2010

# Reconfigurable Computing For Video Coding

Jian Huang
*University of Central Florida*

# Reconfigurable Computing for Video Coding

by

Jian Huang

B.S. East China University of Science and Technology, 2003
M.S. University of North Texas, 2007

A dissertation submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in the School of Electrical Engineering and Computer Science
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Summer Term
2010

Major Professor: Jooheung Lee

# ABSTRACT

Video coding is widely used in our daily life. Due to its high computational complexity, hardware implementation is usually preferred. In this research, we investigate both ASIC hardware design approach and reconfigurable hardware design approach for video coding applications.

First, we present a unified architecture that can perform Discrete Cosine Transform (DCT), Inverse Discrete Cosine Transform (IDCT), DCT domain motion estimation and compensation (DCT-ME/MC). Our proposed architecture is a Wavefront Array-based Processor with a highly modular structure consisting of $8 \times 8$ Processing Elements (PEs). By utilizing statistical properties and arithmetic operations, it can be used as a high performance hardware accelerator for video transcoding applications. We show how different core algorithms can be mapped onto the same hardware fabric and can be executed through the pre-defined PEs. In addition to the simplified design process of the proposed architecture and savings of the hardware resources, we also demonstrate that high throughput rate can be achieved for IDCT and DCT-MC by fully utilizing the sparseness property of DCT coefficient matrix.

Compared to fixed hardware architecture using ASIC design approach, reconfigurable hardware design approach has higher flexibility, lower cost, and faster time-to-market. We

propose a self-reconfigurable platform which can reconfigure the architecture of DCT computations during run-time using dynamic partial reconfiguration. The scalable architecture for DCT computations can compute different number of DCT coefficients in the zig-zag scan order to adapt to different requirements, such as power consumption, hardware resource, and performance. We propose a configuration manager which is implemented in the embedded processor in order to adaptively control the reconfiguration of scalable DCT architecture during run-time. In addition, we use LZSS algorithm for compression of the partial bitstreams and on-chip BlockRAM as a cache to reduce latency overhead for loading the partial bitstreams from the off-chip memory for run-time reconfiguration. A hardware module is designed for parallel reconfiguration of the partial bitstreams. The experimental results show that our approach can reduce the external memory accesses by 69% and can achieve 400 MBytes/s reconfiguration rate. Detailed trade-offs of power, throughput, and quality are investigated, and used as a criterion for self-reconfiguration. Prediction algorithm of zero quantized DCT (ZQDCT) to control the run-time reconfiguration of the proposed scalable architecture has been used, and 12 different modes of DCT computations including zonal coding, multi-block processing, and parallel-sequential stage modes are supported to reduce power consumptions, required hardware resources, and computation time with a small quality degradation. Detailed trade-offs of power, throughput, and quality are investigated, and used as a criterion for self-reconfiguration to meet the requirements set by the users.

Dedicated to,

My parents

# ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my major professor, Dr. Jooheung Lee, for his support and guidance on my PhD research. He has provided me with a lot of valuable advices. I would also like to thank Dr. Ronald DeMara, Dr. Jun Wang, and Dr. Morgan Wang for serving as members of my dissertation committee. I am thankful to my labmates, Sumedha and Rakan for their generous support. I am also grateful to my friends, Xiang, Kejiu, Pengju, Ping, Ying, Ke for their encouragements during my PhD study.

# TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# CHAPTER 1
# INTRODUCTION

## 1.1 Reconfigurable Computing

Due to the increasing complexity of today's SoC design, it becomes common to use pre-defined Intellectual Property (IP) cores to meet the requirements such as manufacturing yield and time-to-market schedule. These standard IP-cores are used as building blocks to simplify the design of more complex systems through ASIC or FPGA design flows. Currently, various IP cores for image/video signal processing applications are commercially available in the market. These cores are pre-constructed circuits with details of area, power, and performance provided by the vendors, and difficult to be modified by the users. These offerings are suitable for the standard ASIC design flow since these cores are designed separately to perform tailored complex operations to maximize the performance, and hardware architecture of IP core fixed at design time will not be adjusted in ASIC implementation after fabrication process. However, its fixed architecture provides inherent limitation of achieving adaptive computing capabilities in terms of area, power, and performance when it is used on the FPGAs. Therefore, it becomes difficult to apply dynamic approaches for designing highly adaptable and evolvable system by using those commercial hard IP cores.

Reconfigurable hardware, such as Field Programmable Gate Array (FPGA), can now accommodate digital systems with more than 10 million equivalent gates in its reconfigurable fabric. Moreover, its abundant hardware resources, such as configurable logic blocks and programmable interconnect resources, can be considered as "ready to be used" virtual multiprocessors. Therefore, its unlimited capabilities of various design configurations and dedicated hardware components, such as microprocessors, DSP logics, memory blocks, and other specific modules, make FPGAs one of the ideal platforms to implement and test homogeneous or heterogeneous multiprocessor SoC (MPSoC) [SRR06] [GHP08].

Currently, more and more applications are choosing FPGAs as an implementation platform due to its flexibility, low cost, and fast time-to-market. In addition, adaptability of computing systems to support various multimedia formats and equipments with different computational requirements is highly desirable in the ubiquitous and seamless computing era. However, it reduces the benefits significantly to use pre-constructed IP-cores on the reconfigurable hardware platform since these traditional approaches suitable for the standard ASIC design flow can not change its own architecture efficiently to adapt to changing environments.

Dynamic partial reconfiguration is a more advanced methodology of FPGA reconfigurability from Xilinx [LBM06]. The main advantage of dynamic partial reconfiguration is that it can change the configuration of the FPGA during the run-time. In addition, it can reduce the bitstream sizes and the reconfiguration time. Self-reconfiguration can be achieved using

Internal Configuration Access Port (ICAP) as a configuration interface and use embedded processor to control the reconfiguration during the run-time.

Majer et al. developed a new FPGA-based reconfigurable computer called the Erlangen Slot Machine (ESM) [MTA07]. It uses slot based architecture which allows the slots to be reconfigured independently of each other during run-time.

Claus et al. proposed a reconfigurable hardware architecture for video-based driver assistance applications in future automotive systems [CZM07]. Different operations such as shape engine, tunnel engine, and taillight engine, can be dynamically reconfigured during run-time.

Braun et al. presented a waveform-like reconfiguration for dynamic partial reconfiguration [BPK08]. It decreases the overhead of reconfiguration by dividing the reconfiguration modules according to the specific data graph. Therefore, some parts of the data graph start processing while the following parts of the data graph are still being reconfigured.

Bayar and Yudakul designed a parallel configuration access port core with bitstream decompression module for self-reconfiguration [Bay08]. The partial bitstreams are stored in the BlockRAM and decompressed via cPCAP core at the time of reconfiguration. They used Virtex-II device and can achieve reconfiguration speed at 50 MBytes. However, the disadvantage of their approach is to increase reconfiguration overhead due to the decompression process during the run-time reconfiguration. Since they store all the partial bitstreams in the BlockRAM, this approach will be inefficient when the number of partial bitstreams is increasing.

## 1.2 Video Coding

Video coding technology has been developed rapidly over the last decades. MPEG-2 and MPEG-4 standards developed by ISO/IEC Moving Picture Expert Group (MPEG) are widely used in Digital Television (DTV) Broadcasting, DVD, Video-On-Demand (VOD), and video recording in digital camera. H.263 and H.264 standards by International Telecommunication Union - Telecommunication Standardization Sector (ITU-T) are commonly used in video conferencing and video telephony [WSB03] [OBL04] [RR03]. Functional blocks with high computational complexities, such as Discrete Cosine Transform (DCT), Inverse Discrete Cosine Transform (IDCT), Motion Estimation & Compensation (ME & MC) are the core algorithms commonly used in these standards. DCT, motion estimation and compensation, Quantization, and entropy coding are used to remove spatial, temporal, and statistical redundancies present in the video data. These tasks together consume about 90% of the total computation time in video coding [GGA92]. These functional blocks have high computational complexities and also significantly influence the visual quality of reconstructed images at a given bit rate. Due to their high computational complexities, hardware implementations have been preferred to satisfy real-time requirements of video coding systems [PC05].

A typical video encoder structure is shown in Fig. 1.1 [HB96]. Motion estimation is used to reduce inter-frame temporal redundancy, i.e., generating motion vectors by comparing current frame with reference frame. On the other hand, DCT is used to reduce intra-frame spatial redundancy by transferring data from spatial domain to frequency domain.

After DCT coefficients are computed, most energy will be concentrated into low frequency coefficients, and many other high frequency coefficients will be close to zero. Compression is achieved by discarding these near zero coefficients. Besides DCT, quantization (Q), variable length coding (VLC), and buffer control (BUF) unit are also included in a typical encoder system. The inverse quantization (IQ) and inverse DCT (IDCT) are used to reconstruct the reference frame, which will be used in motion compensated predictor and motion estimation unit.

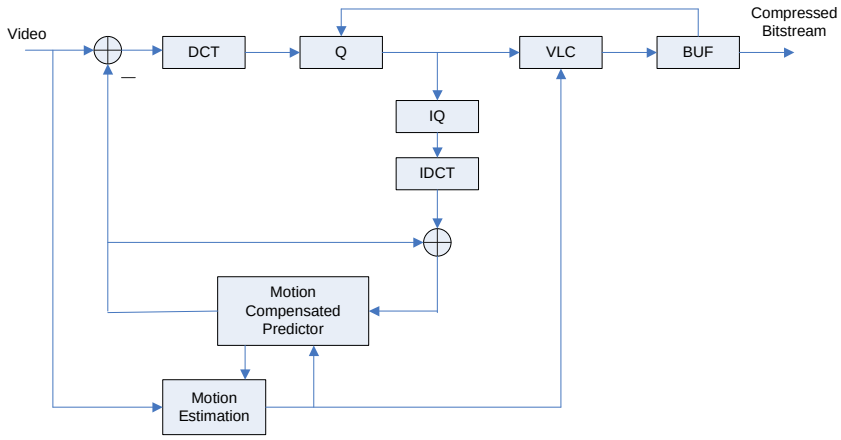



Figure 1.1: Typical video encoder structure

Typically, DCT coefficients are calculated on an $8 \times 8$ block by block basis. However, since most of the high frequency coefficients are likely to be zeros after quantization process, it becomes inefficient to compute all of the 64 DCT coefficients and discard most of the high frequency DCT coefficients during the quantization process. Adaptive control of the number of DCT coefficients to be encoded can reduce the power consumptions, increase the throughput, and reduce the bandwidth, with a small trade-off in image/video quality.

Xanthopoulos and Chandrakasan proposed a low-power DA-based DCT core using adaptive bitwidth and arithmetic activity considering signal correlations and quantization [XC00]. First, they used a MSB rejection (MSBR) module to reduce the number of arithmetic operations required in the presence of correlated inputs. Second, they used a row-column classification (RCC) module to reduce the overall signal activity by introducing a small error in the arithmetic computation. While they added MSB circuits and RCC circuits to control the inputs to the DCT computations, their architecture is fixed for DCT computations.

Vleuten et al. proposed a low-complexity scalable DCT image compression scheme [VKH00]. It eliminates entropy coding and quantization and achieves quality scalability by encoding the DCT coefficients bit plane by bit plane. They used an adaptive rectangular zone for DCT encoding.

Kinane et al. proposed an energy-efficient hardware architecture for variable N-point 1D-DCT which can be used to implement the shape adaptive DCT [KMO04]. They used a new distributed arithmetic architecture (NEDA) for the 1D-DCT implementation [SPC02]. They used clock gating to shut down the redundant logic based on the value N.

Chang et al. proposed an implementation of $8 \times 8$ 2D-DCT chip design using direct 2-D algorithm. The parallel distributed arithmetic technique is used to realize constant multiplication [CJC00]. ROM is used in the 1D-DCT basic cell, and transposed memory is used for 2D-DCT implementation. The chip is implemented using TSMC 0.6 $\mu m$ technology, and 138 mW power consumption at 100MHz is achieved.

Ghosh et al. presented an implementation of a 2D-DCT architecture using coefficient distributed arithmetic (CoDA) for low-power applications [GVB05]. It uses no ROMs, and minimum number of additions are used. A transpose memory is used for 2D-DCT implementation. The design is implemented using FPGA consuming 281 mW power at 50MHz.

Gong et al. proposed a new cost-effective VLSI implementation of a 2-D DCT [GHC04]. Different from [CJC00] and [GVB05], it uses transpose free row column decomposition method. It replaces the transpose circuits with permutation networks and parallel memory modules. The design is implemented using 0.25 $\mu m$ CMOS technology, and the area was about 1.5 $mm^2$ with the operating clock frequency at 125 MHz.

## 1.3 Transcoding

With the development of communication technology and consumer electronics, multimedia contents are transmitted over a heterogeneous wired or wireless networks with different bandwidth requirements, and received by various terminal devices such as laptop, personal digital assistant (PDA), and cell phone with different capabilities (computing, display, power, quality). Video transcoding is a key technology to adapt the video contents dynamically to meet different requirements of network capabilities and terminal device capabilities [XLS05] [VCS03] [AWS05]. Video transcoding is to transform the encoded video streams from one format to another. There are different kinds of transcoders. First, it can be used to change the bit-rate for different bandwidth requirements [YSX99] [AG05].

Second, it can be used to change spatial or temporal resolution [ZYB98]. Third, it can be used to convert from one standard to another [QSL06]. Fourth, it can be used to insert new information such as logo, watermark, and error-resilience features to the encoded video streams [HG98] [SK96].

Transcoding architectures can be categorized into spatial domain approach and DCT domain approach. Spatial domain approach is to change the video streams' parameters in the spatial domain [YSX99], while DCT domain approach is to change the video streams' parameters in the DCT domain [AG05] [ZYB98]. Spatial domain transcoder is more computation-intensive than DCT domain transcoder because spatial domain transcoder needs to perform IDCT and DCT pair to transform the video streams. An example of the DCT domain transcoder is shown in Fig. 1.2. CIF to QCIF down conversion in the DCT domain is achieved without using IDCT and DCT pair. Motion compensation is performed directly in the DCT domain. 40% reduction in computational complexity is achieved by using this approach.

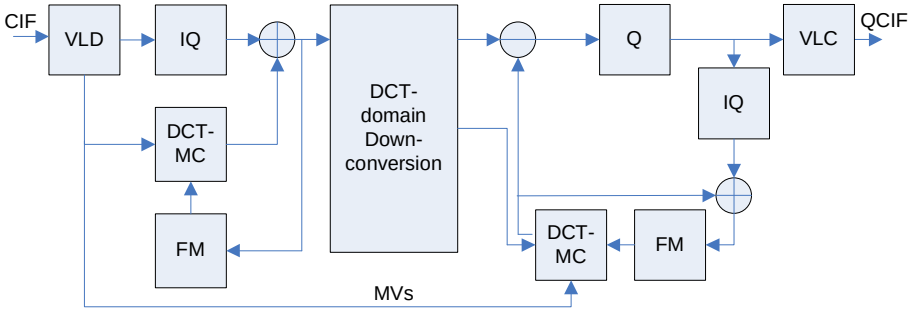



Figure 1.2: CIF to QCIF down-conversion in the DCT domain.

DCT-MC is the major computation-expensive component in the DCT domain transcoder. Therefore, improving performance of DCT-MC is critical to enhance the overall performance of DCT domain transcoder. Compared to software implementation, hardware implementation of DCT-MC can improve the performance of DCT-MC by exploring the data parallelism of DCT-MC computations. Our implementation can take advantage of the sparseness of DCT coefficients matrix to further reduce the computational complexity of DCT-MC.

In addition to performing DCT-MC, our proposed unified architecture can also perform DCT, IDCT, and Motion Estimation in the DCT domain. This can save the design time and hardware resources for implementing these functions separately. These additional functions which can be performed using our unified architecture make the video transcoding system highly flexible. For example, IDCT can be used to display the reconstructed video without using extra hardware resources. IDCT computation performed by our architecture can also take advantage of the sparseness of DCT coefficients matrix. DCT can be used for logo insertion [HG98]. ME in the DCT domain can be used for motion vector re-estimation for temporal transcoder to overcome the quality degradation due to the mismatch between prediction and residual components [YSL99].

The proposed unified architecture consists of highly regular, parallel, and pipelined Processing Elements (PEs), and greatly simplifies complicated design processes of the computation-intensive algorithms, such as DCT, IDCT, ME & MC in the DCT domain by mapping those different algorithms onto the same hardware fabric.

## 1.4 ZQDCT Prediction

Due to the Quantization, most of the high frequency DCT coefficients become zeros, especially for very low bit-rate video coding. Therefore, it is very inefficient to compute all the DCT coefficients and discard them during the quantization processes.

There are extensive works for predicting zero quantized DCT (ZQDCT) coefficients to reduce computational complexity of DCT and Q computations. In [PS99], Laplacian distributions based prediction algorithm is proposed to detect ZQDCT for different zones including all-zeros, dc only, $2 \times 2$ DCT, and $4 \times 4$ DCT. Sum of Absolute Difference (SAD) values and Quantization Parameter (QP) are used to compare with different thresholds for predictions. In [WKK07], a hybrid model derived from Gaussian distribution and statistical analysis of the dynamic range of DCT coefficients is used for ZQDCT predictions of different zones.

The above mentioned methods are all targeted for software implementations. However, due to the high computational complexity, most high-quality video encoders are implemented using hardware. Our motivation is that traditional H/W design approach in video coding system is difficult to effectively utilize time-varying computational complexity which highly depends on the input signal characteristics. In other words, capability of achieving adaptive computing during run-time is significantly limited. In this work, we propose highly versatile reconfigurable architecture which mainly consists of prediction module and computing modules. Prediction module is designed to accurately estimate time varying computing loads of DCT block during run-time, and used to selectively choose and update the required number

of computing modules. Then, we demonstrate that by fully taking advantage of precise estimation of time-varying computational complexity and distribution of hardware resources accordingly to the reconfigurable hardware fabric, a number of advantages in terms of power, computational capability, hardware resources, and compression efficiency, can be achieved.

The main contributions of our work are listed as following. First, we develop a quality priority ZQDCT prediction algorithm to decide which DCT mode should be performed. The DCT modes include skipped mode, $2 \times 2$ DCT, $4 \times 4$ DCT, $6 \times 6$ DCT, and $8 \times 8$ DCT. Second, we design a reconfigurable architecture to perform DCT computations adaptively based on our proposed quality priority prediction algorithm. Third, we take advantage of dynamic partial reconfiguration to reduce computation time of $2 \times 2$ DCT and $4 \times 4$ DCT by the proposed multi-block processing mode which is beneficial in throughput increase for high frame-rate and resolution of video sequences [LBM06]. Fourth, we adopt bitstream relocation to reduce the overall bitstream storage size by 79.4%. Experimental results show that using our reconfigurable architecture and quality priority prediction for DCT computations can effectively reduce power consumptions, hardware resources, and computation time with a small quality degradation.

# CHAPTER 2
# RECONFIGURABLE ARCHITECTURE FOR DCT COMPUTATIONS

2-D array based architecture and 1-D decomposition based architecture for scalable DCT computations using dynamic partial reconfiguration are described in this chapter.

## 2.1 DCT Architecture based on 2-D Array

### 2.1.1 Algorithm and Architecture

DCT can be defined by using its transform matrix as shown in (2.1).

$$X = TxT^t \tag{2.1}$$

Equation 2.1 can be modified as Equation (2.2).

$$X = TxT^t = \sum_{m=0}^{7}\sum_{n=0}^{7} x(m,n)T^t_{row}(m)^t \otimes T^t_{row}(n) \tag{2.2}$$

$$T(m,n) = \begin{cases} \sqrt{\frac{1}{N}} & \text{if } m = 0, \\ \sqrt{\frac{2}{N}}\cos\frac{(2n+1)m\pi}{2N} & \text{if } 1 \le m \le N-1 \end{cases} \tag{2.3}$$

where $T_{row}(m)$ is the $m_{th}$ row of transform matrix $T$, as shown in Equation (2.3). $T^t_{row}(m)$ is the $m_{th}$ row of the transposed matrix of $T$. $N = 8$ is used in this work. Operation $\otimes$ is the outer product. Therefore, DCT is performed by using summation of scaled outer product operations.

