequency (HD1080 p)
Wang [15]	UMC 0.18 μm	10.302 gates	416	66 MHz	31	62.21 MHz
Kao [16]	TSMC 0.13 μm	11.229 gates	672	75 MHz	—	—
Lin [17]	TSMC 0.13 μm	94.700 gates*	560	140 MHz	30	140 MHz
This work	FPGA	3.267 ALUTs 2.312 DLRs	36	98.43 MHz	335	8.26 MHz
	TSMC 0.18 μm	28.518 gates	36	129.1 MHz	439	8.26 MHz

* Hardware resources corresponding to the complete intra prediction module.

complexity when the proposed intra-mode decision is used. On the other hand, the expressive reduction in computational complexity is only possible at the cost of a higher loss in compression efficiency. Nevertheless, such losses are not significantly higher than the losses noticed in other related works which provided a much lower computational complexity reduction.

Table 8 presents a comparison between the proposed intra mode decision architecture and other works found in the literature [15–17]. To the best of the authors' knowledge, no inter mode decision architecture implemented independently from the motion estimation module was published in the literature so far.

The proposed intra mode decision architecture requires the smallest number of clock cycles to process an intra macroblock, which represents a reduction of more than 11 times in comparison with [17], and a reduction of more than 18 times when compared with [16]. The architecture also presents the highest throughput among the related works: an increase of more than 11 times and 14 times in the number of HD1080p frames encoded per second for FPGA and TSMC 0.18 versions, respectively, compared with [15]. All the results presented for the proposed intra mode decision architecture were obtained considering maximum frequency operation. The higher throughput achieved with this architecture allows reducing the operating frequency down to 8.26 MHz while still processing HD1080p videos at 30 fps. With such low operation frequency, very low power can be achieved with the architecture, which is an excellent alternative for battery-powered devices.

8. Conclusions

This work proposed a set of heuristics targeting fast mode decision for H.264/AVC and their respective VLSI architectures. The proposed method is based on a hierarchical organization of these mode decision heuristics, focusing on the acceleration of the best mode decision for computational and power-constrained devices. Luminance and chrominance intra-frame decisions were divided into two steps, one of them based on the distortion to choose the best mode for a fixed block size and the other one based on difference of such distortions to choose the best block size. Inter-frames mode decision was divided into a stationarity-based earlier *SKIP* mode decision, a heterogeneity-based sub-partition detection, and two-border strength-based partition and sub-partition sizes decision.

The mode decision computational complexity was decreased in 47 times through the application of the proposed method in comparison to the RDO-based mode decision. On the other hand, a relatively small increase in bit rate and a negligible decrease in PSNR were noticed. To the best of the authors' knowledge, no other previous work found in the literature is capable of reducing computational complexity in more than 6 times in comparison to the RDO technique.

The developed hardware architectures for both decision modules have shown that the proposed method is capable of processing high-resolution video sequences in real time. The minimum requirement of processing at least 30 frames per second is more than fulfilled, which means that the maximum operation frequency can be downscaled for further energy consumption reduction, which is extremely important for portable devices with limited battery resources.

The obtained results encourage the use of the proposed mode decision algorithms and their respective hardware architectures in H.264/AVC encoders applied to complex multimedia systems with support to high-resolution videos. The fast mode decision is especially useful in energy and computationally limited devices, such as smartphones, video cameras, portable TVs, and other multimedia embedded systems.

References

[1] International Telecommunication Union (ITU), *ITU-T Recommendation H.264/AVC (05/03): Advanced Video Coding for Generic Audiovisual Services*, International Telecommunication Union, Geneva, Switzerland, 2003.

[2] T. Wiegand, G. J. Sullivan, G. Bjøntegaard, and A. Luthra, "Overview of the H.264/AVC video coding standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 560–576, 2003.

[3] G. J. Sullivan and T. Wiegand, "Rate-distortion optimization for: Video compression," *IEEE Signal Processing Magazine*, vol. 15, no. 6, pp. 74–90, 1998.

[4] T. Wiegand, H. Schwarz, A. Joch, F. Kossentini, and G. J. Sullivan, "Rate-constrained coder control and comparison of video coding standards," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 688–703, 2003.

[5] W. Fengqin, F. Yangyu, L. Yajing, and L. Weihua, "Fast intra mode decision algorithm in H.264/AVC using characteristics of transformed coefficients," in *Proceedings of the 5th IEEE International Conference on Visual Information Engineering (VIE '08)*, pp. 245–249, July 2008.

[6] Y. I. Jeon, C. H. Han, S. W. Lee, and H. S. Kang, "Fast intra mode decision algorithm using directional gradients for

H.264," in *Proceedings of the 2009 2nd International Congress on Image and Signal Processing (CISP '09)*, October 2009.

[7] H. Kim and Y. Altunbasak, "Low-complexity macroblock mode selection for H.264/AVC encoders," in *Proceedings of the International Conference on Image Processing (ICIP '04)*, pp. 765–768, Atlanta, Ga, USA, October 2004.

[8] J. Lee and B. Jeon, "Fast mode decision for H.264," in *Proceedings of the IEEE International Conference on Multimedia and Expo (ICME '04)*, pp. 1131–1134, June 2004.

[9] Y. M. Lee, J. D. Wu, and Y. Lin, "An improved SATD-based intra mode decision algorithm for H.264/AVC," in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP '09)*, pp. 1029–1032, April 2009.

[10] S. Liquan, L. Zhi, Z. Zhaoyang, and S. Xuli, "Fast inter mode decision using spatial property of motion field," *IEEE Transactions on Multimedia*, vol. 10, no. 6, Article ID 4657454, pp. 1208–1214, 2008.

[11] Y. M. Lee, Y. T. Sun, and Y. Lin, "SATD-based intra mode decision for H.264/AVC video coding," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 20, no. 3, Article ID 5308396, pp. 463–469, 2010.

[12] D. Wu, F. Pan, K. P. Lim et al., "Fast intermode decision in H.264/AVC video coding," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 15, no. 7, pp. 953–958, 2005.

[13] I. Richardson, *H.264/AVC and MPEG-4 Video Compression-Video Coding for Next-Generation Multimedia*, John Wiley and Sons, Chichester, UK, 2003.

[14] K. Suhring, "H.264/AVC Reference Software," In: Fraunhofer Heinrich-Hertz-Institute, http://iphome.hhi.de/suehring/tml/download/.

[15] J. C. Wang, J. F. Wang, J. F. Yang, and J. T. Chen, "A fast mode decision algorithm and its vlsi design for H.264/AVC intra-prediction," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 17, no. 10, pp. 1414–1422, 2007.

[16] Y. C. Kao, H. C. Kuo, Y. T. Lin et al., "A high-performance VLSI architecture for intra prediction and mode decision in H.264/AVC video encoding," in *Proceedings of the 2006 IEEE Asia Pacific Conference on Circuits and Systems (APCCAS '06)*, pp. 562–565, December 2006.

[17] Y. K. Lin, C. W. Ku, D. W. Li, and T. S. Chang, "A 140-MHz 94 K gates HD1080p 30-frames/s intra-only profile H.264 encoder," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 19, no. 3, Article ID 4783023, pp. 432–436, 2009.

[18] M. Weipeng, Y. Shuyuan, G. Li, and P. Chaoke, "An efficient fast mode decision algorithm based on motion cost for h.264 inter prediction," in *Proceedings of the 2nd International Symposium on Intelligent Information Technology Application Workshop (IITA 2008)*, pp. 550–553, December 2008.