The top level architecture of our scalable DCT design is shown in Fig. 2.1. The PowerPC is used to implement the proposed configuration manager that provides architectural support through self-reconfiguration to exploit trade-offs among performance, power, and hardware resource utilization. It also prefetches the compressed partial bitstream, decompresses it, and stores it in the BlockRAM within the FPGA.

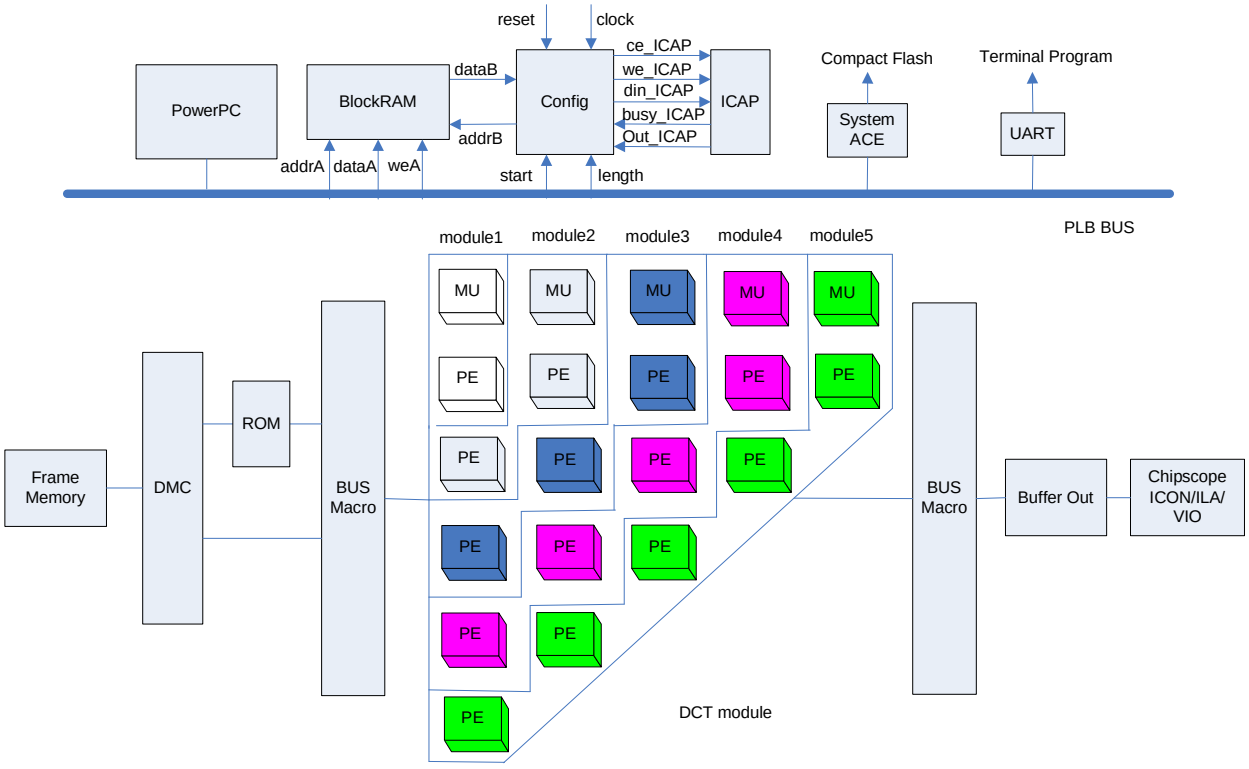



Figure 2.1: Top level architecture of scalable DCT module.

The BlockRAM in this design is a dual port RAM. “dataA” and “dataB” are two data ports of the RAM with 32-bit width. “addrA” and “addrB” are two address ports of the RAM with 14-bit width. The depth of the BlockRAM is made large enough to fit the largest partial bitstream. “weA” is the write enable signal indicating write or read operations of the RAM.

ICAP is Internal Configuration Access Port for Xilinx FPGAs, which can be controlled by internal FPGA logic. It can be used to reconfigure and read back configuration memory. The Config module is the customized hardware module that controls the run-time reconfiguration, i.e., sending the partial bitstream from BlockRAM to ICAP. Its operation is initiated by the embedded processor. Therefore, the Config module to control run-time partial reconfiguration can reduce the loads of the embedded processor and eliminate the overhead of bus communications. The control flow of Config module is shown in Fig. 2.2 [Bay08]. It is generated based on the timing diagram of ICAP reconfiguration. “ce_ICAP” is the chip enable pin of ICAP interface and “we_ICAP” is the write enable pin of ICAP interface. “ce_ICAP” signal must be deasserted when making changes of “we_ICAP” signal. Otherwise, ICAP will enter ABORT process. “din_ICAP” is the data input of ICAP with 32-bit width. “busy_ICAP” is the handshaking signal from ICAP indicating ICAP is busy and can not accept new configuration data. “Out_ICAP” is the output data from ICAP. “start” signal is the control signal to start the reconfiguration process. “length” signal is used to pass the bitstream length information to Config module. “addrA”, “dataA”, “weA”, “start”,

and “length” signals are all generated by PowerPC via PLB Bus using general purpose IO (GPIO).

System ACE is the interface between compact flash card and the FPGA. The initial configuration file and partial bitstream files are stored in the compact flash. System ACE connects the compact flash to the PLB bus. UART is used as a user interface through HyperTerminal.

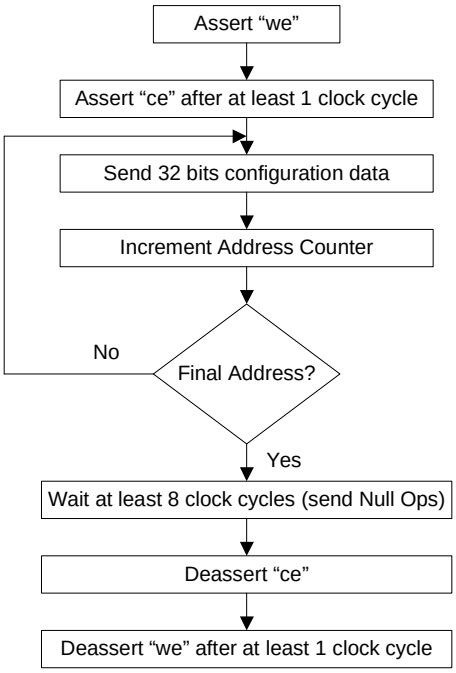



Figure 2.2: Control flow of Config module.

The DCT module is divided into two parts, i.e., static region and reconfigurable region. The static region of DCT module consists of frame memory, data mapper controller (DMC), buffer-out module, and Chipscope ICON/ILA/VIO modules. These modules remain unchanged after initial configuration. DMC is used to fetch the data from frame memory and send them to the array processor. It also generates the address for ROM. ROM

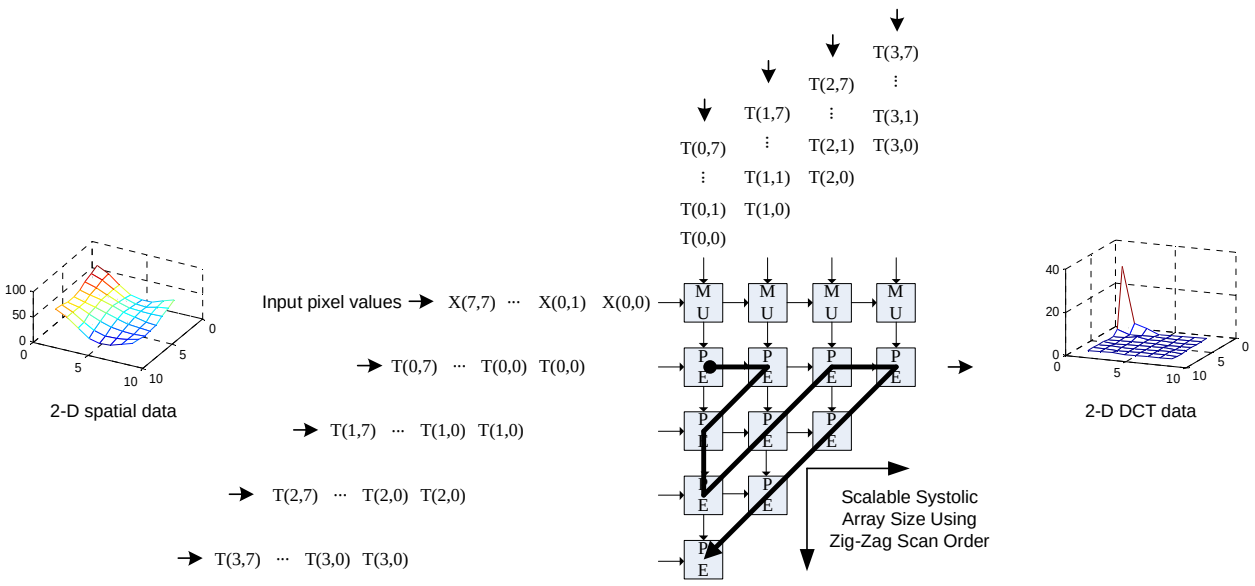



Figure 2.3: Dataflow and architecture of array processor.

stores the transform matrix $T$, as defined in Equation (2.3). Based on the addresses generated from DMC, specific entries of the transform matrix are loaded to the array processor. Buffer-out module buffers the computed DCT coefficients, and outputs them. Chipscope ICON/ILA/VIO are used for verification purposes.

In this work, five modules, i.e., from module1 to module5, are included and tested to implement scalable DCT computation. These five modules can be used to perform $5 \times 5$ DCT to $1 \times 1$ DCT by using five modules together to one module only. For example, if five modules are used, then $5 \times 5$ DCT is performed. If only module1 is used, then $1 \times 1$ DCT is performed. Here, $5 \times 5$ DCT means an $8 \times 8$ DCT with a $5 \times 5$ triangle window of coefficients. 8 point 1D DCT kernel is used for 2D DCT and top left 15 coefficients in zig-zag scan order are calculated. Each module is defined as a partial reconfigurable region (PRR). Bus macros (BM) are used to connect signals between static region and PRR. They

are composed of MUltipliers (MU) and Processing Elements (PE). Each module consists of one MU and the PEs on the diagonal. Fig. 2.3 shows the dataflow of the $4 \times 4$ DCT computations. Here, $x(m, n)$ is the pixel data, and $0 \leq m, n \leq 7$. $T(m, n)$ is the entry of the DCT transform matrix stored in the ROM. The DCT coefficients are first sent to the output buffer after computations. Then, the DCT coefficients are output one by one in the zig-zag scan order from the output buffer. The architectures of MU and PE are illustrated in Fig. 2.4 and Fig. 2.5. The MU consists of two registers and one multiplier. It performs 9 bit $\times$ 12 bit multiplications. "A_in" is for pixel values and "B_in" is for transform matrix entries. The PE is composed of a multiply-accumulator and two registers. It performs 12 bit $\times$ 21 bit multiply accumulations. "C_out" is for output of the computed DCT coefficient. Table 2.1 shows the operations of the PEs.

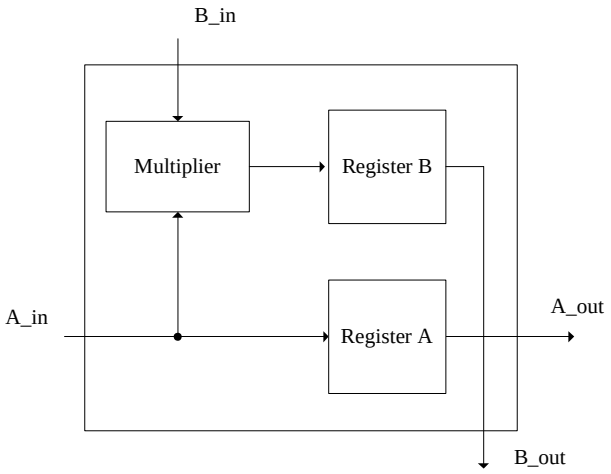



Figure 2.4: Schematic diagram of MU.

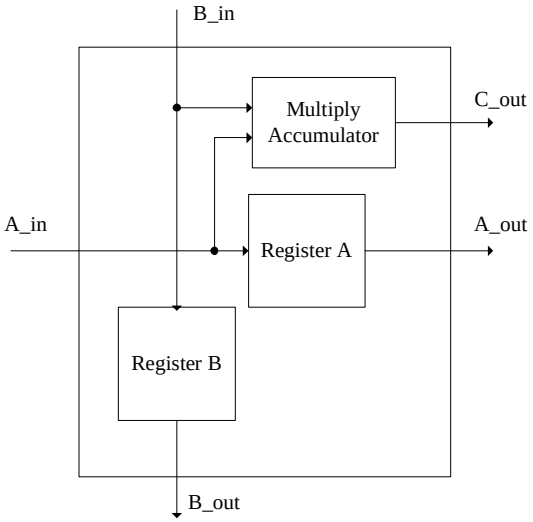



Figure 2.5: Schematic diagram of PE.

Table 2.1: Operations of PEs

| | PE00 | PE01 | PE02 | $\Rightarrow$ |
|---|---|---|---|---|
| 1 | $X^1(0,0) = x(0,0)T(0,0)T(0,0)$ | | | |
| 2 | $X^2(0,0) =$ $X^1(0,0) + x(0,1)T(0,0)T(0,1)$ | $X^1(0,1) = x(0,0)T(0,0)T(1,0)$ | | |
| 3 | $X^3(0,0) =$ $X^2(0,0) + x(0,2)T(0,0)T(0,2)$ | $X^2(0,1) =$ $X^1(0,1) + x(0,1)T(0,0)T(1,1)$ | $X^1(0,2) = x(0,0)T(0,0)T(2,0)$ | $\Downarrow$ |
| 4 | $X^4(0,0) =$ $X^3(0,0) + x(0,3)T(0,0)T(0,3)$ | $X^3(0,1) =$ $X^2(0,1) + x(0,2)T(0,0)T(1,2)$ | $X^2(0,2) =$ $X^1(0,2) + x(0,1)T(0,0)T(2,1)$ | |
| 5 | $X^5(0,0) =$ $X^4(0,0) + x(0,4)T(0,0)T(0,4)$ | $X^4(0,1) =$ $X^3(0,1) + x(0,3)T(0,0)T(1,3)$ | $X^3(0,2) =$ $X^2(0,2) + x(0,2)T(0,0)T(2,2)$ | |

### 2.1.2 Experimental Results and Analysis

Our self-reconfigurable design is implemented in Xilinx Virtex-4 ML410 development system with Virtex-4 FX60 FPGA. Xilinx EDK is used to create the embedded processor system. ISE is used for synthesis process, and Planahead is used for floorplanning, placement, and routing. The compressed partial reconfigurable bitstreams and system.ace files are stored in the compact flash card for configuration. HyperTerminal is used for interacting between

user and the FPGA board. Integrated logic analyzer and virtual IO from Chipscope are used for verification purpose.

Fig. 2.6 and Fig. 2.7 show the PSNR analysis of our scalable DCT computations using H.263 S/W codec. For the experiments, 30 frames of Bridge-far QCIF video sequence are used. The first frame is quantized using intra mode, and the rest of the frames are quantized using inter mode. As shown in Fig. 2.6 and Fig. 2.7, different DCT mode can be selected through the configuration manager when the required PSNR at the given compression ratio is provided at the system level. For example, if QP is set to 15, and PSNR above 30 dB is required, it is indicated that DCT mode using $3 \times 3$ or higher configuration is necessary to achieve the minimum quality of reconstructed video sequences determined at the system level.

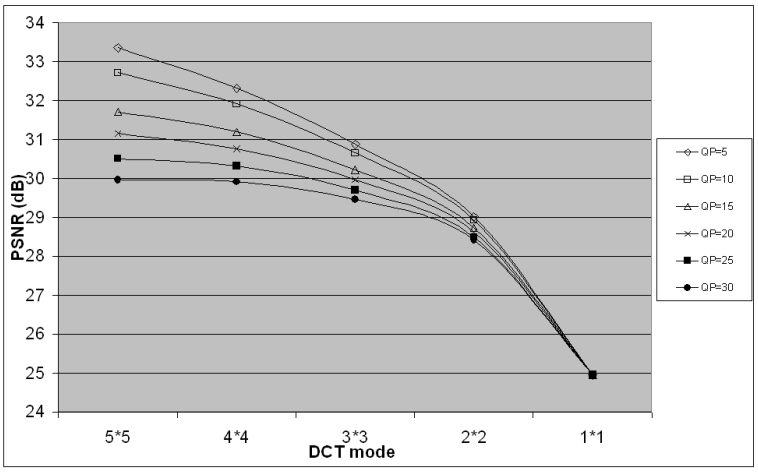



Figure 2.6: Intra mode PSNR analysis.

The power consumption of our scalable DCT is shown in Fig. 2.8. We divide the power consumptions into two parts, i.e., the partial reconfiguration region and the static region.

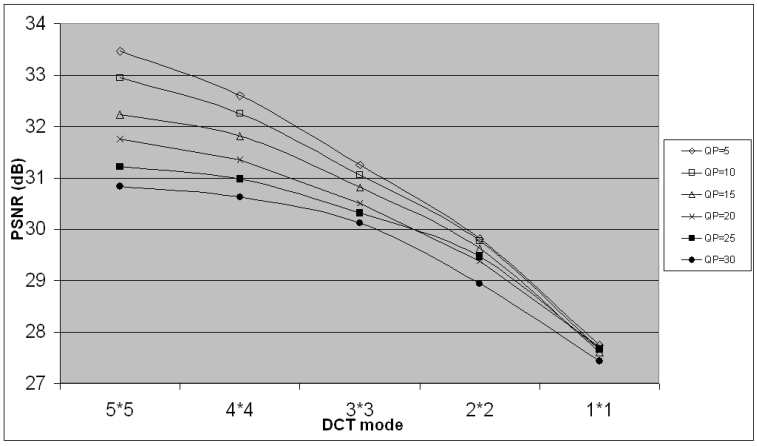



Figure 2.7: Inter mode PSNR analysis.

The power consumption can be controlled by selecting different number of modules for PRR regions. For example, the power consumption for PRR is reduced by 84.5% while reconfiguring $5 \times 5$ DCT to $1 \times 1$ DCT. Fig. 2.8 shows the processing time for one $8 \times 8$ block pixel data changing from 71 clock cycles to 67 clock cycles for different DCT modes. Therefore, run-time self-reconfiguration enables our proposed architecture more adaptable in terms of the power consumption and processing time compared to those fixed architectures for DCT computations.

Table 2.2 shows the comparisons of our proposed scalable DCT architecture with other DCT implementations. Compared with other implementations, our proposed implementation for DCT is more flexible in terms of area, power, and throughput.

Virtex-4 devices provide a 32-bit wide 100 MHz ICAP interface for fast reconfiguration. It has a reduced configuration granularity, i.e., 16 CLBs in height, compared to Virtex-II and Virtex-II Pro devices. Xilinx used a bitstream compression algorithm based on LZ77

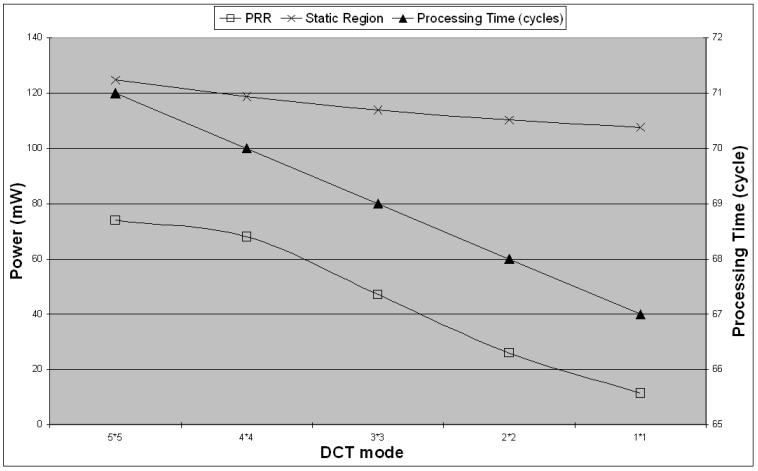



Figure 2.8: Power estimation and throughput analysis.

Table 2.2: Comparisons with other DCT implementations

| | [CJC00] | [GVB05] | [GHC04] | Proposed |
|---|---|---|---|---|
| Area (Gate Counts) | 38004 | 18590 | 52143 | 1506-6570 |
| Operating cycles (Cycles/$8 \times 8$ Block) | 64 | N/A | 64 | 64 |
| Latency | 198 | N/A | 1 | 3-7 |
| Transpose Memory | yes | yes | no | no |
| Clock Frequency (MHz) | 100 | 50 | 125 | 100 |
| Technology | 0.6 $\mu m$ | FPGA (90 nm) | 0.25 $\mu m$ | FPGA (90 nm) |
| Functions | $8 \times 8$ DCT | $8 \times 8$ DCT | $8 \times 8$ DCT | $1 \times 1$ DCT-$5 \times 5$ DCT |
| Power (mW) | 138 | 281 | N/A | 119.1-198.72 |

scheme [Khu01]. LZ77 is a dictionary based data compression algorithm developed by Abraham Lempel and Jocob Ziv in 1977 [ZL77]. It attempts to replace a string of symbols with a reference to a dictionary location of the same string, even strings with no match. In no match case, it takes more space to encode the data. Storer and Szymanski proposed an improved LZSS algorithm based on LZ77. It uses one flag bit to indicate whether the next chunk of data is an offset/length pair or uncoded data. Its decoding steps are shown as follows [Dip]:

- Step 1. Initialize the dictionary to a known value

- Step 2. Read the encoded/not encoded flag.
- Step 3. If the flag indicates an encoded string:
  - Step 3a. Read the encoded length and offset, then copy the specified number of symbols from the dictionary to the decoded output.
  - Step 3b. Otherwise, read the next character and write it to the decoded output.
- Step 4. Shift a copy of the symbols written to the decoded output into the dictionary.
- Step 5. Repeat from Step 2, until all the entire input has been decoded.

The overall compressed bitstream size for five modules implemented in this work is 86.8 KB including 5 blank partial bitstreams and 5 functional partial bitstreams, as shown in Table 2.3. Blank partial bitstream can configure the PRR with no switching activities to reduce dynamic power consumption. The original partial bitstreams' size is 258 KB. Using compressed partial bitstreams can reduce the bitstream size by 66.4%. Table 2.3 also shows comparisons of external memory accesses and time to load original partial bitstreams and compressed partial bitstreams. Everytime the PowerPC accesses the CF card, 512 bytes of configuration data are fetched. Using compressed partial bitstreams can save the number of external memory accesses by 69% in average. In addition, using compressed partial bitstreams can reduce the loading time by 34.6% in average. Time Load O. is the time to load the original bitstreams from the CF card to the BlockRAM directly using PowerPC. Time Load C. is the time to load the compressed bitstreams from the CF card

and decompress/store them into the BlockRAM through PowerPC. The hardware resources required to implement the Config module and the BlockRAM module are shown in Table 2.4.

The decompression latency can be ignored because the decompression process is performed while DCT computations are being performed. The initial system.ace file size is 2.72 MB. Once the FPGA is powered on, it is configured with the initial configuration bitstream. If we use non-partial reconfiguration approach, we will need five ace files which will consume 13.6 MB to implement each of five DCT modules. Using dynamic partial reconfiguration can save the configuration file sizes by 79.4%.

Table 2.3: Comparisons of original bitstreams and compressed bitstreams

| | Module1/Blank1 | Module2/Blank2 | Module3/Blank3 | Module4/Blank4 | Module5/Blank5 |
|---|---|---|---|---|---|
| Original (bytes) | 14816/5760 | 21956/10432 | 34956/22828 | 40784/21492 | 53076/38380 |
| Compressed (bytes) | 5096/1095 | 8043/2030 | 13960/5471 | 15714/3817 | 23001/10685 |
| # of accesses O. | 29/12 | 43/21 | 69/45 | 80/42 | 104/75 |
| # of accesses C. | 10/3 | 16/4 | 28/11 | 31/8 | 45/21 |
| Accesses Saving. | 66.5%/75% | 62.8%/81% | 59.4%/75.6% | 61.3%/81% | 56.7%/72% |
| Reconfig. Time (us) | 37.02/14.40 | 54.89/26.08 | 87.39/57.07 | 101.96/53.73 | 132.69/95.95 |
| Time Load O. (ms) | 82.8/43.3 | 115.5/62.7 | 172.6/119.4 | 196.8/110.6 | 252.9/186.3 |
| Time Load C. (ms) | 43.7/16.2 | 84.1/45.1 | 130/78 | 144.6/69.7 | 193.4/122.9 |
| Load Time Reduced (ms) | 47.2%/62.6% | 22.3%/28% | 24.7%/34.7% | 26.5%/37% | 23.5%/34% |

Table 2.4: Hardware resources

| | Slice Flip Flops | LUTs | RAM16 |
|---|---|---|---|
| Config & BlockRAM modules | 18 out of 50560 | 31 out of 50560 | 25 out of 232 |

The reconfiguration process is shown in Fig. 2.9. First, the initial configuration bitstream with empty BlockRAM is generated. Then, partial bitstreams for different designs are generated. Next, the initial configuration bitstream and the partial bitstreams are stored on the compact flash. Compact flash is used to store the bitstreams externally. When the

FPGA is powered on, it will be configured with the initial bitstream. Then, the compressed partial bitstream is prefetched from the CF card and decompressed by the PowerPC. Then, decompressed partial bitstream is stored in the BlockRAM within the FPGA. Config module can perform the dynamic partial reconfiguration by sending the configuration data from BlockRAM to the ICAP interface.

In [BPK08], Braun et al. presented a waveform-like reconfiguration for dynamic partial reconfiguration. It tries to decrease the overhead of reconfiguration by carefully dividing the reconfiguration modules according to the specific data graph. Therefore, some parts of the data graph start processing while the following parts of the data graph are still being reconfigured. This approach is used to prevent data from being stalled and waiting for the reconfiguration to finish. However, their access to external SRAM memory and uncompressed bitstream size become the limitation to speed up the reconfiguration process. In [CMZ07], Claus et al. proposed a new framework for lowering reconfiguration time using the combitgen tool to reduce the overhead found within the bitstreams. They preload the required bitstreams from CF card into DDR SDRAM during initialization. However, their ICAP controller reading bitstreams from the external SDRAM results in large access time. In [Bay08], Bayar and Yudakul designed a parallel configuration access port core with bitstream decompression module for self reconfiguration. The disadvantage of their approach is to increase reconfiguration overhead due to the concurrent decompression process required during the run-time reconfiguration. In addition, since they need to store all the partial bitstreams in the BlockRAM all the time, this approach will be inefficient when the number

of partial bitstreams is increasing. Using our approach with Virtex-4 FPGA, the reconfiguration time can be improved greatly, as shown in Table 2.5. In our design, while DCT computation is performed, next compressed partial bitstream to be used for increasing or decreasing DCT computation size is loaded and decompressed through PowerPC, and decompressed partial bitstream is pre-stored into the BlockRAM. Therefore, our approach can achieve the maximum reconfiguration speed of Virtex-4 FPGA with the following benefits. First, using on-chip BlockRAM as a configuration cache to pre-store the partial bitstream can remove all the latency overhead caused by the PLB bus operations and the time for accessing external memory since these operations are performed during the DCT computation. Second, BlockRAM size can be reduced since only one partial bitstream to adjust DCT computation size is pre-stored into the BlockRAM.

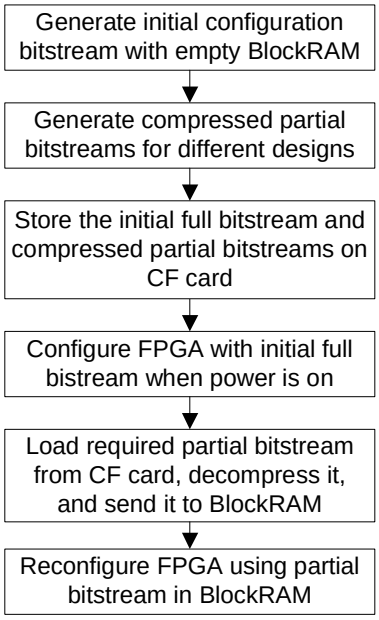



Figure 2.9: Reconfiguration process.

Table 2.5: Reconfiguration time comparisons

| | [BPK08] | [CMZ07] | [Bay08] | Proposed |
|---|---|---|---|---|
| Reconf. Rate (MB/s) | 5.1 | 94.88 | 50 | 400 |
| Device | Virtex 2-2000 | Virtex-II Pro | Spartan-3S200 | Virtex-4 FX60 |
| ICAP Frequency | 66 MHz | 100 MHz | 50 MHz | 100 MHz |

In this work, we combine functional scalability at the algorithm level with hardware architectural scalability through the proposed configuration manager. The user can specify the requirements of the DCT computations, i.e., power consumption, processing time, and quality. Then, the embedded processor chooses optimal DCT configuration mode and reconfigures the FPGA continuously. For example, if the user wants the PSNR to be higher than 33 dB when QP is 5, the configuration manager will choose $5 \times 5$ diagonal DCT configuration based on Fig. 2.6 and Fig. 2.7. If the user wants the overall power consumption to be less than 136 mW, the configuration manager will choose $1 \times 1$ diagonal DCT configuration based on Fig. 2.8. Fig. 2.10, Fig. 2.11, and Fig. 2.12 show trade-off examples through the PSNR, compression ratio, and power analysis results among 8x8 DCT, 5x5 DCT, and 4x4 DCT. Using $5 \times 5$ DCT and $4 \times 4$ DCT can save power consumption by 70.4% and 72.3% respectively with about 2-3 dB degradation of PSNR at the similar bitrate when compared to $8 \times 8$ DCT mode.

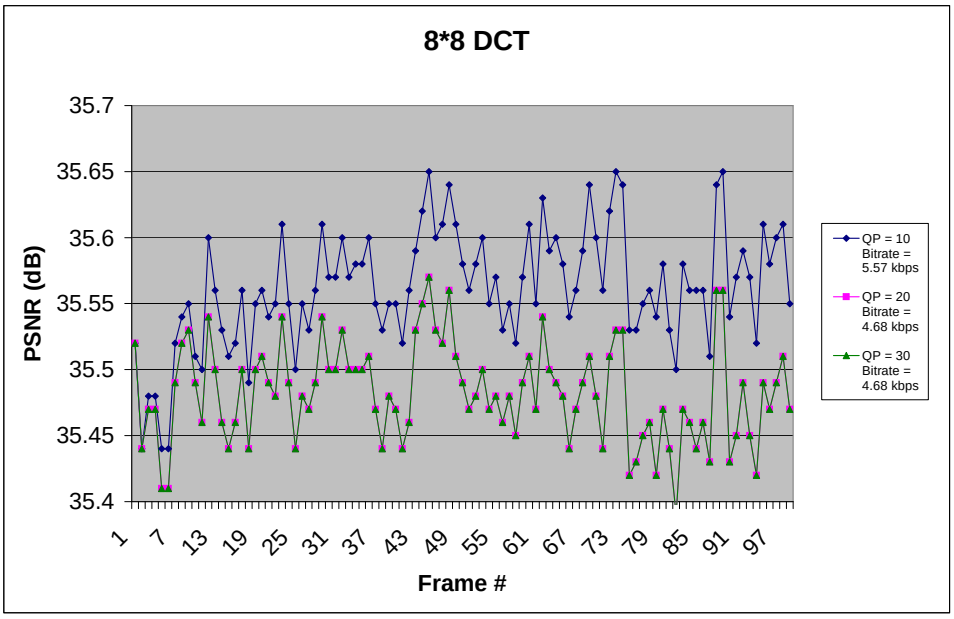



Figure 2.10: Performance analysis of 8*8 DCT

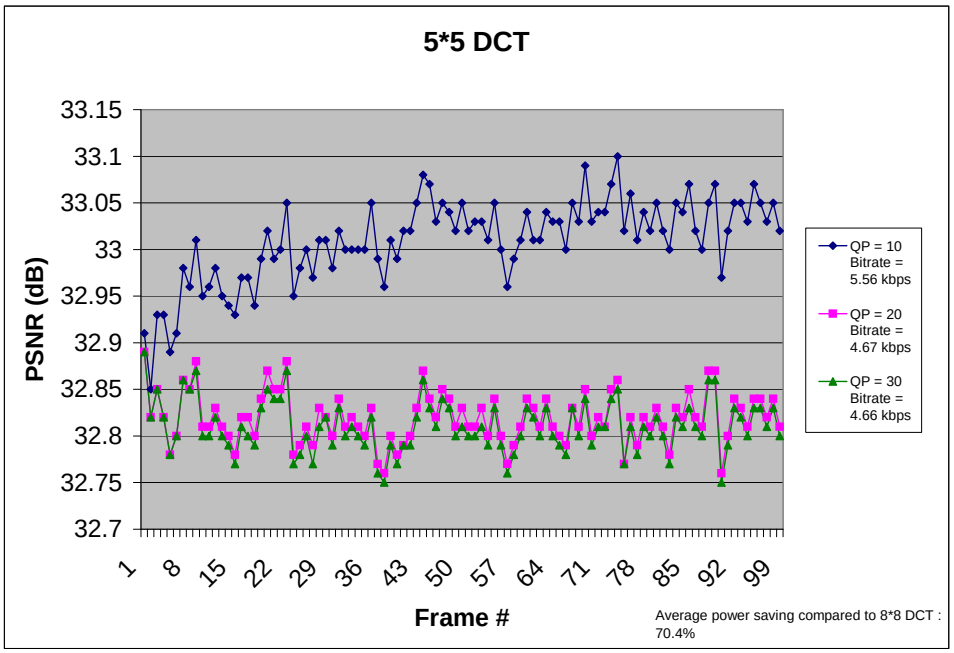



Figure 2.11: Performance analysis of 5*5 DCT

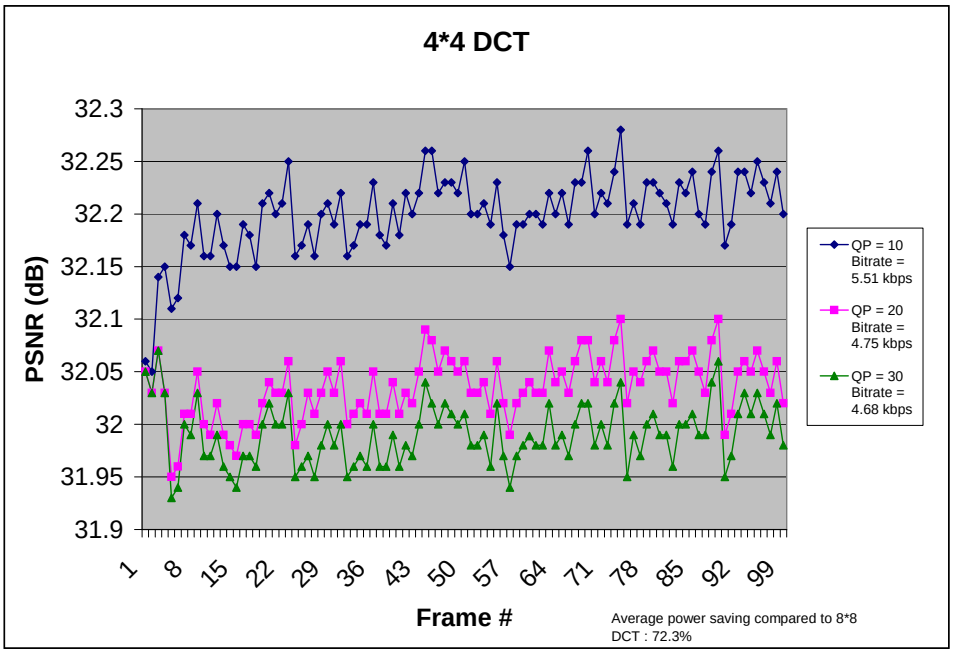



Figure 2.12: Performance analysis of 4*4 DCT

## 2.2 DCT Architecture based on 1-D Decomposition

### 2.2.1 Algorithm and Architecture

2-D DCT is used in many image/video coding standards. The computational complexity can be reduced by decomposing the 2-D DCT into two 1-D DCT computations together with a transpose memory. 1-D DCT is first performed row by row on the input data and the results are saved in a transpose memory. Then, 1-D DCT is performed column by column on the results stored in the transpose memory. The outputs obtained from the second 1-D DCT are the coefficients of the 2-D DCT. We use Chen's algorithm for the 1-D DCT implementation [CSF77], as shown in (2.4).

$$\begin{pmatrix} F(0) \\ F(2) \\ F(4) \\ F(6) \end{pmatrix} = \frac{1}{2} \begin{pmatrix} A & A & A & A \\ B & C & -C & -B \\ A & -A & -A & A \\ C & -B & B & -C \end{pmatrix} \begin{pmatrix} f(0)+f(7) \\ f(1)+f(6) \\ f(2)+f(5) \\ f(3)+f(4) \end{pmatrix}$$

$$\begin{pmatrix} F(1) \\ F(3) \\ F(5) \\ F(7) \end{pmatrix} = \frac{1}{2} \begin{pmatrix} D & E & F & G \\ E & -G & -D & -F \\ F & -D & G & E \\ G & -F & E & -D \end{pmatrix} \begin{pmatrix} f(0)-f(7) \\ f(1)-f(6) \\ f(2)-f(5) \\ f(3)-f(4) \end{pmatrix} \tag{2.4}$$

where $A = \cos\frac{\pi}{4}, B = \cos\frac{\pi}{8}, C = \sin\frac{\pi}{8}, D = \cos\frac{\pi}{16}, E = \cos\frac{3\pi}{16}, F = \sin\frac{3\pi}{16}$ and $G = \sin\frac{\pi}{16}$.

Distributed arithmetic (DA) is a bit-serial operation that performs inner-product computations. The main disadvantage of DA is the latency due to the bit-serial fashion. However, it can be improved by implementing it in a parallel fashion. We implement the DA architecture in a parallel fashion to improve the performance [Whi89]. An example of an inner product computation is shown in (2.5).

$$y = \sum_{k=1}^{K} A_k x_k \tag{2.5}$$

Here, $A_k$ is a fixed coefficient , and $x_k$ is the input data. Using 2's complement binary representation, $x_k$ can be expressed as (2.6).

$$x_k = -b_{k,M-1}2^{M-1} + \sum_{n=0}^{M-2} b_{k,n}2^n \tag{2.6}$$

Each $x_k$ is of $M$-bits, and $b_{k,M-1}$ is the sign bit of $x_k$. $b_{k,n}$ is the $n$th bit of $x_k$, and $b_{k,0}$ is the least significant bit of $x_k$. Combing (2.5) and (2.6) together, the inner product computation can be modified as (2.7).

$$
\begin{aligned}
y &= \sum_{k=1}^{K} A_k \left[ -b_{k,M-1} 2^{M-1} + \sum_{n=0}^{M-2} b_{k,n} 2^n \right] \\
&= \sum_{n=0}^{M-2} \left[ \sum_{k=1}^{K} A_k b_{k,n} \right] 2^n + \sum_{k=1}^{K} A_k \left( -b_{k,M-1} \right) 2^{M-1}
\end{aligned} \tag{2.7}
$$

The computation $\sum_{k=1}^{K} A_k b_{k,n}$ only has $2^K$ possible results since $b_{k,n} \in \{0, 1\}$. These results can be pre-computed and stored in the ROM. The input data $b_{k,n}$ can be used as the address for the ROM. Then, the whole inner product computation can be completed using the ROM and a shift accumulator.