[19] A. C. W. Yu, G. R. Martin, and H. Park, "Fast inter-mode selection in the H.264/AVC standard using a hierarchical decision process," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 18, no. 2, Article ID 4400036, pp. 186–195, 2008.

[20] A. C. Yu, "Efficient block-size selection algorithm for inter-frame coding in H.264/MPEG-4 AVC," in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, pp. III169–III172, May 2004.

[21] G. Corrêa, C. Diniz, S. Bampi, D. Palomino, R. Porto, and L. Agostini, "Homogeneity and distortion-based intra mode decision architecture for H.264/AVC," in *Proceedings of the IEEE International Conference on Electronics, Circuits, and Systems (ICECS '10)*, pp. 591–594, December 2010.

[22] D. Palomino, G. Corrêa, Claudio Diniz, S. Bampi, L. Agostini, and A. Susin, "Algorithm and hardware design of a fast intra-frame mode decision module for H.264/AVC encoders," in *Proceedings of the 24th Symposium on Integrated Circuits and Systems Design (SBCCI '11)*, August 2011.

[23] Altera Corporation: FPGA, CPLD and ASIC, http://www.altera.com/.

[24] Taiwan Semiconductor Manufacturing Company Limited, http://www.tsmc.com/.

[25] The EDA Technology Leader-Mentor Graphics, http://www.mentor.com/.

Hindawi Publishing Corporation
VLSI Design
Volume 2012, Article ID 242989, 10 pages
doi:10.1155/2012/242989

Research Article

Low Cost Design of a Hybrid Architecture of Integer Inverse DCT for H.264, VC-1, AVS, and HEVC

Muhammad Martuza and Khan A. Wahid

Department of Electrical and Computer Engineering, University of Saskatchewan, Saskatoon, SK, Canada S7N 5A9

Correspondence should be addressed to Khan A. Wahid, khan.wahid@usask.ca

Received 2 December 2011; Accepted 6 March 2012

Academic Editor: Maurizio Martina

The paper presents a unified hybrid architecture to compute the 8×8 integer inverse discrete cosine transform (IDCT) of multiple modern video codecs—AVS, H.264/AVC, VC-1, and HEVC (under development). Based on the symmetric structure of the matrices and the similarity in matrix operation, we develop a generalized "decompose and share" algorithm to compute the 8×8 IDCT. The algorithm is later applied to four video standards. The hardware-share approach ensures the maximum circuit reuse during the computation. The architecture is designed with only adders and shifters to reduce the hardware cost significantly. The design is implemented on FPGA and later synthesized in CMOS 0.18 um technology. The results meet the requirements of advanced video coding applications.

1. Introduction

In recent years, different video applications use different video standards, such as H.264/AVC [1], VC-1 [2], and AVS [3]. To improve the coding efficiency further, recently a joint collaboration team on video coding (JCT-VC) is drafting a next generation video coding standards, known tentatively as high efficient video coding (HEVC or H.265) [4]. The target bit rate is half of that of H.264/AVC. Besides, several other effective techniques are proposed in the draft to reduce the complexity of the encoder such as improved intrapicture coding, and simpler VLC coefficients [5]. As a result of these new features, experts predict that the HEVC will dominate the future multimedia market.

In order to meet up the present and future demands of different multimedia applications, it becomes necessary to develop a unified video decoder that can support all popular video standards on a single platform. In recent years, there is a growing interest to develop multistandard inverse transform architectures for advanced multimedia applications. However, most of them do not support AVS, the video codec developed by Chinese government that became the core technology of China Mobile Multimedia Broadcasting (CMMB) [6]. None of the existing works supports the HEVC; thought it is not finalized yet, considering the future prospective of the HEVC [7], it is important to start exploring possible implementation in hardware of the transform unit discussed in the draft.

In this paper, we present a new generalized algorithm and its hardwire implementation of an 8×8 IDCT architecture. The scheme is based on matrix decomposition with sparse matrices and offset computations. These sparse matrices are derived in a way that can be reused maximum number of times during decoding different inverse matrices. All multipliers in the design are replaced by adders and shifters. In the scheme, we first split the 8×8 transformation matrix into two small 4×4 matrices by applying permutation techniques. Then we concurrently perform separate operations on these two matrices to compute the output. It enables parallel operation and yields high throughput, which eventually helps meet the coding requirement of the high resolution video.

The proposed generalized algorithm is later applied to compute the 8 × 8 integer IDCT of AVS. Then we identify the submatrices of AVS and reuse them to compute the IDCT of VC-1. We follow the same principle to compute the other two IDCTs of H.264 and HEVC. For HEVC, we have used the draft matrix discussed in the recent meeting [7]; since it is not yet finalized, we have developed the generalized architecture in such a way that can be easily adjusted to accommodate any changes to the final HEVC format.

2. Previous Works

In recent years, some multistandard inverse transform architectures have been proposed for video applications. Lee's work in [8] presents a 8 × 8 multistandard IDCT architecture based on delta coefficient matrices which can support VC-1, MPEG4, and H.264. It can process up to 21.9 fps for full HD video. Kim's work in [9] describes a design following similar approach of [8] to unify the IDCT and inverse quantization (IQ) operations for those three codecs. However, the design cannot support full HD video format. Qi's work in [10] shows an efficient integrated architecture designed for multistandard inverse transforms of MPEG-2/4, H.264, and VC-1 using factor share (FS) and adder share (AS) strategies for saving circuit resource. The work achieves 100 MHz working frequency for full HD video resolution, but does not support AVS. In another interesting design [11], the authors devise a common architecture by sharing adders and multipliers to perform transform and quantization of H.264, MPEG-4, and VC-1. The common shortcoming of all these designs discussed in [8–11] is that none of them supports the Chinese standard, AVS, nor the HEVC.

In our previous work [12], we have developed a resource shared design using delta coefficient matrices which can compute the 8 × 8 IDCT of VC-1, JPEG, MPEG4, H.264/AVC, and AVS. But due to complex data scheduling and the integration of JPEG (which is an image codec), the decoding capability is limited. The design supports both HD formats, but fails to comply with super resolution (WQXGA). Liu [13] introduces another design to support multiple standards where the design throughput is low (110.8 MHz) and cannot decode HD and WQXGA video. Fan's works in [14, 15] are based on another efficient matrix decomposition algorithm to compute multiple transforms; however, the work is limited to only H.264 and VC-1. There are similar works in [16–18], which are also limited to these two codecs (H.264 and VC-1).

In this paper, we present a generalized low-cost algorithm and its single chip implementation to compute all four modern video standards (AVS, H.264, VC-1, and HEVC). The design meets the requirement of high performance video coding as it can process the HD video at 145 fps, the full HD video at 62 fps, and the WQXGA video at 32 fps. The proposed scheme can be applied to both forward and inverse transformation; however, here we only show the implementation for the inverse process (targeted for decoders).

Table 1: Matrix coefficients of 8-point IDCT.