We use motion estimation as an example to demonstrate the usage of unused reconfigurable PEs in this work. So we briefly describe the algorithm for motion estimation here. Motion estimation is used to reduce temporal redundancy in the video coding. We use sum of absolute difference (SAD) as the matching criterion for block matching motion estimation. The SAD computation is defined in (2.8).

$$
SAD(\Delta x, \Delta y) = \sum_{u=0}^{P-1} \sum_{v=0}^{P-1} |f_{curr}(u, v) - f_{pred,\Delta x,\Delta y}(u, v)| \tag{2.8}
$$

where $f_{curr}$ is the current block, and $f_{pred}$ is the predicted block in the search area. $(\Delta x, \Delta y)$ is the displacement of the predicted block with minimum SAD. $P$ is the size of $f_{curr}$ and $f_{pred}$.

The top level architecture for scalable DCT is shown in Fig. 2.14. These reconfigurable areas can be dynamically reconfigured using partial reconfiguration. There are two ways of reconfiguration of the PEs to achieve quality scalability of DCT computations. First,

the PEs can be removed or added using dynamic partial reconfiguration to achieve zonal coding for the DCT coefficients, as illustrated in Fig. 2.13. Second, the internal logic of PEs can also be changed through dynamic partial reconfiguration to reduce the precision of resultant DCT coefficients. This is reasonable because quantization truncates some of the LSB bits of DCT coefficients. There are two main advantages of our scalable architecture using dynamic partial reconfiguration. First, the DCT computations are not interrupted when switching from different zones or different precisions. Second, the unused PEs can be utilized by reconfiguration for other functions, such as motion estimation computation.

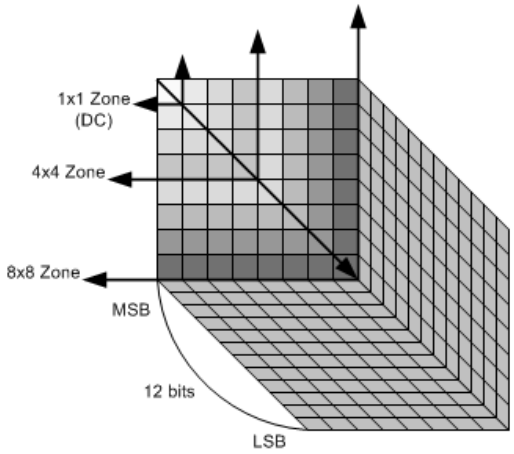



Figure 2.13: Different zones for DCT computation

The controller is used to generate the address and control signals for data fetching and data assigning. The shared memory, data mapping, and butterfly addition/subtraction are combined together. Mapping block is to read the data from the memory or write the data to the memory according to the address and control signals generated by the controller module.

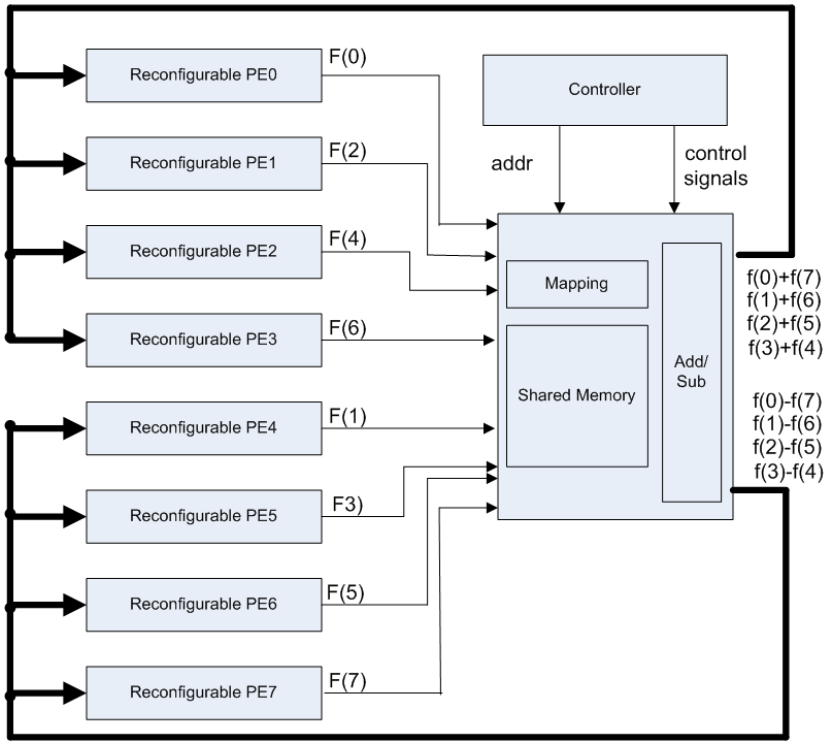



Figure 2.14: Top level architecture of scalable DCT

Addition/subtraction performs butterfly addition or subtraction to generate the input data for the PEs.

The schematic diagram of the reconfigurable PE for DCT computation is shown in Fig. 2.15. We use 13 ROM Shifters (RS) in each PE to parallelize the inner product computations. RS0 accepts the MSB bit plane, and RS12 accepts LSB bit plane. Therefore, 1-D DCT can be performed in one clock cycle. The RS consists of 4 Exclusive-ORs, one 8-word ROM, one adder, and one shifter. The ROM contents and initial register value are shown in Fig. 2.15 . A1, A2, A3, and A4 are four values which can be obtained from each row of the $4 \times 4$ transform matrix in Equation (2.4). Ts is the sign bit timing signal. It is 1 when the input data is sign bit, and 0 for the other bits. The exclusive-or of Ts and x1 controls

the add/subtract computation of the ROM data, i.e., 0 for add, 1 for subtract. The initial register value is added only for LSB bit plane. We also truncate the ROMs to explore the trade-off between the power consumption and the precision.

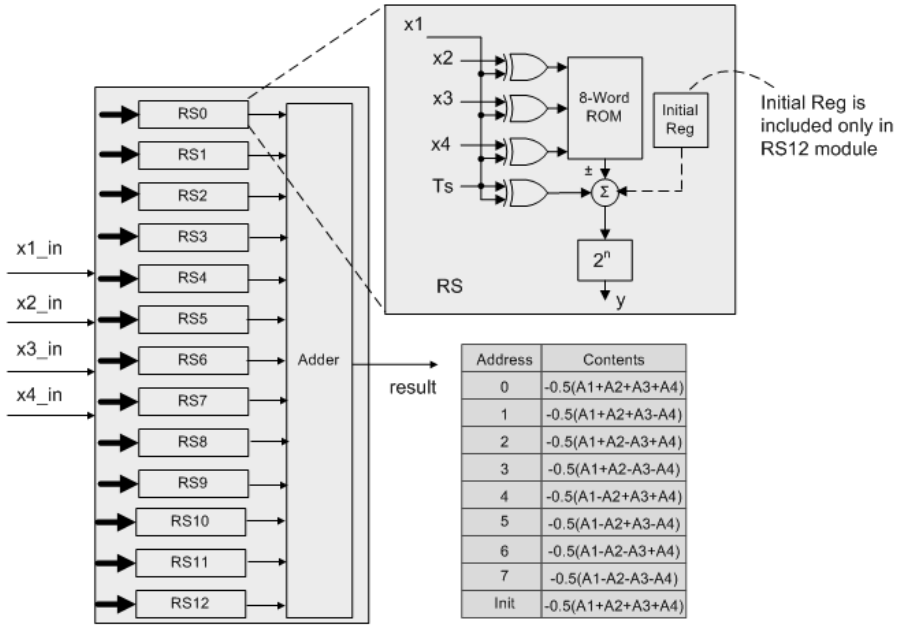



Figure 2.15: Schematic diagram of PE for DCT

We can reconfigure the reconfigurable PE to perform SAD computations for motion estimation. The schematic diagram of the reconfigurable PE for ME is shown in Fig. 2.16. It consists of four Absolute difference units and one adder. Each Absolute difference unit computes absolute difference for one current pixel value and one reference pixel value. The adder accumulates the partial SAD values from Absolute difference units and generates the partial SAD of four pixel values. So each PE can compute the SAD of four current pixel values and four reference pixel values in one clock cycle.

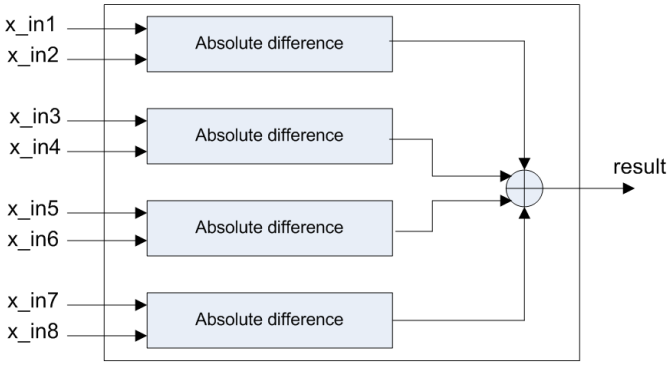



Figure 2.16: Schematic diagram of PE for ME

### 2.2.2 Experimental Results and Analysis

Using the Xilinx Early Access Partial Reconfiguration (EAPR) design flow [Xil06], the scalable architecture is implemented on the Xilinx Virtex-4 SX35 Video Starter Kit. The scalable architecture previously described naturally allows each PE to reside within a separate reconfiguration area for modification of its configuration without disturbing the remaining portion of the FPGA. Fig. 2.17 shows the implementation of the scalable architecture with the locations of the eight reconfiguration areas.

Partial reconfiguration allows flexibility in selecting the quality of precision of a specific PE along with the total number of PEs allocated to the DCT application. Each reconfigurable region is able to implement one PE. In $8 \times 8$ 2D-DCT computations, for example, each reconfigurable area is configured to contain one PE each, totaling 8 PEs. In $1 \times 1$ computations, one reconfigurable area contains one PE while the other 7 reconfigurable areas are made available to other video functions such as motion estimation. In our experiment, four types of PEs are designed: a full precision DCT PE, a partial precision DCT PE, an ME

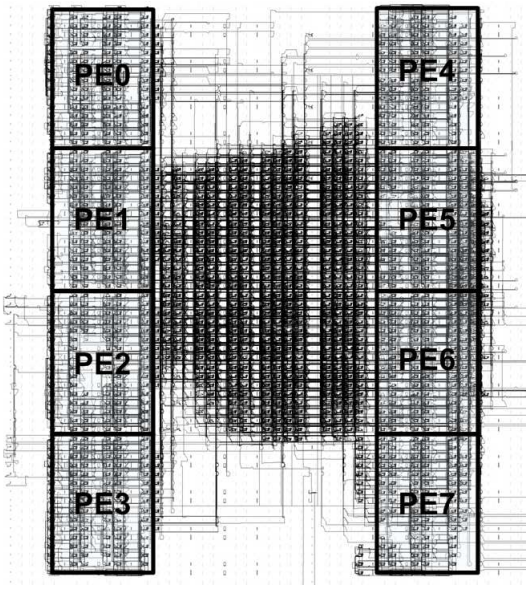



Figure 2.17: Location of 8 PEs on V4SX35

PE, and an Empty PE. The Empty PE allows those unused reconfiguration areas to contain no switching logic to reduce dynamic power consumption.

To prevent the reconfiguration of one area from inadvertently affecting the configuration of another reconfigurable area, no two reconfigurable areas may reside within the same configuration frame of the FPGA device. For example, since configuration frames of Virtex-II devices extend the entire height of the device, no two reconfigurable areas may be stacked vertically. To implement eight partial reconfiguration areas on a Virtex-II device, eight tall and narrow areas must be arranged on the device. However, Virtex-4 FPGAs differ from the Virtex-II and Virtex-II Pro in that the height of its configuration frame is only 16 CLBs [LBM06]. Because of this reduced configuration granularity, reconfigurable areas are allowed to overlap vertically, each residing within its own configuration frame. In the case

of the Virtex-4 SX35 FPGA, which is 96 CLBs in height, up to six reconfigurable areas may be stacked vertically.

For our DCT implementation with eight separate reconfigurable areas, the static logic is arranged in the middle of the FPGA with the PEs on the periphery. As shown in Fig. 2.17, each PE is granted easy access to the static logic. Since the Full Precision PE is the largest of the four configurations, its resource requirements determine the size of the reconfiguration areas. The reconfiguration areas span 16 CLBs in height, whereas the width of each reconfiguration area is minimized to encompass its specific PE design. Table 2.6 lists the hardware resources of the static and the reconfigurable areas, including slices, LUTs (Lookup Table), FFs (Flip Flops), and equivalent gate counts for the Full PE implementation.

Table 2.6: Hardware resource utilization

| | Slices within Area (Slice Utilization) | LUTs within Area (LUTs Utilization) | FFs within Area (FFs Utilization) | Gate Counts |
|---|---|---|---|---|
| Static (Controller, Shared Memory, Mapping, Add/Sub) | 12,416 (16.51%) | 24,832 (15.95%) | 24,832 (3.71%) | 32,067 |
| Reconfigurable (8 PEs) | 2,944 (91.24%) | 5,888 (88.36%) | 5,888 (1.64%) | 41,510 |

As shown in Table 2.7, traditional IP-based implementations on FPGA device have inherent limitations due to their fixed architectures. However, the proposed architecture is highly adaptive in terms of required hardware resources, power consumption, and throughput rate since different number of PEs can be mapped adaptively for DCT and motion estimation algorithms to meet users' requirements. Therefore, our approach can provide more architec-

tural solutions to the users. For comparison purpose, we have implemented the IPs for DCT and motion estimation algorithms separately. DCT is implemented based on the architecture using row-column decomposition and distributed arithmetic, and array-based architecture is used for motion estimation algorithm [CP93].

Table 2.7: Traditional IP comparisons

| | Hardware Resources (gate count) | Power (mW) | Clock Cycles |
|---|---|---|---|
| DCT IP | 59,929 | 26.27 | 102 (required for $8 \times 8$ DCT computation) |
| ME IP | 220,321 | 1002.5 | 721 (required for each MB, search range $(-7 \sim 7)$) |
| Our approach | 73,577 | Depends on PEs' configuration<br>DCT: 24.03-26.27<br>ME: 29.23-131.93 | Depends on PEs' configuration<br>DCT: 25-102<br>ME : 1800-14400 |

A partial bitstream is generated for each reconfiguration area and for each type of PE. For example, 24 partial bitstreams are generated in our implementation of 8 reconfiguration areas and 4 types of PEs. The partial bitstreams generated for each of the Full Precision PEs range from 22,306 bytes to 28,306 bytes. Because the Full Precision PE represents the largest slice utilization, its bitstream sizes are the upper bounds for all types of PEs. For comparison, a bitstream file size of an Empty PE is 10,586 bytes. Before partial bitstreams are used, the FPGA is initialized first with a full bitstream. In designing the initial full bitstream, the user determines the most useful combination of type and number of PEs as the initial configuration of the FPGA- full or partial precision, and $1 \times 1$, $2 \times 2$, etc. The size of the intial bitstream is always 1,712,614 bytes, regardless of whether all 8 Full Precision PEs are implemented or only 1 Full Precision PE with 7 Empty PEs is implemented. In

comparison to a full bitstream, partial bitstreams are significantly smaller, reducing the storage space required to store the various bitstreams. The results show that the file size of a Full Precision PE bitstream is about 1.6% of a full bitstream.

Table 2.8 lists a comparison between one non-partial reconfiguration scenario and two partial reconfiguration scenarios. In the case of non-partial reconfiguration, a full bitstream needs to be generated and stored for each 2D-DCT configuration. For example, a full bitstream of 1,712,614 bytes is required for a $1 \times 1$ Full Precision DCT configuration. To implement an $8 \times 8$ Full Precision DCT function, another full bitstream is required. To implement a $4 \times 4$ Full Precision DCT function with 4 Motion Estimation PEs, a third full bitstream is required. For three distinct hardware arrangements, 4.9 MB of storage space is required. To switch between each of these hardware arrangements, the entire FPGA is reconfigured, stopping all video processing elements. The shortest configuration time needed to switch between hardware arrangements is also the worst at 17 ms. The configuration time is estimated based on the timing of SelectMAP using continuous data loading [Xil05], as shown in (2.9).

$$T_{config} \approx \frac{bytes}{f_{cclk}} \tag{2.9}$$

Here, bytes is the number of bytes of the bitstream stored in the external PROM and the clock frequency for the SelectMap configuration is set to 100 MHz in our estimations.

In an implementation of the scalable architecture using partial reconfiguration, a user stores at least one initial configuration bitstream and all partial bitstreams into an external

Table 2.8: Sizes and configuration times of bitstreams

| | | Bitstream (bytes) | Config. Time (ms) |
|---|---|---|---|
| Non-PR | $1 \times 1$ Full 2D-DCT | 1,712,614 | 17 |
| | $4 \times 4$ DCT & 4 ME PEs | 1,712,614 | 17 |
| | $8 \times 8$ Full 2D-DCT | 1,712,614 | 17 |
| | 3 H/W Arrangements (Best/Worst Config. Time) | 4.9 MB | 17/17 |
| PR | Initial ($8 \times 8$) | 1,712,614 | 17 |
| | 8 Full Precision PEs | 226,448 | 0.28 each |
| | 8 Partial Precision PEs | 226,448 | 0.28 each |
| | 8 Empty PEs | 84,688 | 0.11 each |
| | 16 H/W Arrangements (Best/Worst Config. Time) | 2.1 MB | 0.11/2.24 |
| PR | Initial ($8 \times 8$) | 1,712,614 | 17 |
| | 8 Full Precision PEs | 226,448 | 0.28 each |
| | 8 Partial Precision PEs | 226,448 | 0.28 each |
| | 8 Empty PEs | 84,688 | 0.11 each |
| | 8 Motion Estimation PEs | 226,448 | 0.28 ms each |
| | 88 H/W Arrangements (Best/Worst Config. Time) | 2.3 MB | 0.11/2.24 |

PROM. In calculating the storage requirements, the worst-case Full Precision PE partial bitstream file size- 28,306 bytes- is used for partial bitstream totals. The total space required for implementing the initial bitstream and all three types of 2D-DCT PEs-Full, Partial, and Empty- is approximately 2.1 MB. For this small amount of required storage, 16 distinct hardware arrangements are possible. Switching between these hardware arrangements does not disturb logic residing outside of the reconfiguration areas. The shortest configuration time to switch between arrangements is 0.11 ms by implementing one Empty PE, for example, to switch from $8 \times 8$ DCT to $7 \times 7$ DCT. The longest configuration time is estimated to be 2.24 ms to switch, for example, from $8 \times 8$ Partial Precision to $8 \times 8$ Full Precision.

The addition of eight motion estimation PE bitstreams would only increase the storage requirement by 0.2 MB while increasing the number of possible hardware arrangements from 16 to 88.

We simulate full precision and partial precision PE designs using video sequences with QCIF (Quarter Common Intermediate Format) resolution ($176 \times 144$). We compare the precisions between double precision floating point calculations and hardware calculations using mean square error (MSE). The precision comparisons are shown in Fig. 2.18 . QP is the quantization parameter [Bar96]. While the QP increases, the MSE decreases.

The power estimation comparisons and the throughput comparisons are shown in Fig. 2.19 and Fig. 2.20. The power estimations are performed at 41.79 MHz frequency. Based on our experiments, it takes $2N + 22$ clock cycles to perform the DCT computations for an $N \times N$ zone, and $N \times N$ cycles to output the DCT coefficients. We can see from Fig. 2.18, Fig. 2.19, and Fig. 2.20 that our architecture provides the scalability among precision, power, and throughput.

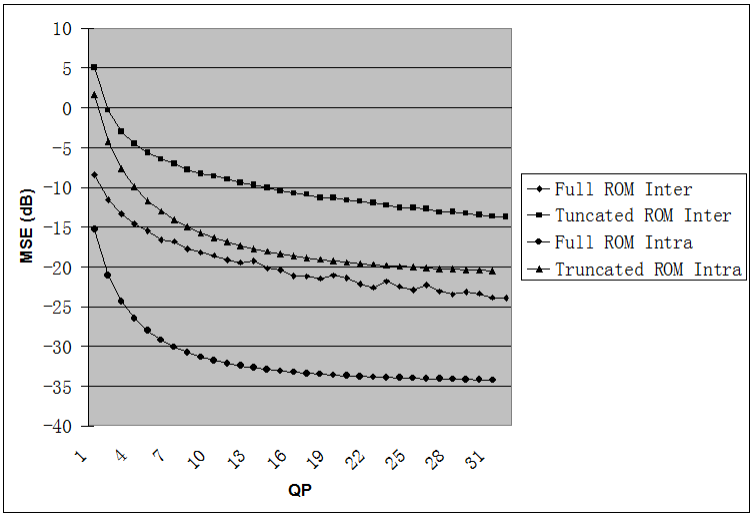



Figure 2.18: Precision comparisons

There are seven modes of motion estimation in H.264/AVC standard, i.e, $16 \times 16$, $8 \times 8$, $4 \times 4$, $16 \times 8$, $8 \times 16$, $8 \times 4$, and $4 \times 8$ [SMW07]. Our reconfigurable PE for motion estimation

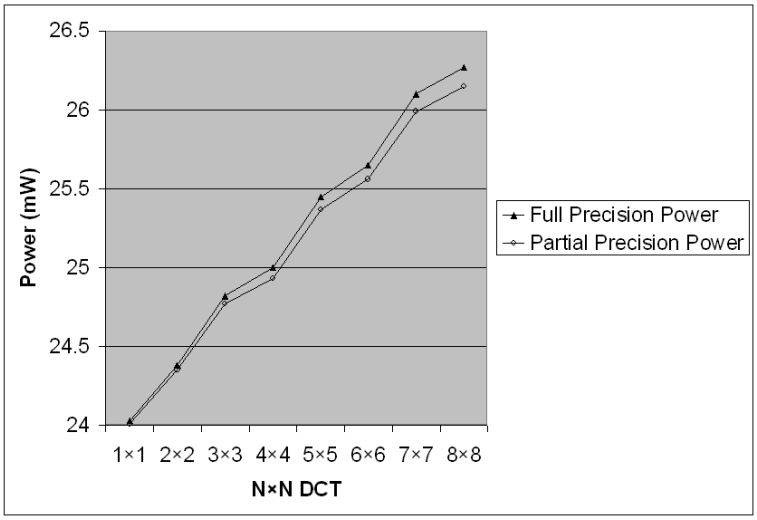



Figure 2.19: Power comparisons

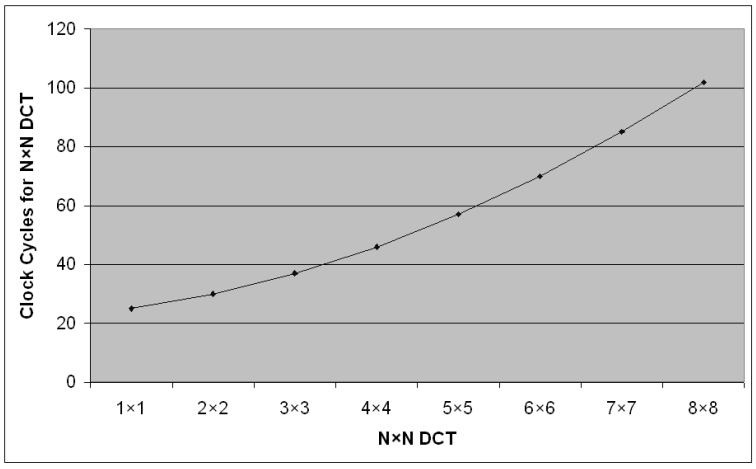



Figure 2.20: Clock cycles for N×N DCT

can be used for any of these modes. Each reconfigurable PE can compute SAD for four pixel values in one clock cycle. If more PEs are used for SAD computations, the throughput of the motion estimation will be increased. Fig. 2.21 shows the clock cycles needed to perform ME per search location for different ME modes.

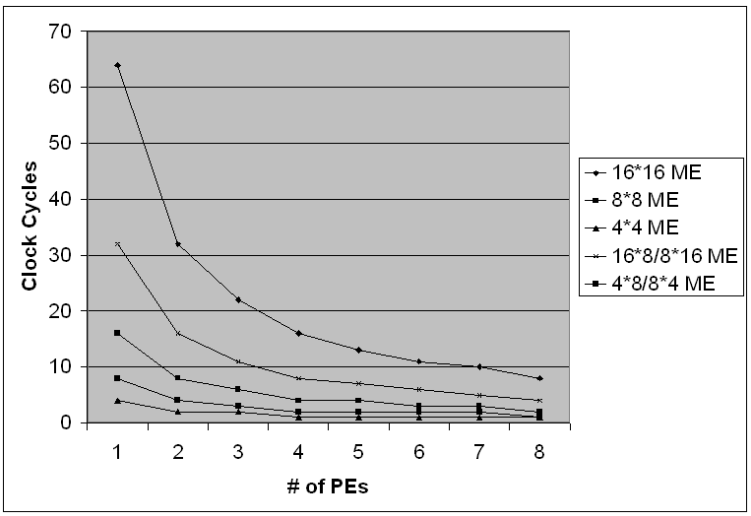



Figure 2.21: Clock cycles for ME per search location

To evaluate the performance of the system, we use QCIF video sequence as an input format with YCbCr 4:2:0. For each frame, there are 594 $8 \times 8$ blocks for DCT and IDCT computations, and 99 $16 \times 16$ macroblocks for ME computations. The number of clock cycles for ME and DCT using different configurations is shown in Table 2.9. Using dynamic partial reconfiguration, ME and DCT can be performed concurrently.

We use $16 \times 16$ motion estimation mode as an example here. There are eight scenarios for performing DCT and ME for each frame using dynamic partial reconfiguration, as shown in Fig. 2.22. First, all of 8 PEs are used for ME. Then, based on the pre-calculated number of clock cycles required to complete $N \times N$ DCT and ME for each frame, partial reconfiguration

Table 2.9: Clock cycles for ME and DCT for different configurations

| N = # of PEs for DCT | # of PEs for ME | Clock cycles for $N \times N$ DCT | Clock cycles for $16 \times 16$ ME | Clock cycles for DCT/frame |
|---|---|---|---|---|
| 0 | 8 | \ | 8 | \ |
| 1 | 7 | 25 | 10 | 14850 |
| 2 | 6 | 30 | 11 | 17820 |
| 3 | 5 | 37 | 13 | 21978 |
| 4 | 4 | 46 | 16 | 27324 |
| 5 | 3 | 57 | 22 | 33858 |
| 6 | 2 | 70 | 32 | 41580 |
| 7 | 1 | 85 | 64 | 50490 |
| 8 | 0 | 102 | \ | 60588 |

starts to change the internal logics of $N$ PEs that will be used to perform $N \times N$ DCT. Here, we can choose $N$ adaptively according to the current compression ratio. $N$ can be set to a small number especially at a high QP since most of high frequency components are likely to be quantized to zeros anyway. The rest of the PEs $(8 - N)$ remain to be used for the completion of the ME operations. Then, $N$ PEs are reconfigured back again to perform ME operations for the next frame.

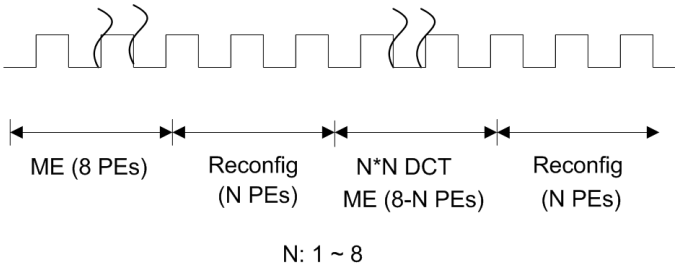



Figure 2.22: Timing diagram for DCT and ME

The clock cycles to perform DCT and ME per each frame for these eight scenarios are shown in Fig. 2.23. The combination of 1x1 DCT and 7 PEs for ME has the lowest clock cycles required to complete these two functions, and the combination of $8 \times 8$ DCT without performing ME concurrently has the highest clock cycles. The reconfiguration overhead

comparisons are shown in Fig. 2.24. Similarly, $1 \times 1$ DCT has the lowest reconfiguration overhead per each frame. The power savings are shown in Figure 2.25. $1 \times 1$ DCT has the best power saving, and $8 \times 8$ DCT consumes most of the power.

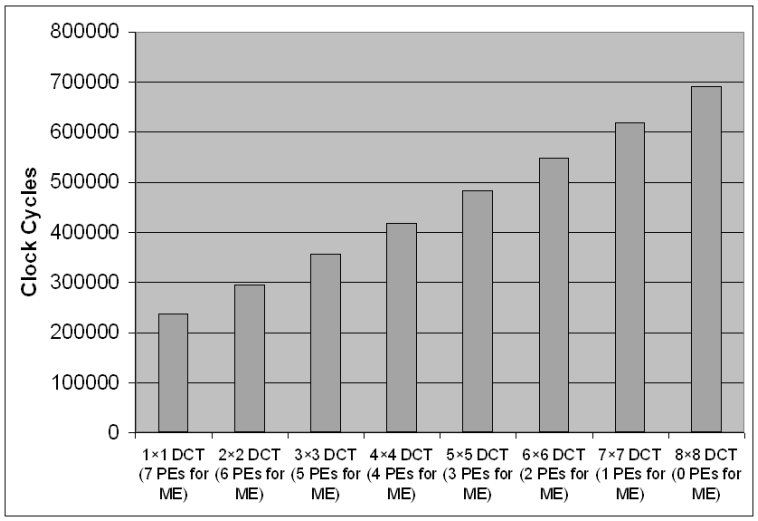



Figure 2.23: Performance comparisons for different combinations of DCT and ME

Reduction of the DCT dimension from $8 \times 8$ to $1 \times 1$ can reduce overall power consumption, total number of clock cycles required for DCT and ME, and the reconfiguration overhead. However, this would increase the possibility of degrading the visual quality of the reconstructed images especially at the low compression ratio. The quality degradations caused by different modes of DCT computations using inter frame coding mode are shown in Fig. 2.26. The PSNR is estimated using Foreman sequence. $8 \times 8$ DCT has the best quality, while $1 \times 1$ DCT has the lowest quality. The quality degradations become smaller when the QP becomes larger.

We use $4 \times 4$ DCT with 4 PEs for ME as an example for analysis. Compared with $8 \times 8$ DCT, the quality degradation of $4 \times 4$ DCT is only by 1.02 dB when QP is set to 16. In the

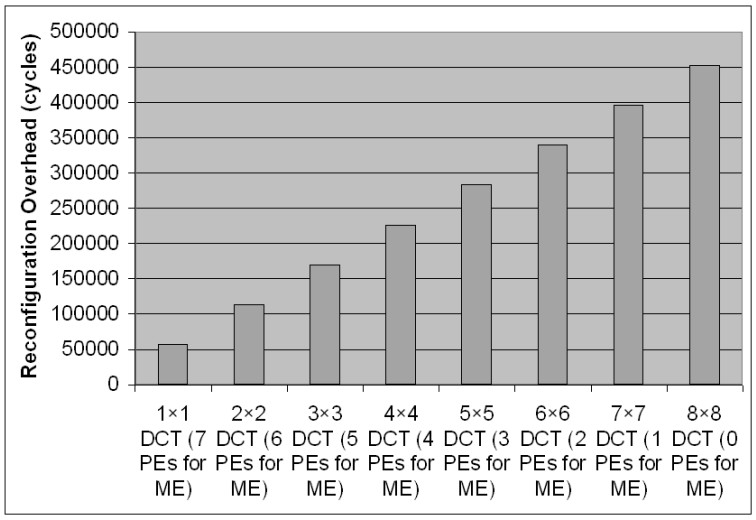



Figure 2.24: Reconfiguration overhead comparisons

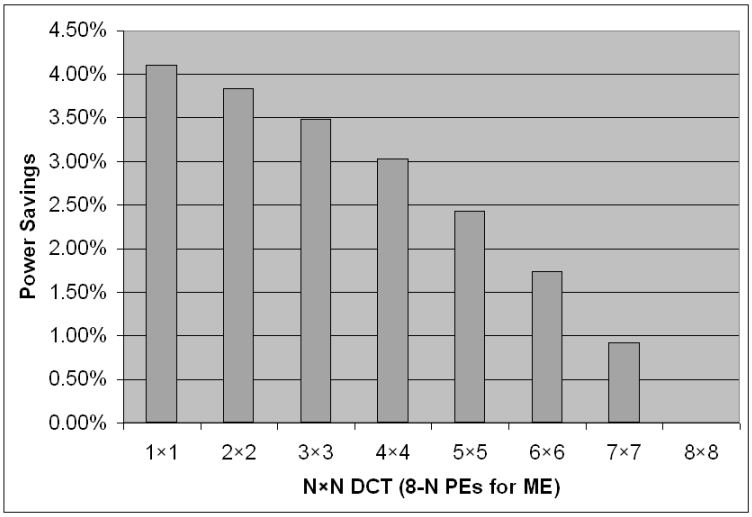



Figure 2.25: Power savings for different combinations of DCT and ME

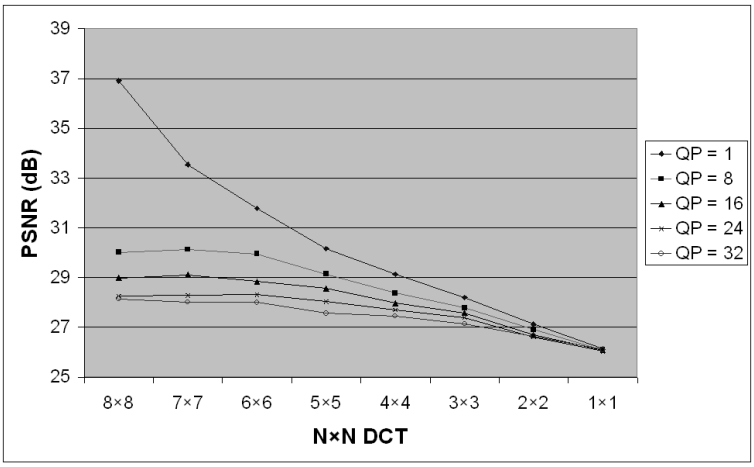



Figure 2.26: PSNR comparisons for different modes of DCT computations

meantime, it saves power consumption by 3.03%, reduce the clock cycles for processing by 39.5%, and reduce the reconfiguration overhead by 50%.

# CHAPTER 3
# EFFICIENT VLSI ARCHITECTURE FOR VIDEO TRANSCODING

## 3.1 Background and Related Works

### 3.1.1 Scaled Outer Products for DCT and IDCT Computations

DCT can be performed using scaled outer products as described in Equation (2.2). IDCT can be performed using scaled outer products in a similar way. IDCT can be defined by using its transform matrix as shown in Equation (3.1).

$$x = T^t X T \tag{3.1}$$

Equation 3.1 can be modified as Equation (3.2).

$$X = T^t x T = \sum_{m=0}^{7} \sum_{n=0}^{7} X(m,n) T_{row}(m)^t \otimes T_{row}(n) \tag{3.2}$$

where $T_{row}(m)$ is the $m_{th}$ row of transform matrix $T$, as shown in Equation (2.3). $T^t_{row}(m)$ is the $m_{th}$ row of the transposed matrix of $T$. $N = 8$ is used in this work. Operation $\otimes$ is the outer product. Therefore, computation of both DCT and IDCT can be performed by iterative accumulation of scaled outer products. Computational complexity of IDCT can be

reduced significantly since only nonzero DCT coefficients contribute to the computation of the scaled outer products in Equation (3.2).

### 3.1.2 Motion Estimation and Compensation in the DCT Domain

Motion estimation and compensation can be performed in either spatial domain or DCT domain. Spatial domain approach comparing pixel values directly to find the best matching block has been widely used for most of video compression applications due to its simplicity and effectiveness. Also, there have been many studies on DCT domain approaches since only a small number of low frequency coefficients after the transformation of highly correlated video signals can be used for motion estimation process [LVI06] [CC99] [KL98]. Our proposed architecture can perform motion estimation and compensation in the DCT domain which can be very useful for the video transcoding applications.

Vertical shifting matrix $V_i$ and horizontal shifting matrix $H_i$ to support both integer-pixel and half-pixel motion estimation and compensation in the DCT domain can be derived by using a displacement matrix $D_n$ as follows. Here, $I_n$ is an $n \times n$ identity matrix.

$$D_n = \begin{pmatrix} 0 & \vdots & I_n \\ \cdots & \vdots & \cdots \\ 0 & \vdots & 0 \end{pmatrix} \tag{3.3}$$

$$V_0 = V_1 = \frac{1}{2}(D_{8-\lfloor \Delta x \rfloor} + D_{8-\lceil \Delta x \rceil}) \tag{3.4}$$

$$V_2 = V_3 = \frac{1}{2}(D_{\lfloor \Delta x \rfloor} + D_{\lceil \Delta x \rceil}) \tag{3.5}$$

$$H_0 = H_1 = \frac{1}{2}(D_{8-\lfloor \Delta y \rfloor} + D_{8-\lceil \Delta y \rceil}) \tag{3.6}$$

$$H_2 = H_3 = \frac{1}{2}(D_{\lfloor \Delta y \rfloor} + D_{\lceil \Delta y \rceil}) \tag{3.7}$$

$(\Delta x, \Delta y)$ indicates a current search location for motion estimation operation and a motion vector information for motion compensation operation. A predicted/compensated block $\hat{f}$ in the DCT domain can be computed using $(\Delta x, \Delta y)$ and the DCT coefficients of the surrounding blocks, i.e., $\hat{f}_0$, $\hat{f}_1$, $\hat{f}_2$, and $\hat{f}_3$, as shown in (3.8).

$$\begin{aligned} \hat{f} &= \sum_{i=0}^{3} \hat{V}_i \hat{f}_i \hat{H}_i \\ &= \sum_{i=0}^{3} \sum_{m=0}^{7} \sum_{n=0}^{7} \hat{f}_i(m,n) \hat{V}_{ic}(m) \otimes \hat{H}_{ic}^t(n)^t \\ &= \sum_{m=0}^{7} \sum_{n=0}^{7} \sum_{i=0}^{3} \hat{f}_i(m,n) \hat{V}_{ic}(m) \otimes \hat{H}_{ic}^t(n)^t \end{aligned} \tag{3.8}$$

Here, $\hat{f}_i(m,n)$ is defined as $m_{th}$ row and $n_{th}$ column entry of $f_i$. $\hat{V}_{ic}(m)$ is defined as $m_{th}$ column of $\hat{V}_i$, and $\hat{H}_{ic}(n)$ is defined as $n_{th}$ column of $\hat{H}_i$. Therefore, $\hat{H}_{ic}^t(n)^t$ is a transpose of $n_{th}$ column of $\hat{H}_i^t$. As shown in (3.8), the computational complexity of obtaining a predicted/compensated block $\hat{f}$ in the DCT domain can be reduced significantly since outer product term, $\hat{V}_{ic}(m) \otimes \hat{H}_{ic}^t(n)^t$, needs to be computed only for the nonzero scaling factor,

$\hat{f}_i(m,n)$. Due to the energy compaction property of DCT transformation and quantization process, $\hat{f}$ is typically a very sparse matrix.