	AVS [3]	VC-1 [2]	H.264 [1]	HEVC [7]
a	8	12	8	64
b	10	16	12	89
c	9	15	10	75
d	6	9	6	50
e	2	4	3	18
f	10	16	8	83
g	4	6	4	36

3. Proposed Generalized Algorithm for 8 × 8 IDCT

In a video compression system, the transform coding usually employs an 8-point II-type DCT. Since, the forward DCT uses the same basis coefficients and is the transpose of the IDCT matrix, the proposed IDCT scheme is easily applicable to it without any added cost or complexity. The 8-point 1D forward and inverse DCT coefficient matrices are expressed in general form as F and I respectively (below in (1), where, $a, b, c \ldots, g$ denote seven different transform coefficients):

$$F = \begin{bmatrix} a & a & a & a & a & a & a & a \\ b & c & d & e & -e & -d & -c & -b \\ f & g & -g & -f & -f & -g & g & f \\ c & -e & -b & -d & d & b & e & -c \\ a & -a & -a & a & a & -a & -a & a \\ d & -b & e & c & -c & -e & b & -d \\ g & -f & f & -g & -g & f & -f & g \\ e & -d & c & -b & b & -c & d & -e \end{bmatrix},$$

$$I = \begin{bmatrix} a & b & f & c & a & d & g & e \\ a & c & g & -e & -a & -b & -f & -d \\ a & d & -g & -b & -a & e & f & c \\ a & e & -f & -d & a & c & -g & -b \\ a & -e & -f & d & a & -c & -g & b \\ a & -d & -g & b & -a & -e & f & -c \\ a & -c & g & e & -a & b & -f & d \\ a & -b & f & -c & a & -d & g & -e \end{bmatrix}. \quad (1)$$

In this paper, we have denoted the 8 × 8 IDCT transform matrices for AVS, VC-1, H.264/AVC, and HEVC by the letters A, V, H and HV respectively. These seven coefficients $(a, b, c \ldots, g)$ for each of the transforms are different, but integer in nature (as shown in Table 1).

3.1. Development of a Generalized "Decompose and Share" Algorithm. First of all, we derive a generalized matrix decomposition scheme by utilizing the symmetric structure of the matrices and factoring the 8 × 8 matrix into two 4 × 4 sub-matrices as shown below:

$$I = P_0 \cdot I_0, \quad (2)$$

where

$$I_0 = \begin{bmatrix} a & 0 & f & 0 & a & 0 & g & 0 \\ a & 0 & g & 0 & -a & 0 & -f & 0 \\ a & 0 & -g & 0 & -a & 0 & f & 0 \\ a & 0 & -f & 0 & a & 0 & -g & 0 \\ 0 & -e & 0 & d & 0 & -c & 0 & b \\ 0 & -d & 0 & b & 0 & -e & 0 & -c \\ 0 & -c & 0 & e & 0 & b & 0 & d \\ 0 & -b & 0 & -c & 0 & -d & 0 & -e \end{bmatrix},$$

$$P_0 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & 1 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}. \tag{3}$$

The computational complexity of P_0 is only 8 additions. To reduce the complexity of I_0, we use permutation techniques by performing the operations: $I_0 = \widetilde{I} \cdot P_C$.

Where

$$P_C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix},$$

$$\widetilde{I} = \begin{bmatrix} a & f & a & g & 0 & 0 & 0 & 0 \\ a & g & -a & -f & 0 & 0 & 0 & 0 \\ a & -g & -a & f & 0 & 0 & 0 & 0 \\ a & -f & a & -g & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -e & d & -c & b \\ 0 & 0 & 0 & 0 & -d & b & -e & -c \\ 0 & 0 & 0 & 0 & -c & e & b & d \\ 0 & 0 & 0 & 0 & -b & -c & -d & -e \end{bmatrix}. \tag{4}$$

There is no computational cost for P_C as it only permutes the input data set (just needs rewiring). $\widetilde{I}$ can be further decomposed into two 4×4 submatrices, $\widetilde{I}_{00}$ and $\widetilde{I}_{11}$, by the direct sum operation ("$\oplus$") as shown below:

$$\widetilde{I} = \widetilde{I}_{00} \oplus \widetilde{I}_{11}. \tag{5}$$

Thus,

$$I = P_0 \cdot \left(\widetilde{I}_{00} \oplus \widetilde{I}_{11}\right) \cdot P_C, \tag{6}$$

where

$$\widetilde{I}_{00} = \begin{bmatrix} a & f & a & g \\ a & g & -a & -f \\ a & -g & -a & f \\ a & -f & a & -g \end{bmatrix},$$

$$\widetilde{I}_{11} = \begin{bmatrix} -e & d & -c & b \\ -d & b & -e & -c \\ -c & e & b & d \\ -b & -c & -d & -e \end{bmatrix}. \tag{7}$$

Equation (6) forms the general expression of (1). We will use $\widetilde{I}_{00}$ and $\widetilde{I}_{11}$ as the basic building blocks to compute other 8×8 IDCTs. Since, the coefficients in $\widetilde{I}_{00}$ and $\widetilde{I}_{11}$ are fixed, they can be independently implemented, enabling fast computation.

In the following section, we show how (6) can be applied to different IDCT matrices. Another new feature of the proposed scheme is that we take the advantage of the similarity in matrix operation to further optimize the implementation. First of all, we apply (6) to efficiently implement the transformation matrix of AVS. Based on it and the generalized structure, we develop the matrix of VC-1 so that we can share as many units (from AVS) as possible. Next, we develop the IDCT matrix of H.264 based on the same principle (decompose and share from AVS and VC-1). In this stage, we are able to achieve the maximum sharing as it will be shown later (in Section 3.4) that the implementation of H.264 does not cost any extra hardware. Finally, we develop the IDCT of HEVC by further decomposing and reusing the units already implemented (with a minimum addition of extra units).

3.2. Matrix Decomposition for AVS. Let us now construct A (from (1) and Table 1) and apply (6) to compute the 4×4 submatrices, $\widetilde{A}_{00}$ and $\widetilde{A}_{11}$. We then right shift $\widetilde{A}_{00}$ by three bits and decompose it as follows:

$$\frac{\widetilde{A}_{00}}{8} = \begin{bmatrix} 1 & \frac{5}{4} & 1 & \frac{1}{2} \\ 1 & \frac{1}{2} & -1 & -\frac{5}{4} \\ 1 & -\frac{1}{2} & -1 & \frac{5}{4} \\ 1 & -\frac{5}{4} & 1 & -\frac{1}{2} \end{bmatrix} = A_1 \cdot A_2, \tag{8}$$

where

$$A_1 = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & -1 \\ 1 & 0 & -1 & 0 \end{bmatrix}, \qquad A_2 = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & -1 & 0 \\ 0 & \frac{5}{4} & 0 & \frac{1}{2} \\ 0 & \frac{1}{2} & 0 & -\frac{5}{4} \end{bmatrix}. \tag{9}$$

Like P_0, the computational cost of A_1 is only 4 additions. For A_2, we implement $(5/4) \cdot x$ as $(1 + 1/4) \cdot x$—that is right

shift x (arbitrary data) by two bits and then add with x. So, the cost is 6 add and 6 shift operations. Thus in (8), the total computational cost is 10 addition and 6 shift operations. In similar way, we can decompose $\widetilde{A}_{11}$ as shown below:

$$\frac{\widetilde{A}_{11}}{4} = \begin{bmatrix} -\frac{1}{2} & \frac{3}{2} & -\frac{9}{4} & \frac{5}{2} \\ -\frac{3}{2} & \frac{5}{2} & -\frac{1}{2} & -\frac{9}{4} \\ -\frac{9}{4} & \frac{1}{2} & \frac{5}{2} & \frac{3}{2} \\ -\frac{5}{2} & -\frac{9}{4} & -\frac{3}{2} & -\frac{1}{2} \end{bmatrix} = A_3 \cdot A_4, \tag{10}$$

where

$$A_3 = \begin{bmatrix} -1 & 0 & \frac{3}{2} & 1 \\ 0 & 1 & 1 & -\frac{3}{2} \\ -\frac{3}{2} & 1 & -1 & 0 \\ -1 & -\frac{3}{2} & 0 & -1 \end{bmatrix}, \qquad A_4 = \begin{bmatrix} \frac{3}{2} & 0 & 0 & -1 \\ 0 & \frac{3}{2} & 1 & 0 \\ 0 & 1 & -\frac{3}{2} & 0 \\ 1 & 0 & 0 & \frac{3}{2} \end{bmatrix}. \tag{11}$$

For both A_3 and A_4, the coefficient (3/2) can be shared and the cost is: 12 additions and 4 shift operations for A_3; 8 additions and 4 shift operations for A_4. From (8)–(10), we can summarize the final expression of the 8×8 IDCT for AVS as:

$$A = 4 \cdot P_0 \cdot [(A_1 \cdot 2A_2) \oplus (A_3 \cdot A_4)] \cdot P_C. \tag{12}$$

Thus, the total computational cost to implement A is 38 additions and 26 shift operations. In the next section, we will apply (6) to VC-1 and subsequently decompose the matrix in a way so that we can reuse the units already developed for the AVS (from (12)).

3.3. Matrix Decomposition for VC-1. We follow the same principles, as discussed in (8) and (10), to decompose the 8×8 IDCT for the VC-1:

$$V = P_0 \cdot \widetilde{V} \cdot P_C, \tag{13}$$

where

$$\widetilde{V} = \widetilde{V}_{00} \oplus \widetilde{V}_{11}. \tag{14}$$

Now considering the symmetric property and the coefficient distribution patterns between $\widetilde{A}_{00}/8$ (in (8)) and $\widetilde{V}_{00}/8$, we decompose $\widetilde{V}_{00}/8$ as:

$$\frac{\widetilde{V}_{00}}{8} = \begin{bmatrix} \frac{3}{2} & 2 & \frac{3}{2} & \frac{3}{4} \\ \frac{3}{2} & \frac{3}{4} & -\frac{3}{2} & -2 \\ \frac{3}{2} & -\frac{3}{4} & -\frac{3}{2} & 2 \\ \frac{3}{2} & -2 & \frac{3}{2} & -\frac{3}{4} \end{bmatrix} = A_1 \cdot V_2, \tag{15}$$

where

$$V_2 = \begin{bmatrix} \frac{3}{2} & 0 & \frac{3}{2} & 0 \\ \frac{3}{2} & 0 & -\frac{3}{2} & 0 \\ 0 & 2 & 0 & \frac{3}{4} \\ 0 & \frac{3}{4} & 0 & -2 \end{bmatrix} = 2A_2 - V_3,$$

$$V_3 = \begin{bmatrix} \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ \frac{1}{2} & 0 & -\frac{1}{2} & 0 \\ 0 & \frac{1}{2} & 0 & \frac{1}{4} \\ 0 & \frac{1}{4} & 0 & -\frac{1}{2} \end{bmatrix}. \tag{16}$$

From (16), (15) can be reexpressed as:

$$\frac{\widetilde{V}_{00}}{8} = A_1 \cdot (2A_2 - V_3). \tag{17}$$

Now it can be seen how the implementation of AVS matrix (from (12)) can be reused in (17). This matrix decomposition enables hardware sharing and results in significant saving in implementation resources. From (17), the total cost of V_3 and $\widetilde{V}_{00}/8$ is 8 additions and 6 shift operations.

Next based on our careful observation between the computational similarities between $\widetilde{A}_{11}/4$ (in (10)) and $\widetilde{V}_{11}/8$, we devise the decomposition scheme of $\widetilde{V}_{11}/8$ as:

$$\frac{\widetilde{V}_{11}}{8} = \begin{bmatrix} -\frac{1}{2} & \frac{9}{8} & -\frac{15}{8} & 2 \\ -\frac{9}{8} & 2 & -\frac{1}{2} & -\frac{15}{8} \\ -\frac{15}{8} & \frac{1}{2} & 2 & \frac{9}{8} \\ -2 & -\frac{15}{8} & -\frac{9}{8} & -\frac{1}{2} \end{bmatrix} = V_4 \cdot A_{4v}, \tag{18}$$

where $V_4 = A_3 + A_{3V}$,

$$A_{4v} = \begin{bmatrix} \frac{1}{4} & 0 & 0 & -1 \\ 0 & \frac{1}{4} & 1 & 0 \\ 0 & 1 & -\frac{1}{4} & 0 \\ 1 & 0 & 0 & \frac{1}{4} \end{bmatrix}, \qquad A_{3v} = \begin{bmatrix} -1 & -\frac{3}{2} & 0 & -1 \\ \frac{3}{2} & -1 & 1 & 0 \\ 0 & 1 & 1 & -\frac{3}{2} \\ 1 & 0 & -\frac{3}{2} & -1 \end{bmatrix}. \tag{19}$$

By substituting (19) in (18), $\widetilde{V}_{11}/8$ is expressed as:

$$\frac{\widetilde{V}_{11}}{8} = (A_3 + A_{3V}) \cdot A_{4v}. \tag{20}$$

Note that A_{4v} in (19) is structurally similar to A_4 in (10) except the change in the diagonal coefficients. So we only need to implement it; the rest is shared from the architecture of A_4. We do so by adding 4 multiplexers at the output of the four left diagonal elements of A_4 matrix. Then according to (19), we reuse A_3 to compute V_4. As the new matrix A_{3v} can be derived from A_3 by rearranging the rows and changing the polarity of some input bits, we share it from the design of A_3 by adding 4 multiplexers only. Finally, the expression of $\widetilde{V}_{00}$ and $\widetilde{V}_{11}$ from (17) and (20) are substituted in (13) to get the final expression of the IDCT for VC-1:

$$V = 8 \cdot P_0 \cdot \{[A_1 \cdot (2A_2 - V_3)] \oplus [(A_3 + A_{3V}) \cdot A_{4v}]\} \cdot P_C. \tag{21}$$

It is seen from (21) that to implement V, the only new unit that is required is V_3; the rest is shared from the implementation of AVS (from (12)). So, the total computational cost for VC-1 is 12 additions and 10 shift operations.