During the motion estimation process in the DCT domain, we use Sum of Absolute Differences (SAD) as our matching criterion as shown in (3.9). $\hat{f}_{curr}$ is a current input block in the DCT domain and $\hat{f}_{pred}$ is a predicted block from the previous frame. $i$ and $j$ are constrained by the search range parameters.

$$SAD_{tr}(i,j) = \sum_{u=0}^{N-1}\sum_{j=0}^{N-1} |\hat{f}_{curr}(u,v) - \hat{f}_{pred}(u+i,v+j)| \tag{3.9}$$

## 3.2 Design of Unified Architecture

In this section, we propose Wavefront Array-based Processor that can perform DCT, IDCT, motion estimation and compensation in the DCT domain. Based on the derived equations in (2.2), (3.2), and (3.8), we design our unified architecture to perform their key computations. We can see that the key computation for DCT, IDCT, and motion compensation in the DCT domain can be performed by iterative accumulation of scaled outer products, as shown in 3.10.

$$Y^k = Y^{k-1} + sC(m) \otimes R(n) \tag{3.10}$$

Here, $k$ increases from 1 to the total number of iterations required for each algorithm. $Y^0$ is an $8 \times 8$ zero matrix and $s$ is a scalar value. $C(m)$ is a column vector and $R(n)$ is a row vector. Motion estimation in the DCT domain can be performed using SAD computation as shown in (3.9). Our proposed architecture can perform accumulation of scaled outer products and SAD computations.

The schematic diagram of our proposed unified architecture is shown in Fig. 3.1 . It consists of array processor, controller, and ROM. The array processor consists of multiplier (MUL), processing element (PE), output buffer, adder (ADD), and comparator (CMP). The ROM is used to store DCT transformed shifting matrices and DCT transformation matrix. The controller fetches data from the ROM and decides which function the array processor will perform. Based on the specific function, its corresponding dataflow of either DCT coefficients or pixel values entering the array processor is selected by the controller. The output buffer is a parallel-in serial-out buffer. It selects the data from different rows of PE array, and then the Data_out outputs the results one by one. So it takes 8 cycles to store 64 DCT/IDCT results to the output buffer. After storing the results to the output buffer, the PEs can perform computations for new data. The results in the output buffer are sent out in serial. Latency to output the results from the output buffer can be ignored due to the pipelining operations.

The schematic diagram of PE in the array processor is shown in Fig. 3.2. PE can perform different functions such as DCT/IDCT, ME/MC by the control signals. The outer product computation part in the PE consists of a multiplier, an adder, and an accumulation register

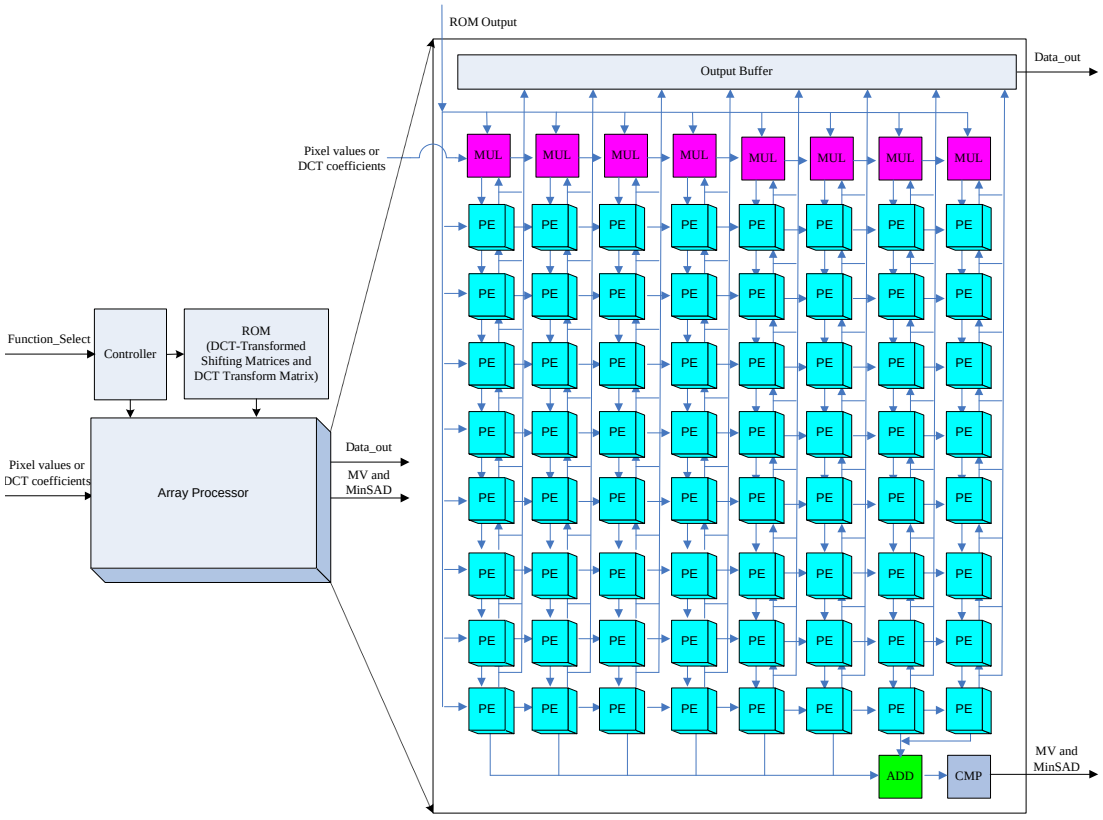



Figure 3.1: Schematic diagram of the proposed unified architecture

(reg_accu) used to store the result of multiply accumulation. The SAD computation part in the PE consists of an adder, a subtractor, an absolute module, and a register (reg_sum) used to store partial SAD value. The reg_ain and reg_bin are used to pass A_in and B_in data to the next PE directly. A_in and A_out have 12-bit widths, and B_in and B_out have 24-bit widths. The final results after scaled outer product computation are stored in the reg_accu registers of all the PEs. Ctrl_Shift signal is used to pass the value in the reg_accu to the PE above through C_in and C_out, which both have 36 bit widths. Start_Accu is used to reset the reg_accu register at the beginning of the outer product computation. Ctrl_Store signal is used to transfer the data from the reg_accu into the reg_curr. The reg_curr is used to store

the current block data in the DCT domain. Ctrl_ME signal is used for passing either B_in or reg_sum to the next PE. The reg_sum is passed to the next PE when the PE is performing SAD computation, and B_in is passed to the next PE when the PE is performing outer product computation. The control signals are pipelined in our design. Each row of the PEs shares the same control signals. So by properly assigning the values for the control signals, ME and one of DCT/IDCT/MC computations can be performed concurrently.

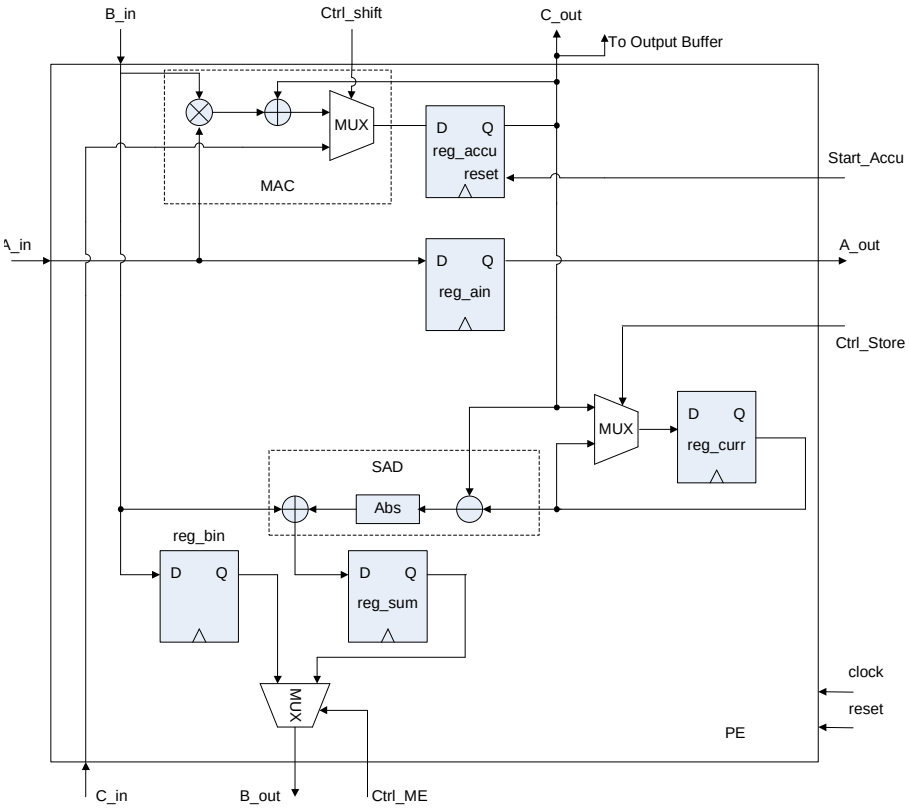



Figure 3.2: Schematic diagram of PE

The schematic diagram of MUL is shown in Fig. 3.3. It multiplies two 12-bit operands (A_in, B_in) and outputs a 24-bit result (B_out). A_out passes A_in directly to the next

MUL. A_in is used to input the scalar number for the scaled outer product computation in (3.10).

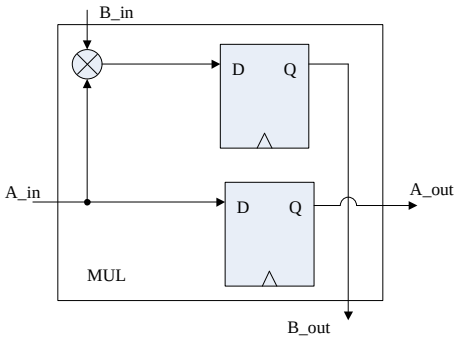



Figure 3.3: Schematic diagram of MUL

The schematic diagram of ADD is shown in Fig. 3.4. It sums the partial SAD passed from the last row of the array processor. The schematic diagram of CMP is shown in Fig. 3.5. It compares the SAD passed from ADD with the previous SAD value stored in the register, and stores the minimum SAD (minsad) together with the motion vector information (reg_x, reg_y). It outputs the motion vector and the minimum SAD value. Our unified architecture can be also extended to perform 1616 motion estimation by adding the minimum SAD values of four $8 \times 8$ blocks to generate the SAD value of $16 \times 16$ block.

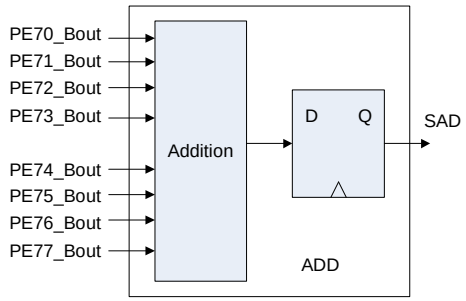



Figure 3.4: Schematic diagram of ADD

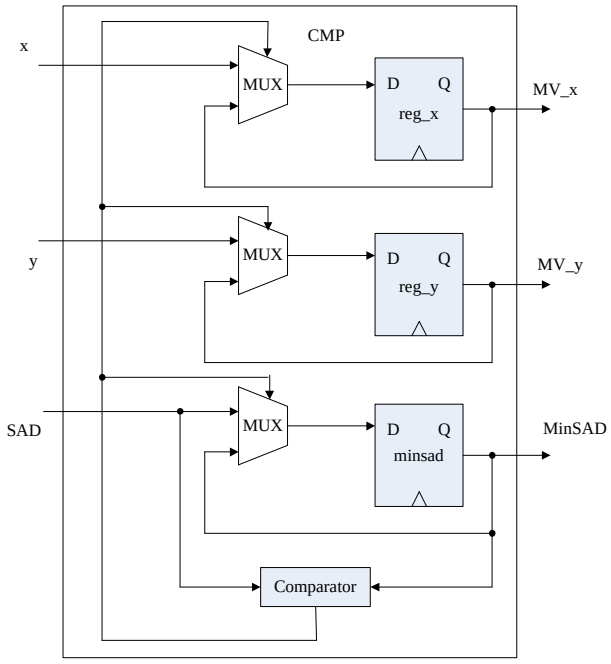



Figure 3.5: Schematic diagram of CMP

## 3.3 Dataflow Characteristics and Performance Evaluations

In this section, we describe how different core algorithms, such as DCT, IDCT, motion estimation and compensation in the DCT domain, are mapped onto the proposed unified architecture effectively by utilizing their dataflow characteristics and arithmetic operations.

Let $X$ have the following nonzero DCT coefficients after quantization process:

$$X = \begin{bmatrix} X(0,0) & 0 & 0 & \cdots & 0 \\ X(1,0) & 0 & 0 & \cdots & 0 \\ 0 & X(2,1) & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 0 \end{bmatrix} \tag{3.11}$$

Here, only $X(0,0)$, $X(1,0)$, and $X(2,1)$ are nonzero coefficients. The dataflow for computing IDCT using this example is shown in Fig. 3.6. IDCT kernels fetched from the ROM enter the top and the left sides of the proposed unified architecture with three nonzero DCT coefficients. Each DCT coefficient is used as a scaling factor in MUL blocks. The data movement to perform the outer product of the column vector $T_{row}(m)^t$ and the row vector$T_{row}$ scaled by the non-zero DCT coefficient $X(m,n)$, propagates from the top-left corner to the bottom-right corner of the $8 \times 8$ PE array. These computational wavefronts corresponding to the mathematical recursions in (3.10) can provide a high throughput rate due to their pipelined operations and sparseness of the 2-D DCT coefficient matrix. For the given DCT coefficient matrix $X$, detailed operations performed on the two representative PEs are shown in Fig. 3.6. Since non-zero DCT coefficients in zig-zag scan order are input to the proposed unified architecture, pipelined computational wavefronts for 2-D IDCT operations can recon-

struct images progressively from low visual quality to high visual quality. Therefore, visual quality can be controlled easily by deciding the number of the mathematical recursions. As shown in Fig. 3.7 and Fig. 3.8, this property can be used to provide a graceful trade-off between visual quality and computational complexity. Additionally, the proposed 2-D IDCT at the decoder can save processing time further by eliminating the process of inserting zeros between non-zero DCT coefficients at the Run-Length Decoder (RLD) block and the process of converting zig-zag scan order into raster scan order. The accuracy of 2-D IDCT computation is also verified according to IEEE Standard 1180-1990 by using data sets of 10,000 blocks with range -256 to 255, -5 to 5, and -300 to 300, as shown in Table 3.1 [ITU96].

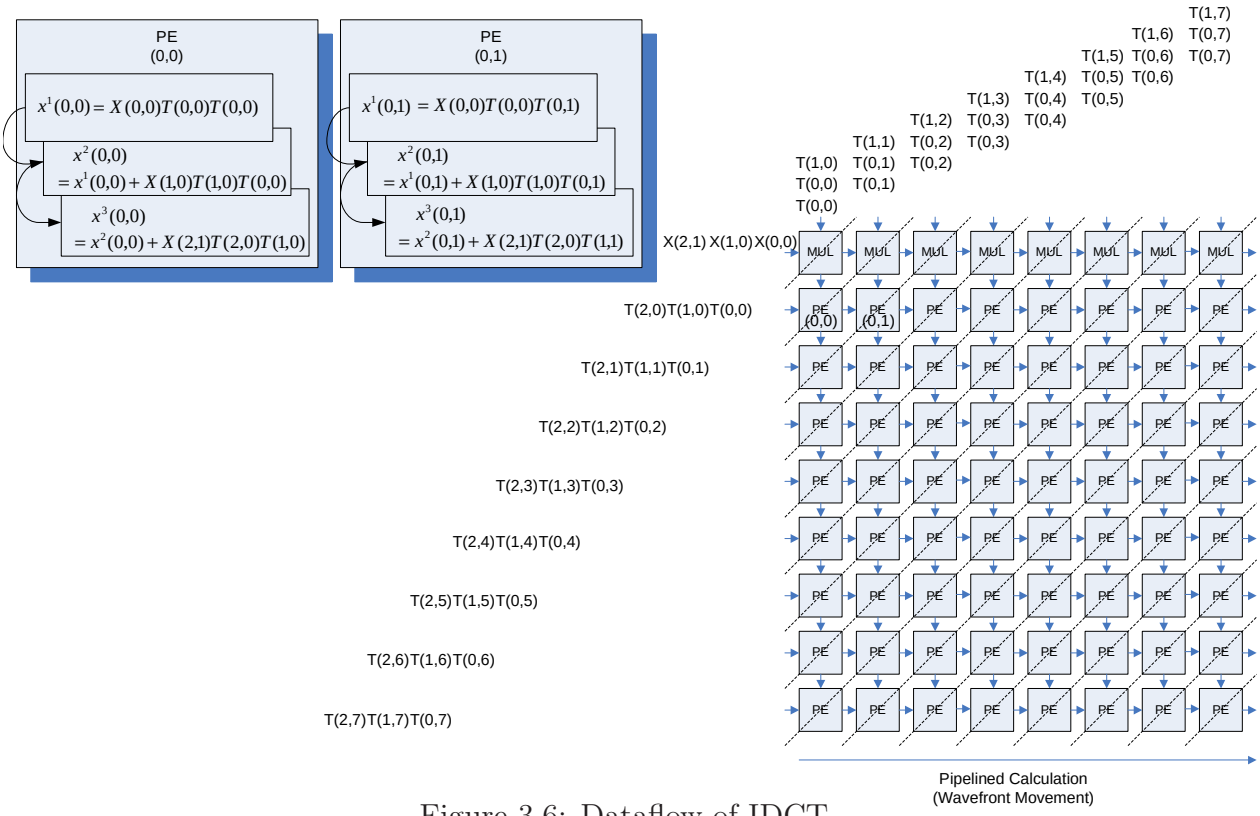



Figure 3.6: Dataflow of IDCT

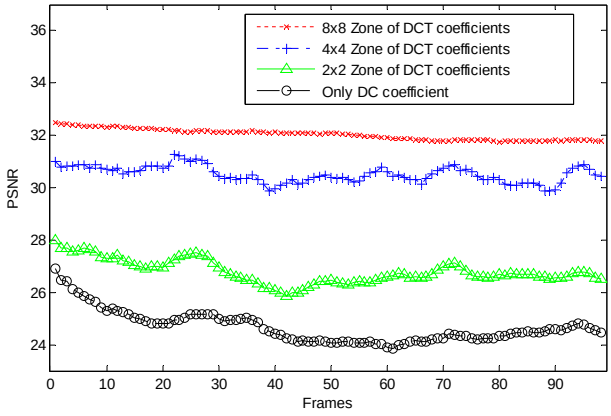



Figure 3.7: PSNR plot using different number of non-zero DCT coefficients in Zig-Zag order (Paris.CIF & QP=10).

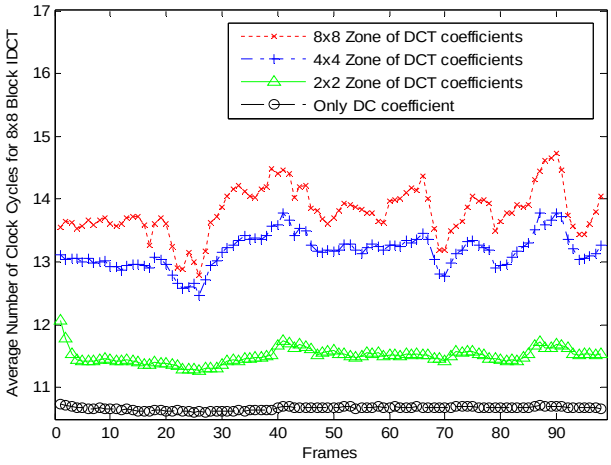



Figure 3.8: Average number of clock cycles for 8x8 block IDCT (Paris.CIF & QP=10).

Let $x$ be the pixel data matrix which is defined as follows:

$$x = \begin{bmatrix} x(0,0) & x(0,1) & \cdots & x(0,7) \\ x(1,0) & x(1,1) & \cdots & x(1,7) \\ \vdots & \vdots & \ddots & \vdots \\ x(7,0) & x(7,1) & \cdots & x(7,7) \end{bmatrix} \tag{3.12}$$

Table 3.1: Accuracy test of the proposed 2-D IDCT

| Pixel Range | PMSE | PME | OMSE | OME |
|---|---|---|---|---|
| -5 to 5 | 0.0015 | 0.0005 | 0.000448 | -0.000017 |
| -256 to 256 | 0.0218 | 0.0054 | 0.01928 | 0.000011 |
| -300 to 300 | 0.0228 | 0.0038 | 0.018872 | 0.000403 |

PMSE: Peak Mean Square Error ($\leq 0.06$)
PME: Peak Mean Error ($\leq 0.015$)
OMSE: Overall Mean Square Error ($\leq 0.02$)
OME: Overall Mean Error ($\leq 0.0015$)

Then, computation of DCT can be performed by using the data flow of DCT kernels shown in Fig. 3.9. Each pixel is used as a scaling factor in MUL blocks and pipelined computation of outer products and accumulation are performed through $8 \times 8$ PE array. The final results stored in these $8 \times 8$ PEs form 2-D DCT coefficient matrix.