3.4. Matrix Decomposition for H.264/AVC. Following similar procedure illustrated in the two previous sections, we can simplify the 8×8 transformation matrix for H.264/AVC as shown below:

$$H = P_0 \cdot \widetilde{H} \cdot P_C, \tag{22}$$

where

$$\widetilde{H} = \widetilde{H}_{00} \oplus \widetilde{H}_{11}. \tag{23}$$

In order to ensure the maximum unit sharing, we decompose $\widetilde{H}_{00}/8$ as below:

$$\frac{\widetilde{H}_{00}}{8} = \begin{bmatrix} 1 & 1 & 1 & \frac{1}{2} \\ 1 & \frac{1}{2} & -1 & -1 \\ 1 & -\frac{1}{2} & -1 & 1 \\ 1 & -1 & 1 & -\frac{1}{2} \end{bmatrix} = A_1 \cdot A_{2h}, \tag{24}$$

where

$$A_{2h} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & \frac{1}{2} \\ 0 & \frac{1}{2} & 0 & -1 \end{bmatrix}. \tag{25}$$

In (24), A_1 is directly reused from (12). To share A_{2h} from the architecture of A_2 we simply add two multiplexer units. So there is no additional cost in terms of adders and shifters to compute $\widetilde{H}_{00}/8$. Similarly, we can decompose $\widetilde{H}_{11}/8$ as:

$$\frac{\widetilde{H}_{11}}{8} = \begin{bmatrix} -\frac{3}{8} & \frac{3}{4} & -\frac{5}{4} & \frac{3}{2} \\ -\frac{3}{4} & \frac{3}{2} & -\frac{3}{8} & -\frac{5}{4} \\ -\frac{5}{4} & \frac{3}{8} & \frac{3}{2} & \frac{3}{4} \\ -\frac{3}{2} & -\frac{5}{4} & -\frac{3}{4} & -\frac{3}{8} \end{bmatrix} = A_{3h} \cdot A_{4v}, \tag{26}$$

where

$$A_{3h} = \begin{bmatrix} -\frac{3}{2} & -1 & 1 & 0 \\ 1 & 0 & \frac{3}{2} & -1 \\ -1 & \frac{3}{2} & 0 & -1 \\ 0 & -1 & -1 & -\frac{3}{2} \end{bmatrix}. \tag{27}$$

Here A_{4v} is directly reused from (21) and we share A_{3h} from the architecture of A_3. In this sharing we do not even need to use any multiplexers, because we have already done so while sharing A_{3v} from A_3 in Section 3.3. The final expression of the 8×8 IDCT for H.264 (with all shared units) can be summarized as follows:

$$H = 8 \cdot P_0 \cdot \{[A_1 \cdot A_{2h}] \oplus [A_{3h} \cdot A_{4v}]\} \cdot P_C. \tag{28}$$

It is interesting to note that all terms in (28) are implemented from the terms of (12) and (21); thus, in the proposed scheme, there is no additional cost to implement the IDCT for H.264 which results in significant hardware savings.

3.5. Matrix Decomposition for HEVC. In this section, we develop the transformation matrix for the HEVC based on the principles described before. The 8×8 matrix can be decomposed as:

$$HV = P_0 \cdot \widetilde{HV} \cdot P_C, \tag{29}$$

where

$$\widetilde{HV} = \widetilde{HV_{00}} \oplus \widetilde{HV_{11}}, \tag{30}$$

$$\frac{\widetilde{HV_{00}}}{4} = \begin{bmatrix} 16 & \frac{83}{4} & 16 & 9 \\ 16 & 9 & -16 & -\frac{83}{4} \\ 16 & -9 & -16 & \frac{83}{4} \\ 16 & -\frac{83}{4} & 16 & -9 \end{bmatrix} = A_1.\ (16A_2 + HV_1),$$

$$HV_1 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & \frac{3}{4} & 0 & 1 \\ 0 & 1 & 0 & -\frac{3}{4} \end{bmatrix}. \tag{31}$$

The computational cost of HV_1 is 4 additions and 4 shift operations. Here the coefficient $(3/4) \cdot x$ is factorized as $(x - x/4)$. So the cost of $\widetilde{HV_{00}}$ in (30) is 4 additions and 8 shift operations. Similarly, we decompose $\widetilde{HV_{11}}/4$ as:

$$\frac{\widetilde{HV_{11}}}{4} = \begin{bmatrix} -\frac{9}{2} & \frac{25}{2} & -\frac{75}{4} & \frac{89}{4} \\ -\frac{25}{2} & \frac{89}{4} & -\frac{9}{2} & -\frac{75}{4} \\ -\frac{75}{4} & \frac{9}{2} & \frac{89}{4} & \frac{25}{2} \\ -\frac{89}{4} & -\frac{75}{4} & -\frac{25}{2} & -\frac{9}{2} \end{bmatrix} = (8A_3 + HV_2) \cdot A_{4HV}, \tag{32}$$

where

$$HV_2 = \begin{bmatrix} \frac{7}{4} & \frac{5}{4} & -2 & 0 \\ -\frac{5}{4} & 0 & -\frac{7}{4} & 2 \\ 2 & -\frac{7}{4} & 0 & \frac{5}{4} \\ 0 & 2 & \frac{5}{4} & \frac{7}{4} \end{bmatrix}, \qquad A_{4HV} = \begin{bmatrix} 2 & 0 & 0 & -1 \\ 0 & 2 & 1 & 0 \\ 0 & 1 & -2 & 0 \\ 1 & 0 & 0 & 2 \end{bmatrix}. \tag{33}$$

Combining (29)–(32), we compute the proposed 8×8 IDCT for HEVC as given below:

$$HV = 4 \cdot\ P_0 \cdot \{[A_1.\ (16A_2 + HV_1)] \oplus [(8A_3 + HV_2) \cdot A_{4HV}]\} \cdot P_C. \tag{34}$$

In (34), only the new matrices, HV_1 and HV_2, will be implemented and the rest will be shared from (12). So the total computational cost to implement HV in the proposed design is 24 additions and 28 shift operations. It is important to note that we have carefully decomposed HV so that if there is any change in the final standard, all one needs to do is to update (30) and (32) with new parameters without interrupting the entire design. In summery, the proposed unified design costs 74 additions and 64 shift operations to perform the inverse transformation of four defined video standards.

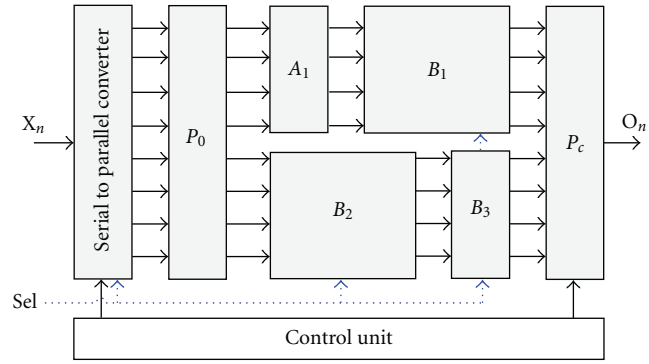

Figure 1: Block diagram of the proposed architecture.

4. Hardware Implementation of the Shared Architecture

In the implementation of the multistandard architectures on a single platform, we have shared the entire hardware unit of the 4×4 matrices, instead of sharing individual adders, shifters, or other factors (as done in [10]). It ensures maximum reduction of hardware cost in our design. The overall block diagram of our proposed scheme is shown in Figure 1. We can see from Figure 1 that the P_0 block splits the 8-point decomposition process to two independent 4-point processes; since these two processes work concurrently, the design throughput is highly increased. The blocks B_1, B_2, and B_3 perform different operations (shared) as shown in Table 2.

Figure 2(a) shows the design of the serial to parallel converter (S2P) block. It performs left shift and then stores the input one by one into eight registers in 8 clock cycles, and at the 9th cycle, all stored input samples are sent to next block, P_0. Here the S2P block apparently functions like a temporary memory buffer as it stores the rows of the input matrix inside eight registers. As a result, the proposed design does not require additional memory architecture. The wrapper architecture (P_C) is shown in Figure 2(b). In this multicodec system, only one IDCT and its associated computational units are activated at a time by the control unit and the select pin (Sel); the rest is disabled. The other blocks are shown in Figure 3. In different stages of the design, several multiplexers are used to ensure proper computation of the IDCT in operation. Finally, the P_C block combines two different set of data and generates one output. In Figure 3, In_0, In_1, …, In_3 represent the inputs coming from the previous block and Out_0, Out_1, …, Out_3 represent the outputs going to the next block. As an example, in

Table 2: Controlling selection of subblocks.