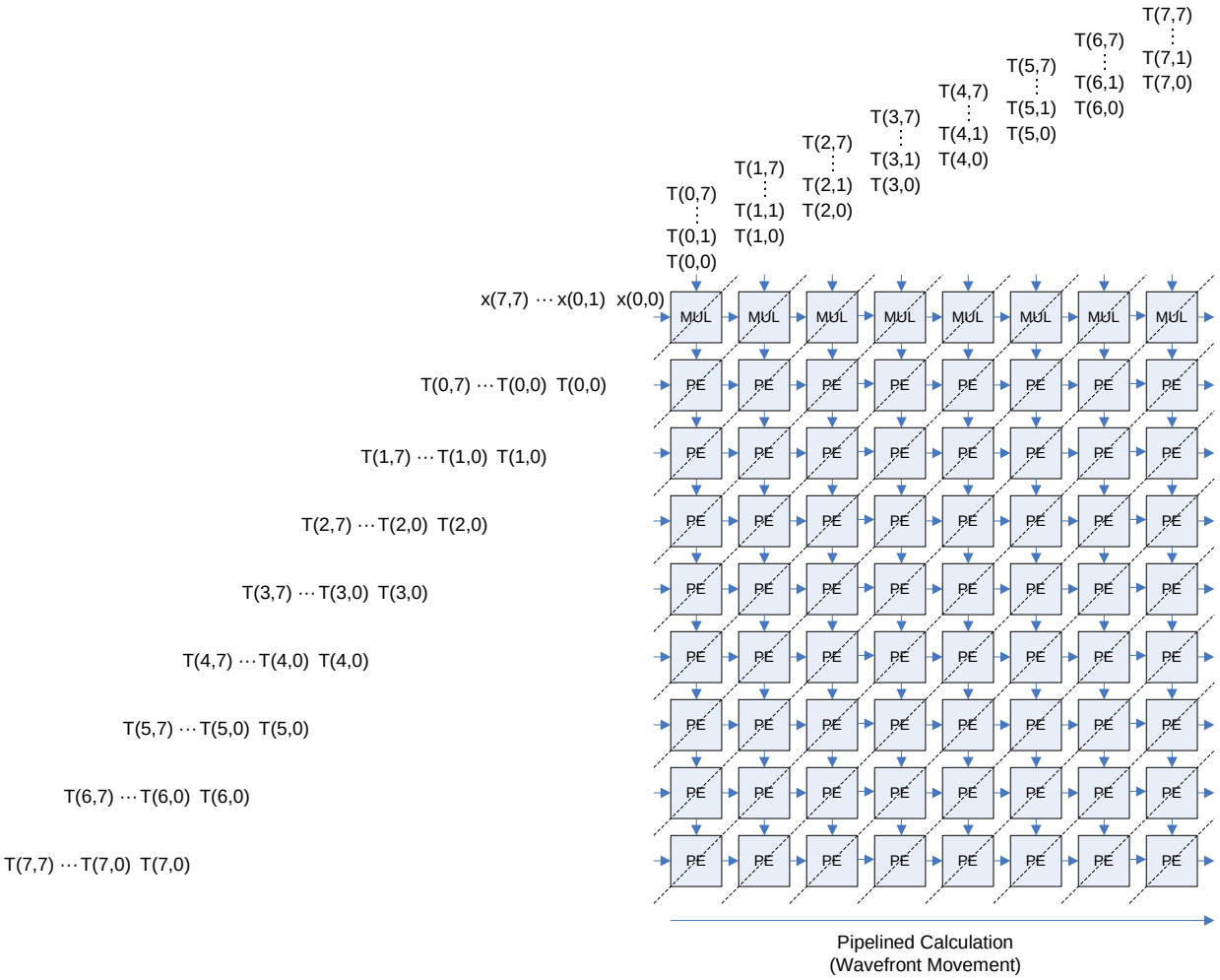



Figure 3.9: Dataflow of DCT

The timing diagram of DCT and IDCT of our unified architecture is shown in Fig. 3.10. We first load the pixel values for DCT computation or non-zero DCT coefficients for IDCT computation into the array processor. After 10 clock cycle latency for the computation, the results can be transferred to the output buffer. Since the computations are pipelined, new input data for DCT or IDCT computations can be loaded into the array processor concurrently while the previous results are transferred to the output buffer. Conventional DCT/IDCT architectures employing a row-column decomposition method often have performance bottleneck due to their transpose memory requiring parallel-to-serial and serial-to-parallel circuits. However, the proposed architecture using direct 2-D algorithm does not require the transpose memory. Therefore, only 10 clock cycle latency is required and high throughput rate can be achieved by pipelining the computational wavefronts.

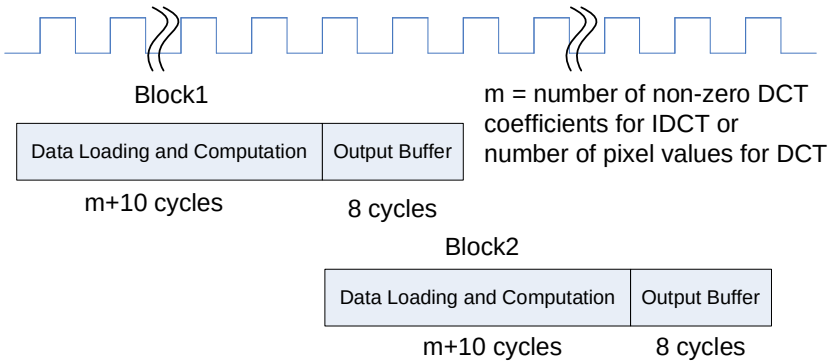



Figure 3.10: Timing diagram of IDCT and DCT

Similar to IDCT computation, ME & MC in the DCT domain can also take advantage of the sparseness property of the DCT coefficient matrix of the surrounding block $f_i$. For example, let $\hat{f}_0$, $\hat{f}_1$, $\hat{f}_2$, and $\hat{f}_3$ have the following nonzero coefficients after quantization:

$$\hat{f}_0 = \begin{bmatrix} \hat{f}_0(0,0) & \hat{f}_0(0,1) & 0 & \cdots & 0 \\ 0 & 0 & 0 & \cdots & \vdots \\ 0 & 0 & 0 & \cdots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & \cdots & \cdots & 0 \end{bmatrix} \tag{3.13}$$

$$\hat{f}_1 = \begin{bmatrix} \hat{f}_1(0,0) & \hat{f}_1(0,1) & 0 & \cdots & 0 \\ 0 & 0 & 0 & \cdots & \vdots \\ 0 & 0 & 0 & \cdots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & \cdots & \cdots & 0 \end{bmatrix} \tag{3.14}$$

$$\hat{f}_2 = \begin{bmatrix} \hat{f}_2(0,0) & \hat{f}_2(0,1) & 0 & \cdots & 0 \\ 0 & 0 & 0 & \cdots & \vdots \\ 0 & 0 & 0 & \cdots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & \cdots & \cdots & 0 \end{bmatrix} \tag{3.15}$$

$$\hat{f}_3 = \begin{bmatrix} \hat{f}_3(0,0) & \hat{f}_3(0,1) & 0 & \cdots & 0 \\ 0 & 0 & 0 & \cdots & \vdots \\ 0 & 0 & 0 & \cdots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & \cdots & \cdots & 0 \end{bmatrix} \tag{3.16}$$

The dataflow of the motion estimation in the transform domain using this example is shown in Fig. 3.11. When compared to spatial domain ME and MC, transform domain approach can reduce the total number of clock cycles required to perform ME and MC by utilizing the sparseness property of 2-D DCT coefficient matrix. During the motion estimation process, SAD computation is performed in a row by row fashion, and its computation

flow is shown in Fig. 3.12. Pipelined operations for SAD and scaled outer product computation enable the proposed unified architecture to perform ME and one of the DCT, IDCT, and MC, concurrently. For example, after the partial SAD results are passed from the first row to the second row of PEs during the ME process, the first row of PEs can start immediately to perform a different algorithm using the scaled outer product computation. The timing diagrams of ME and MC are shown in Fig. 3.13 and Fig. 3.14, respectively. For motion estimation in the transform domain, we first reconstruct the predicted block at the search location using the equation in (3.8). Then, SAD computation using the predicted block and the current block is performed, and the motion vector is generated after searching all the locations. Total processing time including initial delay requires $4 \times m + 18 + 1$ clock cycles for each search location in the transform domain motion estimation.

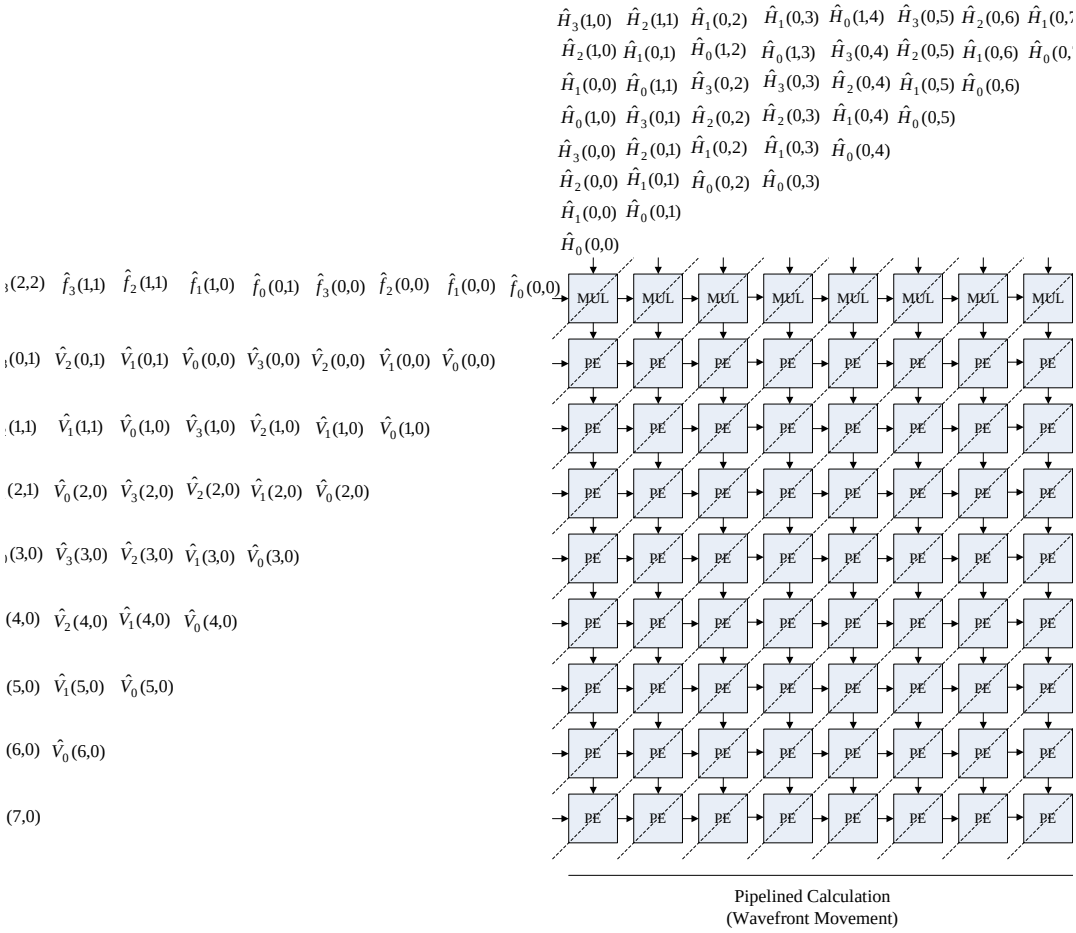



Figure 3.11: Dataflow of motion estimation in the transform domain

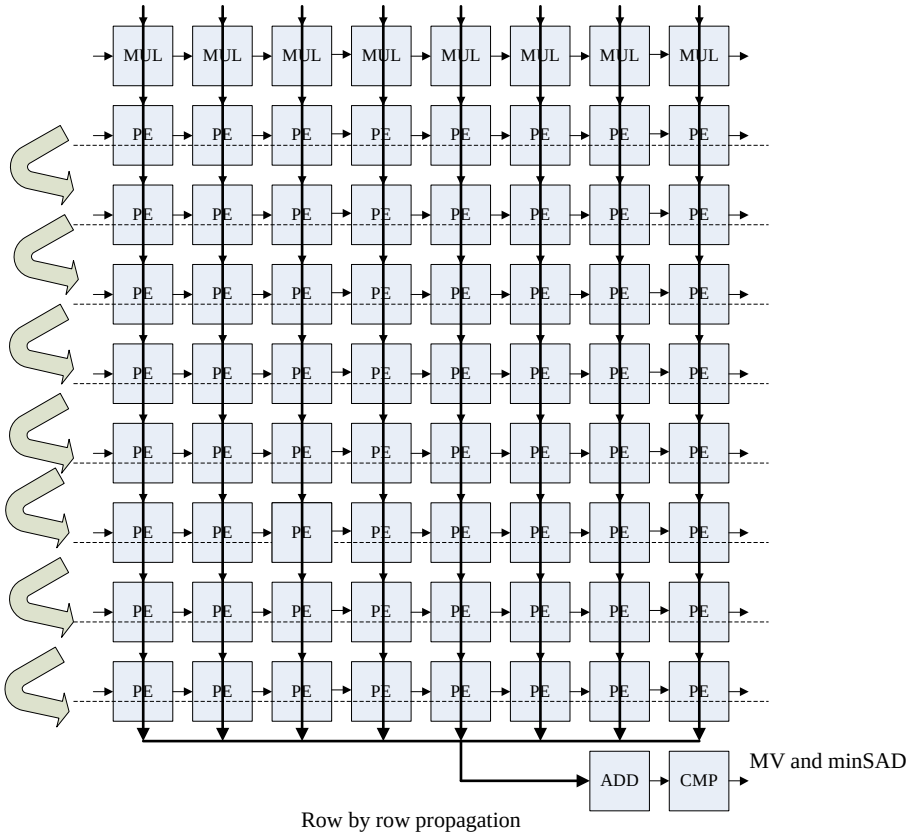



Figure 3.12: Computation of SAD

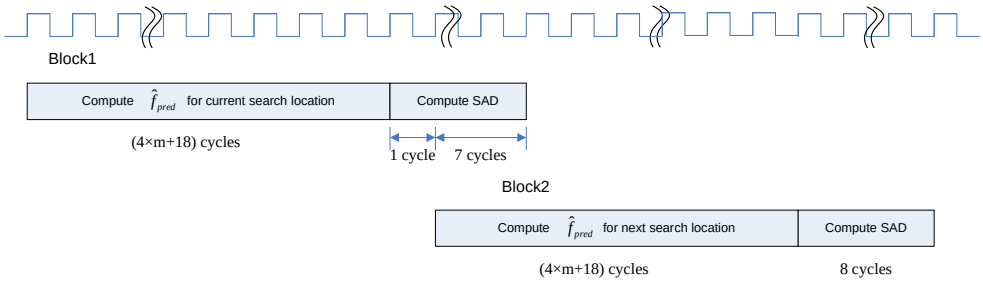



Figure 3.13: Timing diagram of ME in the transform domain

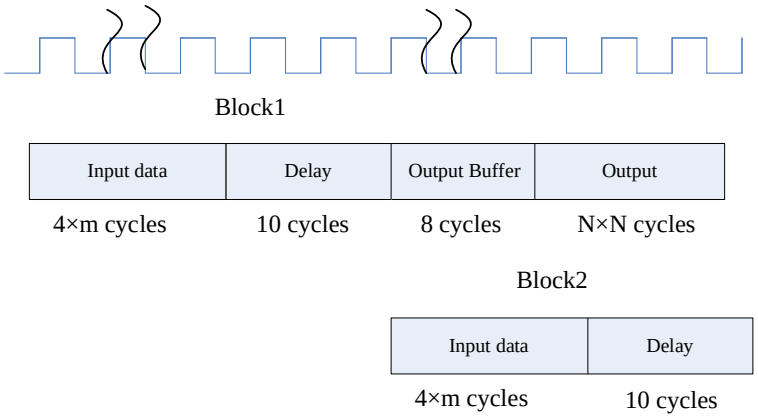



Figure 3.14: Timing diagram of MC in the transform domain

## 3.4 Experimental Results

Our unified architecture consumes 337k gate counts for the hardware implementation. The maximum operating frequency of the system is 66.9 MHz based on the timing analysis.

The most important part for our unified architecture is that it can perform MC and ME in the DCT domain while [DKF05] [GSJ02] [KK99] [GHC04] [HL04] [YH95] can not perform these two functions. These two functions are extremely important for video transcoding in the DCT domain due to its low computational complexity. It was shown that DCT-domain motion compensation and down-conversion with a scheme for coding mode selection and motion vector scaling can provide about 40% reduction of computational complexity when compared to the spatial-domain approach [ZYB98]. Table 2 shows the comparisons of DCT domain transcoder and spatial domain transcoder. CPDT is a cascaded pixel domain transcoder [YSX99]. It shares motion vectors in both decoder and encoder parts to omit motion estimation. SDDT is a simplified DCT domain transcoder [AG05]. It only uses one DCT-MC in the decoder part. Therefore, it can only be used for bit-rate transcoding. CDDT is a cascaded DCT domain transcoder [ZYB98]. It uses two DCT-MC in both decoder and encoder part. It can be used for spatial resolution down-scaling. We can see from Table 3.2 that DCT domain transcoders do not need DCT/IDCT pair, and have lower computational complexity and fast computation time.

In [XLS05], 300-frame CIF "Foreman" sequence was encoded at QP = 7, and then transcoded at QP = 15 on a 1.8 GHz computer system. Since no hardware accelerator was

Table 3.2: Comparisons of DCT domain transcoder and spatial domain transcoder

| Architecture | Spatial domain trans. (CPDT) [YSX99] | Transform domain trans. (SDDT) [AG05] | Transform domain trans. (CDDT) [ZYB98] |
|---|---|---|---|
| Functional Blocks | Need IDCT/DCT/Spatial MC | No IDCT/DCT<br>Need DCT-MC | No DCT/IDCT<br>Need DCT-MC |
| Computation Complexity | High | Low | Medium |
| Computation time | 7 FPS | 23.2 FPS | 14.1 FPS |

used for these transcoders, only 7 23.2 FPS can be transcoded in real time. However, our hardware implementation can finish the DCT-MC for each block in $4m + 10$ clock cycles. Therefore, $1920 \times 1080$ HDTV resolution at 30 FPS can be transcoded in real-time if the proposed architecture is used for SDDT or CDDT with 30 MHz operating clock frequency.

In addition to the performance advantage of DCT-MC and the benefits of using DCT-MC in the DCT domain transcoder, our architecture can provide additional benefits to the transcoding applications by realizing DCT/IDCT/DCT-ME on the same hardware fabric. Using the same architecture can reduce the design time and simplify the design process, and make the transcoding system highly flexible. When combined with an embedded processor, different transcoding applications can be implemented using the same chip.

The comparisons between the proposed unified architecture and some representative hardware architectures are shown in Table 3.3. The area and frequency information of [HL04] and [YH95] are given by Chen et al. [CCH06]. The throughput rates for transform domain ME & MC and IDCT algorithms in our proposed architecture depend on the number of non-zero DCT coefficients. Non-zero information is given by run-length coding (RLC) or run-length decoding (RLD) modules in the video encoding or decoding system. No extra

hardware module for detecting and skipping zero values is necessary. Therefore, our proposed unified architecture can be more attractive for low bit rate image/video coding applications.