Select (Sel)	IDCT	B_1 $(16/2)\cdot(A_2/A_{2h})-(V_3/HV_1)$	B_2 $(8/1)\cdot[A_3/A_{3h}/(A_3+A_{3v})]+HV_2$	B_3 $A_4/A_{4v}/A_{4HV}$
00	AVS	$2A_2$	A_3	A_4
01	VC-1	$2A_2-V_3$	(A_3+A_{3v})	A_{4v}
10	H.264	A_{2h}	A_{3h}	A_{4v}
11	HEVC	$16A_2-HV_1$	$8A_3+HV_2$	A_{4HV}

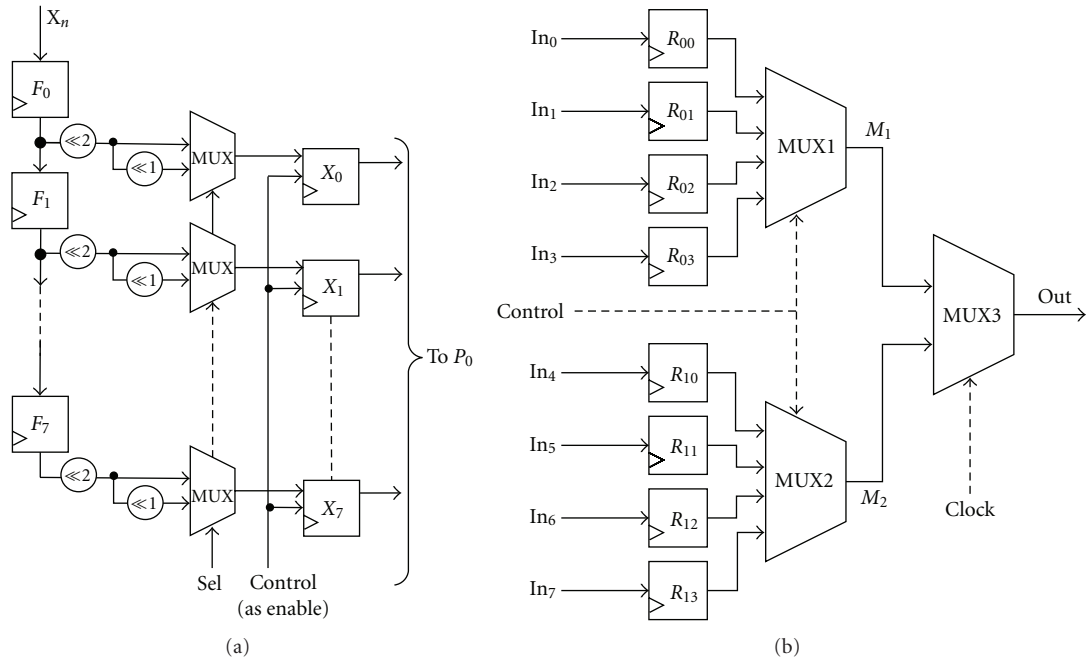

Figure 2: (a) Serial to parallel converter with shifting; (b) output wrapper (P_C block).

Table 3: Comparison of the cost of adders and shifters.

Codecs	Number of adders	Number of shifters
JPEG + MPEG-2/4 + H.264 + VC-1 in [8]	112	—
JPEG + MPEG-2/4 + H.264 + VC-1 in [10]	70	—
H.264 + VC-1 in [14]	76	28
MPEG-2/4 + H.264 + VC-1 + AVS in [13]	76	—
JPEG + MPEG-2/4 + H.264 + VC-1 + AVS in [12]	58	31
Proposed—H.264 + VC-1 + AVS + HEVC	74	64

Figure 3(c) for the shared design of V_3/HV_1, the inputs are coming from A_2/A_{2h} subblock and the outputs are going to P_C block.

The state diagram of the control unit is shown in Figure 4. Here, "r" is reset and "c" is a 3-bit internal counter run by the system clock. There are one reset and four active states. The states of the control signals are also shown in the diagram; for example, in state 1 (S1), S2P is storing the input vector while the output wrapper (P_C block) enables R_{00} from MUX1 and R_{10} from MUX2. Table 2 shows the units that are active depending on the status of the select pin. For example, the select signal will be "00" when the user wants to perform the IDCT of AVS codec. In that case, B_1, B_2, and B_3 will function as $2\cdot A_2$, A_3, and A_4, respectively (the rest is inactive as found in (12)).

5. Performance Analysis and Comparisons

The proposed design is implemented in Verilog and its operation is verified using Xilinx Vertex4 LX60 FPGA. The total number of LUTs needed for this proposed architecture is 2,242. The design is later synthesized using 0.18 μm CMOS technology. The architecture costs 39.3 K gates and 12.15 K standard cells with a maximum operating frequency of 200.8 MHz. The estimated power consumption is 29.9 mW with 3 V supply.

In order to demonstrate the sharing efficiency, we have compared the adder count of our design with the 8-point standalone IDCT matrices of three standards: AVS, VC-1, and H.264/AVC (as presented in [12]). The results are shown in Figure 5. As of today, there is no implementation of the 8×8 IDCT of HEVC; thus, we have implemented it separately for the sake of better comparison. Now, we

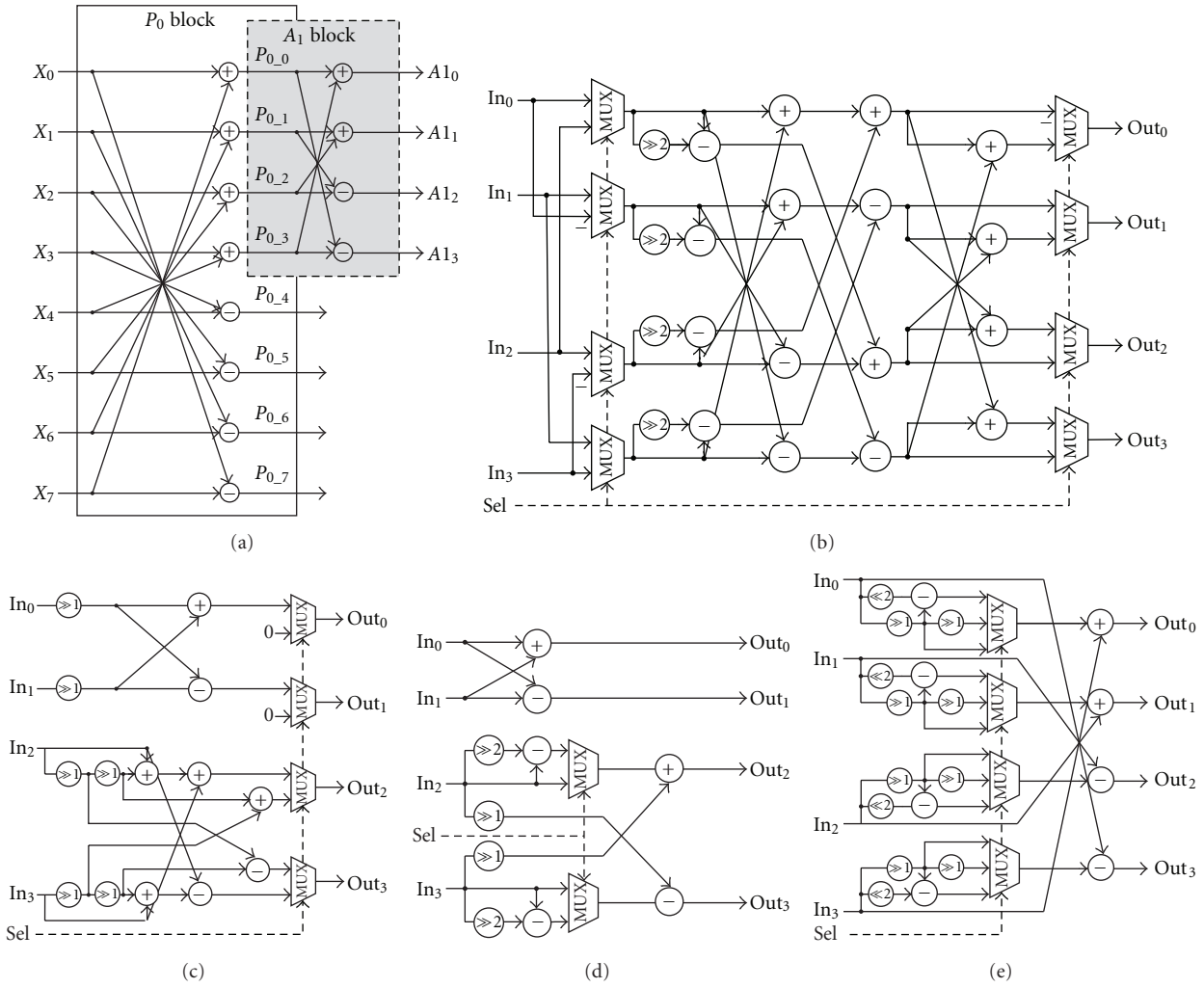

Figure 3: Shared architecture of (a) P_0 and A_1; (b) A_3, A_{3h} and $(A_3 + A_{3v})$; (c) V_3 and HV_1; (d) A_2 and A_{2h}; (e) A_4, A_{4v} and A_{4HV}.

Table 4: Comparison of the resource-shared 8-point 1-D IDCT architecture.

Scheme	Tech.	Gate count	Freq. (MHz)	Full HD support?	Super Resolution support?	Supporting standards			
						H.264	VC-1	AVS	HEVC
Lee's [8]	0.13 μm	19.1 K	136	Y	o	Y	Y	o	o
Kim's [9]	0.13 μm	30.9 K	151	o	o	Y	Y	o	o
Qi's [10]	0.13 μm	18 K	100	Y	o	Y	Y	o	o
Lee's [11]	0.13 μm	10.5 K	123	o	o	Y	Y	o	o
Wahid's [12]	0.18 μm	19.8 K	194.7	Y	o	Y	Y	Y	o
Liu's [13]	0.13 μm	16.5 K	110.8	—	—	Y	Y	Y	o
Fan's [14]	0.18 μm	7.14 K	100	—	—	Y	Y	o	o
Li's [19]	0.18 μm	13.7 K	200	Y	o	Y	o	o	o
Proposed	0.18 μm	39.3 K	200.8	Y	Y	Y	Y	Y	Y

"Y": yes; "o": No; "—": no information.

can see from Figure 5 that a total of 104 adders is required to implement these four transforms without sharing. The proposed shared design can compute all of them with 28.9% less adders. Moreover, the savings achieved in individual standards due to the sharing are also marked on the figure.

It is important to note that, though the proposed design costs 38 adders to implement AVS, it does not cost any additional adder units to implement H.264. Hence, AVS and H.264 combined together cost only 38 adders (compared to 48 for standalone implementations). The cost

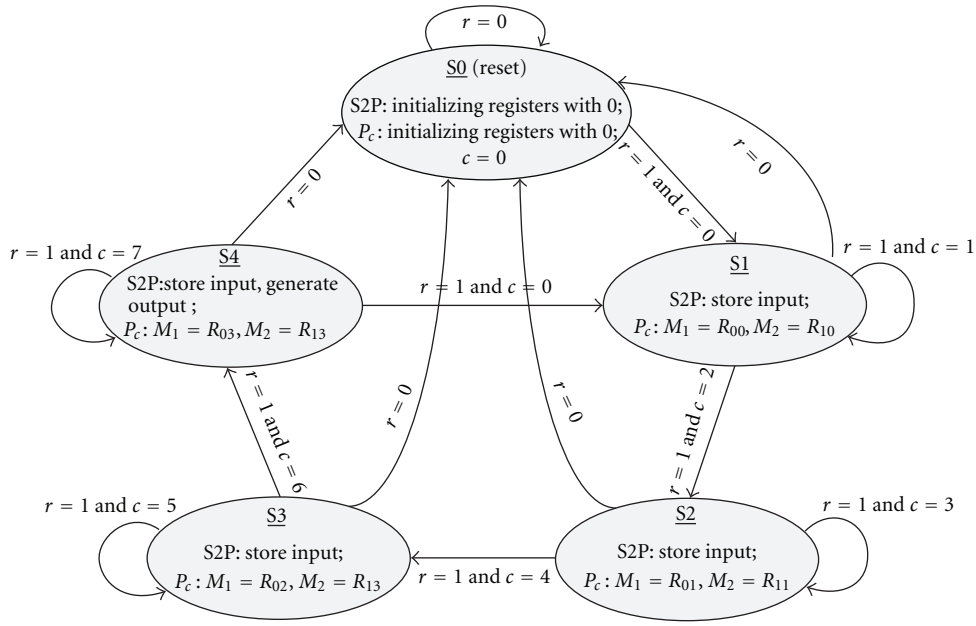

Figure 4: State diagram of control unit.

Table 5: Comparison of decoding capability (minimum support of three codecs).

Scheme	HD resolution				Super resolution	
	1920 × 1080		1280 × 720		2560 × 1600	
	Time to transmit 1 frame (msec)	Frame per second (fps)	Time to transmit 1 frame (msec)	Frame per second (fps)	Time to transmit 1 frame (msec)	Frame per second (fps)
Lee's [8]	22.8	44	10.2	98	x	x
Kim's[9]	x	x	10.2	98	x	x
Qi's [10]	31.1	32	13.8	72	x	x
Wahid's [12]	16.7	60	7.1	140	x	x
Proposed	15.5	64	6.9	145	30.6	32

"—": no information; "x": not supported by the hardware.

of implementing shift operation is considered insignificant in the computation. In Table 3, we compare the cost of the proposed scheme with available existing designs in the literature. None of the designs in this table supports HEVC (which is computationally expensive due to large matrix parameters as shown in Table 1). Although, the designs in [10, 12] cost fewer adders, it is shown later that the proposed scheme outperforms it in decoding capacity. Considering the fact that, the proposed architecture can decode the IDCT for four video codecs, it consumes the least number of adders compared to others.

In Table 4, we have summarized the performance in terms of gate count, maximum working frequency, and standard support with other designs. Only the design in [19] has frequency closer to us, but it supports only H.264. Similarly, designs in [11, 14] support only two codecs and accordingly cost lesser hardware than ours. Among other designs [8–10, 12, 13] are comparable to our design as they support as many as three codecs. While working at maximum capacity, the proposed design can process 200.8 million pixels/sec.

In order to have a better assessment among comparable designs (e.g., minimum support of three codecs), in Table 5 we compare the decoding capability (using 4 : 2 : 0 luma-chroma sampling) of the proposed approach with that of [8–10, 12]. In our work, the maximum achieved frame rate of a 1080 p video is = $200.8 \times 10^6/(1920 \times 1080 + 2 \times 960 \times 540) = 64.56 \approx 64$ fps, which is the highest compared to all other designs in Table 5. Considering the current trends to use super resolution monitors, in this table we have also compared the decoding capabilities for the Wide Quad eXtended Graphics Array (WQXGA, with resolution of 2560 × 1600 pixels). Thus, it can be seen that the proposed design cannot only decode AVS, H.264/AVC, VC-1, and HEVC videos, but also can maintain relatively higher operational frequency to meet the requirements of real time transmission (the target fps to transmit HD, full HD, and QWXGA video are 120, 60, and 30, resp.). From the performance analysis,

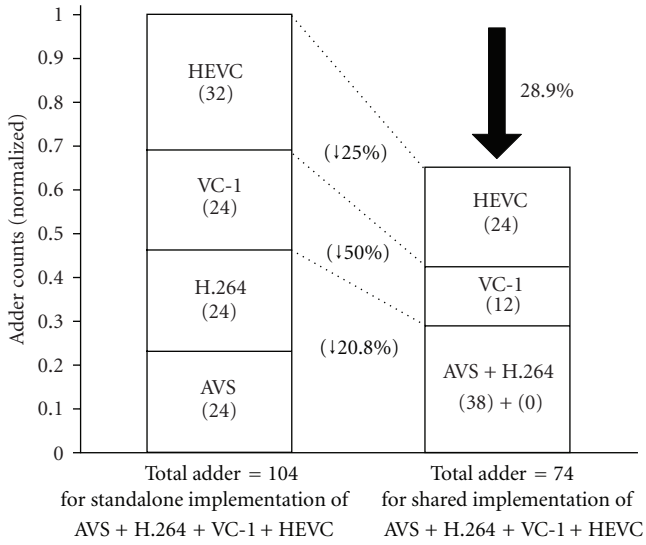

Figure 5: Cost of the proposed scheme—standalone versus cost-shared.

the scheme is found to be competitive as it can transmit the highest number of frames per seconds and, hence, takes the least time to transmit one frame at a given resolution.

6. Conclusion

In this paper, we present a generalized algorithm and a hardware-shared architecture by using the symmetric property of the integer matrices and the matrix decomposition to compute the 8-point 1-D IDCT for four modern video codecs: H.264/AVC, VC-1, AVS, and HEVC (draft in stage). The architecture is designed in such a way that can accommodate any change in the final release of the HEVC. We first apply the generalized scheme to AVS-based transform unit, and then gradually build the rest of the transform units on top of another to maximize the sharing. The performance analysis shows that the proposed design satisfies the requirement of all four codecs and achieves the highest decoding capability. Overall, the architecture is suitable for low-cost implementation in modern multicodec systems.

Acknowledgment

The authors would like to acknowledge the Natural Science and Engineering Research Council of Canada (NSERC) for its support to this research paper.

References

[1] ITU-T Rec, "H.264/ISO/IEC 14496-10 AVC," 2003.

[2] "Standard for Television: VC-1 Compressed Video Bitstream Format and Decoding Process," SMPTE 421M, 2006.

[3] GB/T 20090.1, "Information technology - Advanced coding of audio and video – Part 1: System," Chinese AVS standard.

[4] G. J. Sullivan and J.-R. Ohm, "Recent developments in standardization of high efficiency video coding (HEVC)," in *Applications of Digital Image Processing XXXIII*, vol. 7798 of *Proceedings of SPIE*, August 2010.

[5] K. Ugur, K. Andersson, A. Fuldseth et al., "High performance, low complexity video coding and the emerging hevc standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 20, no. 12, pp. 1688–1697, 2010.

[6] C. C. Ju, Y. C. Chang, C. Y. Cheng et al., "A full-HD 60fps AVS/H.264/VC-1/MPEG-2 video decoder for digital home applications," in *International Symposium on VLSI Design, Automation and Test (VLSI-DAT '11)*, pp. 117–120, April 2011.

[7] Joint Collaborative Team – Video Coding, *CE10: Core transform design for HEVC*, JCTVC-G495, Geneva, Switzerland, 2011.

[8] S. Lee and K. Cho, "Architecture of transform circuit for video decoder supporting multiple standards," *Electronics Letters*, vol. 44, no. 4, pp. 274–276, 2008.

[9] S. Kim, H. Chang, S. Lee, and K. Cho, "VLSI design to unify IDCT and IQ circuit for multistandard video decoder," in *12th International Symposium on Integrated Circuits (ISIC '09)*, pp. 328–331, December 2009.

[10] H. Qi, Q. Huang, and W. Gao, "A low-cost very large scale integration architecture for multistandard inverse transform," *IEEE Transactions on Circuits and Systems II*, vol. 57, no. 7, pp. 551–555, 2010.

[11] S. Lee and K. Cho, "Circuit implementation for transform and quantization operations of H.264/MPEG-4/VC-1 video decoder," in *International Conference on Design and Technology of Integrated Systems in Nanoscale Era (DTIS '07)*, pp. 102–107, September 2007.

[12] K. A. Wahid, M. Martuza, M. Das, and C. McCrosky, "Efficient hardware implementation of 8×8 integer cosine transforms for multiple video codecs," *Journal of Real-Time Image Processing*. In press.

[13] G. Liu, "An area-efficient IDCT architecture for multiple video standards," in *2nd International Conference on Information Science and Engineering (ICISE '10)*, pp. 3518–3522, December 2010.

[14] C. P. Fan and G. A. Su, "Efficient low-cost sharing design of fast 1-D inverse integer transform algorithms for H.264/AVC and VC-1," *IEEE Signal Processing Letters*, vol. 15, pp. 926–929, 2008.

[15] C. Fan and G. Su, "Fast algorithm and low-cost hardware-sharing design of multiple integer transforms for VC-1," *IEEE Transactions on Circuits and Systems II*, vol. 56, pp. 788–792, 2009.

[16] D. Zhou, Z. You, J. Zhu et al., "A 1080p@60fps multi-standard video decoder chip designed for power and cost efficiency in a system perspective," in *Symposium on VLSI Circuits*, pp. 262–263, June 2009.

[17] C. P. Fan and Y. L. Lin, "Implementations of low-cost hardware sharing architectures for fast 8×8 and 4×4 integer transforms in H.264/AVC," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. 90, no. 2, pp. 511–516, 2007.

[18] Y. C. Chao, S. T. Wei, C. H. Kao, B. D. Liu, and J. F. Yang, "An efficient architecture of multiple 8×8 transforms for H.264/AVC and VC-1 decoders," in *1st International Conference on Green Circuits and Systems (ICGCS '10)*, pp. 595–598, June 2010.

[19] Y. Li, Y. He, and S. Mei, "A highly parallel joint VLSI architecture for transforms in H.264/AVC," *Journal of Signal Processing Systems*, vol. 50, no. 1, pp. 19–32, 2008.