Comparing with the multi-core architecture which can perform spatial ME/MC, DCT, IDCT [DKF05], our proposed architecture saves the area by 67.9% including the DCT domain ME/MC. From Fig. 3.8, we can see that the average number of cycles for $8 \times 8$ IDCT is below 15. Our architecture can perform IDCT operation 5.8 times faster than that in [DKF05]. Comparing with the FPGA implementation of video coding system [GSJ02], our architecture is also smaller in area and has higher operating clock frequency. Although the area in [GSJ02] includes quantization and inverse quantization, it does not provide functional blocks for DCT domain ME/MC.

Comparing with the DCT/IDCT implementations in [KK99], [GHC04], our architecture can perform more functions, i.e., MC in the DCT domain and ME in the DCT domain. If the video transcoding system uses the architectures in [KK99], [GHC04] for the DCT/IDCT implementations, it will need additional modules for ME/MC implementations. The throughput of DCT/IDCT using our unified architecture is also outperforming those in [KK99], [GHC04]. For example, it takes 96 clocks for $8 \times 8$ DCT/IDCT in [KK99]. It only takes 74 clocks for $8 \times 8$ DCT and 15 clocks for $8 \times 8$ IDCT using our unified architecture (m=5). Comparing to the architecture for spatial ME in [YH95], our unified architecture saves the hardware resource, and it can perform more functions.

Table 3.3: Comparison of different hardware architectures

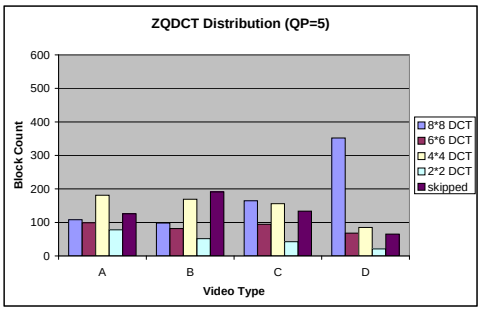
| Architecture | [DKF05] | [GSJ02] | [KK99] | [GHC04] | [HL04] | [YH95] | Proposed |
|---|---|---|---|---|---|---|---|
| No of PEs | – | – | – | – | $N^2$ | $4p^2$ | $N^2$ |
| Transpose Memory | – | yes | yes | no | – | – | no |
| Operating Cycles (Cycles/$8 \times 8$ Block) | 138 For DCT<br>190 For IDCT<br>99 For ME<br>110 For MC | – | 96 | $N^2$ | $(2p + N - 1)^2$ | $N^2$ | $N^2 + 10$ for DCT<br>$M + 10$ for IDCT<br>$4 \times M + 19$ for DCT-ME<br>$4 \times M + 10$ for DCT-MC |
| No. of DATA ACCESS | – | – | $N^2$ | $N^2$ | $N^2 + (2P + N - 1)^2$ | $3N^2$ | $N^2$ for DCT<br>$M$ for IDCT<br>$4 \times M$ for ME/MC |
| Area | 1050 KGates | 400 KGates | 25.3 KGates | 52 KGates | 231 KGates | 2907 KGates | 337 Kgates |
| Frequency (MHz) | 145 | 12 | 80 | 125 | 152.5 | 6.9 | 66.9 |
| Functions | Spatial ME/MC DCT/IDCT | DCT/IDCT/Q/IQ /ME/MC | DCT/IDCT | DCT/IDCT | Spatial ME | Spatial ME | DCT/IDCT/DCT-MC/DCT-ME |

# CHAPTER 4
# RECONFIGURABLE ARCHITECTURE FOR ZQDCT USING COMPUTATIONAL COMPLEXITY PREDICTION AND BITSTREAM RELOCATION

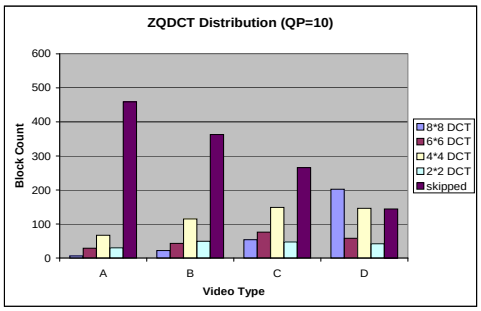
Due to the high computational complexity of Discrete Cosine Transform (DCT) computations, prediction of zero quantized DCT (ZQDCT) coefficients has been extensively studied to reduce the computational complexity of DCT computations. In this dissertation, we propose a reconfigurable architecture to support ZQDCT computations. 12 different modes of DCT computations including zonal coding, multi-block processing, and parallel sequential stage mode can be performed using proposed architecture. We develop a hybrid model-based quality priority algorithm to reduce power consumptions, required hardware resources, and computation time with a small quality degradation.

## 4.1 Quality Priority Prediction

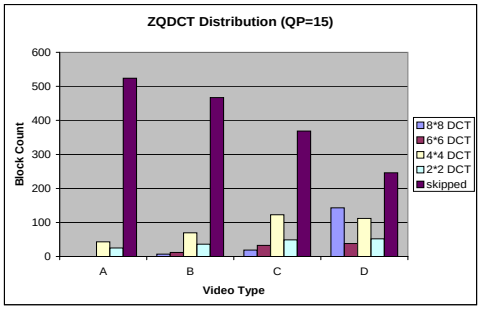
We use Peak Signal to Noise Ratio (PSNR) for the quality measurement. We use SAD and QP to predict the DCT modes to achieve low quality degradation while reducing the computational complexity. Let $f(x, y), 0 \leq x, y \leq 7$ be the prediction error for DCT computations of inter-frame. $SAD = \sum_{x=0}^{7} \sum_{y=0}^{7} |f(x, y)|$ is the sum of absolute differences



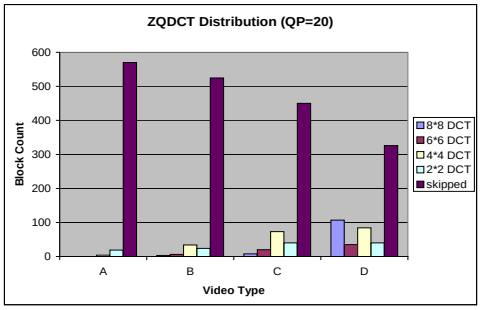

(a) QP=5



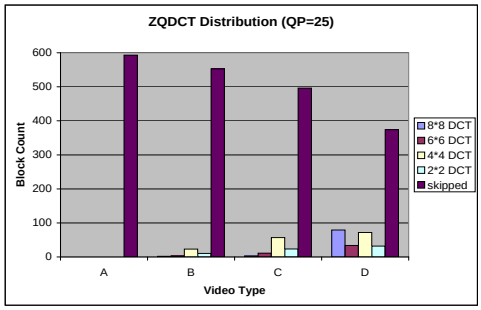

(b) QP=10




(c) QP=15




(d) QP=20




(e) QP=25

Figure 4.1: ZQDCT Distribution

of the prediction error. It can be easily obtained from the motion estimation module of video encoder. The DCT coefficients of $f(x,y)$ after $8 \times 8$ DCT computations are defined as $F(u,v), 0 \leq u,v \leq 7$. Both $f(x,y)$ and $F(u,v)$ can be modeled using Gaussian distribution [WKK07]. Suppose that the probability density function (pdf) of $f(x,y)$ is defined with zero mean and variance $\sigma^2$ as $p(x) = \frac{1}{\sqrt{2\Pi}\sigma} e^{-\frac{x^2}{2\sigma^2}}$. Expected value of $|x|$ will be $E[|x|] = \sqrt{\frac{2}{\Pi}}\sigma \approx \frac{SAD}{N}$. Therefore,

$$\sigma \approx \sqrt{\frac{\Pi}{2}} \frac{SAD}{N} \tag{4.1}$$

The variance of $F(u,v)$ can be written as

$$\sigma_F^2(u,v) = \sigma^2 [ARA^T]_{u,u} [ARA^T]_{v,v} \tag{4.2}$$

where

$$R = \begin{pmatrix} 1 & \rho & \rho^2 & \cdots & \rho^7 \\ \rho & 1 & \rho & \cdots & \rho^6 \\ \rho^2 & \rho & 1 & \cdots & \rho^5 \\ & \vdots & & \ddots & \\ \rho^7 & \cdots & \cdots & \cdots & 1 \end{pmatrix} \tag{4.3}$$

Here, the correlation coefficient $\rho = 0.6$ is used. The $u$th row of $A$ is the basis vector $\frac{1}{2} C(u) cos(\frac{(2x+1)u\pi}{16})$. The probability of $F(u,v)$ being quantized to zero is controlled by (4.4).

$$\gamma \sigma_F < \alpha Q_P \tag{4.4}$$

The DCT coefficient will be quantized to zero with 99.73% probability when $\gamma = 3$. Here, $\alpha = 2.5$ is used for H.263 inter-frame video coding.

From (4.1), (4.2), and (4.4), we can get the criterion for estimating zero quantized DCT coefficients, as shown in (4.5).

$$SAD < \beta_G(u,v) \times \alpha Q_P \tag{4.5}$$

The threshold matrix $\beta_G$ is defined in (4.6).

$$\beta_G(u,v) = \frac{\sqrt{2}N}{\gamma\sqrt{\pi[ARA^T]_{u,u}[ARA^T]_{v,v}}} \tag{4.6}$$

Here, $N = 64$. Based on the Gaussian distribution model in (4.5) and the comprehensive analysis of the dynamic range of DCT coefficients, we first analyze the distributions of Zero Quantized DCT (ZQDCT) coefficients for different video sequences using Algorithm 1 [WKK07].

**Algorithm 1** Counting blocks for ZQDCT

Count the number of blocks for one frame using the following thresholds.
**if** $SAD < 8\alpha Q_P$ **then**
  increase a counter by one for skipped mode.
**else if** $8\alpha Q_P \leq SAD < 9.26\alpha Q_P$ **then**
  increase a counter by one for $2 \times 2$ DCT mode.
**else if** $9.26\alpha Q_P \leq SAD < 14.34\alpha Q_P$ **then**
  increase a counter by one for $4 \times 4$ DCT mode.
**else if** $14.34\alpha Q_P \leq SAD < 18.07\alpha Q_P$ **then**
  increase a counter by one for $6 \times 6$ DCT mode.
**else**
  increase a counter by one for $8 \times 8$ DCT mode.
**end if**

We categorize the video sequences into four different types based on the motion activities, .i.e., from Type A to Type D. Type A includes Bridge-far. Type B includes Claire, Highway, Hall, Grandma. Type C includes Container, News, Carphone, Salesman, Silent, Foreman.

Type D includes Football. The average number of block counts for different video sequences and different QPs are shown in Figure 4.1. 100 frames are used for analysis for each video sequence and they are in Quarter Common Intermediate Format (QCIF).

Then, we analyze the quality degradations for each video sequence type, as shown in Table 4.1. We perform zonal codings for each sequence. We use the results from Table 4.1 for choosing certain DCT mode for an arbitrary video sequence. Assuming acceptable quality degradation is less than 1, the DCT mode prediction is shown in Table 4.2. Given an arbitrary video sequence, the algorithm for quality priority prediction is shown in Algorithm 2.

Table 4.1: PSNR degradation for typical video sequences

| | QP | Type A | Type B | Type C | Type D |
|---|---|---|---|---|---|
| $6 \times 6$ | 5 | 0.922 | 1.486 | 1.579 | 1.590 |
| | 10 | 0.0055 | 0.3361 | 0.273 | 0.2904 |
| | 15 | 0 | 0.1523 | 0.10325 | 0.1013 |
| | 20 | 0 | 0.078 | 0.041 | 0.036 |
| | 25 | 0 | 0.0023 | 0.024 | 0.0285 |
| $4 \times 4$ | 5 | 1.702 | 3.565 | 4.29 | 5.682 |
| | 10 | 0.026 | 1.047 | 1.367 | 2.171 |
| | 15 | 0 | 0.577 | 0.678 | 1.126 |
| | 20 | 0 | 0.301 | 0.331 | 0.552 |
| | 25 | 0 | 0.157 | 0.184 | 0.364 |
| $2 \times 2$ | 5 | 2.393 | 6.086 | 7.121 | 9.996 |
| | 10 | 0.061 | 2.604 | 3.313 | 5.52 |
| | 15 | 0.0004 | 1.683 | 2.089 | 3.703 |
| | 20 | 0 | 1.146 | 1.354 | 2.551 |
| | 25 | 0 | 0.829 | 0.941 | 1.845 |
| skipped | 5 | 2.785 | 8.105 | 9.541 | 13.459 |
| | 10 | 0.08 | 4.348 | 5.538 | 8.739 |
| | 15 | 0.0004 | 3.232 | 4.136 | 6.742 |
| | 20 | 0 | 2.519 | 3.176 | 5.471 |
| | 25 | 0 | 2.026 | 2.561 | 4.41 |

**Algorithm 2** Quality Priority Algorithm

1. Count number of blocks based on Algorithm 1 for each frame.
2. Calculate the SAD values among the arbitrary video sequence and the typical video sequences in Figure 4.1.
3. Map the video sequence type by selecting the smallest SAD.
4. Predict the DCT mode based on Table 4.2.

Table 4.2: DCT mode prediction

| | QP=5 | QP=10 | QP=15 | QP=20 | QP=25 |
|---|---|---|---|---|---|
| Type A | $6 \times 6$ | skipped | skipped | skipped | skipped |
| Type B | $8 \times 8$ | $6 \times 6$ | $4 \times 4$ | $4 \times 4$ | $2 \times 2$ |
| Type C | $8 \times 8$ | $6 \times 6$ | $4 \times 4$ | $4 \times 4$ | $2 \times 2$ |
| Type D | $8 \times 8$ | $6 \times 6$ | $6 \times 6$ | $4 \times 4$ | $4 \times 4$ |

Table 4.3 shows PSNR and bit rate using quality priority prediction algorithm in H.263 when QP=15 and frames per second (fps) = 30 for 100 frames. By using calculated percentage of each DCT mode selected for each frame by the quality priority prediction algorithm, overall computational complexity is reduced greatly with a small quality degradation, as shown in Table 4.3. The actual frequency of reconfiguration depends on the input video sequence characteristics and Quantization Parameter to maximize the benefits by run-time reconfiguration in the given condition.

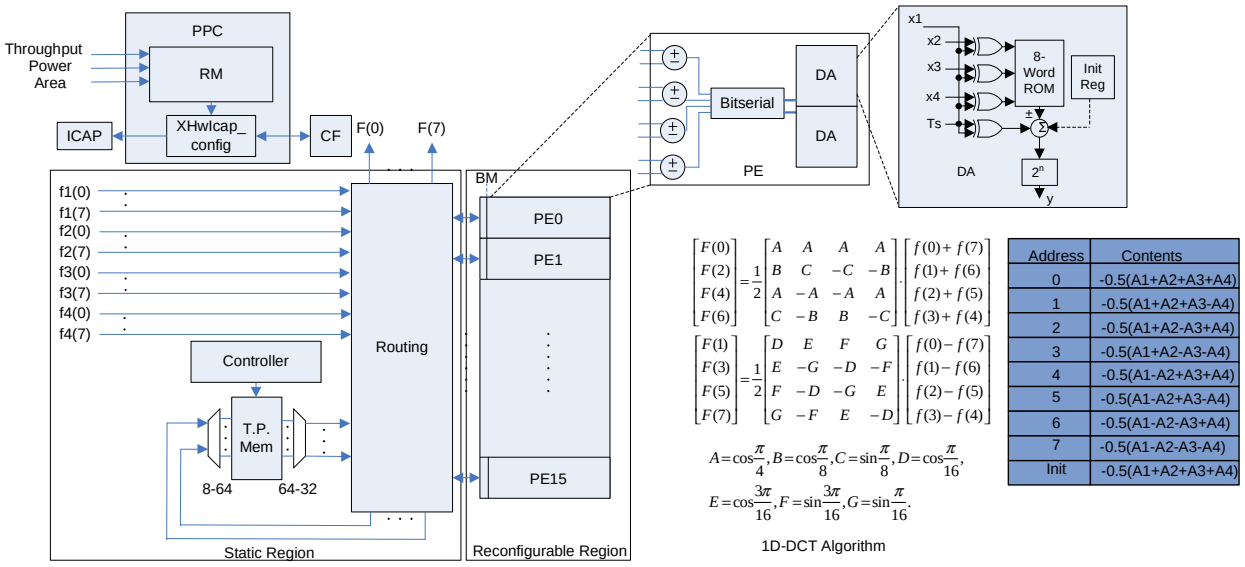
Table 4.3: PSNR and BPS comparisons for QP=15

| | Bridge-far | Foreman | Football | Mother |
|---|---|---|---|---|
| PSNR Original(dB) | 35.482 | 30.176 | 29.417 | 31.257 |
| PSNR Predicted (dB) | 35.4817 | 29.497 | 29.096 | 30.931 |
| BPS Original (Kbits/sec) | 4.8105 | 64.044 | 312.146 | 28.952 |
| BPS Predicted (Kbits/sec) | 4.81 | 53.369 | 304.62 | 26.381 |
| Overall Computation Reduction | 99% | 74.3% | 51.9% | 74.3% |

## 4.2 Reconfigurable Architecture

2D-DCT is computed using two 1D-DCT stages with a transpose memory. Chen's algorithm is used for 1D-DCT computation [CSF77], as shown in Fig. 4.2. The key computations

of 1D-DCT are inner products. Distributed Arithmetic (DA) is used to perform the inner product computations.




| Address | Contents |
|---|---|
| 0 | -0.5(A1+A2+A3+A4) |
| 1 | -0.5(A1+A2+A3-A4) |
| 2 | -0.5(A1+A2-A3+A4) |
| 3 | -0.5(A1+A2-A3-A4) |
| 4 | -0.5(A1-A2+A3+A4) |
| 5 | -0.5(A1-A2+A3-A4) |
| 6 | -0.5(A1-A2-A3+A4) |
| 7 | -0.5(A1-A2-A3-A4) |
| Init | -0.5(A1+A2+A3+A4) |

Figure 4.2: Hardware platform for reconfigurable DCT architecture

The top level architecture of reconfigurable DCT for scalable computation is shown in Fig. 4.2. For the advantage of dynamic partial reconfiguration, the design is divided into one static region and multiple partial reconfigurable regions (PRRs). Static region remains unchanged all the time. PRRs can be changed during run-time. The static region in our design includes Controller, Transpose Memory (T.P. Mem), and Interconnections. We use embedded processor, PPC to implement Reconfigurable Manager (RM) and XHwIcap_config module. There are 32 input ports in our design for multi-block processing. If $8 \times 8$ DCT is performed, only input ports from f1(0) to f1(7) are used. The other input ports are used for increasing the input bandwidth. For example, in $2 \times 2$ DCT mode, four blocks can be processed at the same time by using 32 input ports. T.P. Mem is used to transpose the

intermediate results from the first stage 1D-DCT. Controller is used to generate the read address, write address, and control signals for T.P. Mem.

ICAP is the Internal Configuration Access Port for FPGA reconfiguration. Partial bitstream is stored in the CF card. XHwIcap_config module is implemented using PPC for performing dynamic partial reconfiguration. It fetches the partial bitstream from the CF card and sends it to the ICAP interface. It also performs bitstream relocation. RM module is used for priority prediction and determines which mode of DCT computation should be reconfigured. It takes quality, computation time, power, and area parameters for different priority prediction. The routing module includes the interconnections to input and output data from PEs.

The reconfigurable region in our design includes 16 PRRs which require bus marco (BM) between PRRs and Static Region. Different numbers and types of PEs can be reconfigured for different DCT computation modes to meet different computation time, PSNR, power, and area goals. In $4 \times 4$ and $2 \times 2$ zonal coding modes, we can take advantage of dynamic partial reconfiguration to configure the unused PEs to perform DCT computations for the extra pixel blocks to realize multi-block processing. In addition to different mode for DCT zonal coding, different stage modes can also be achieved, i.e., two stage parallel mode or one stage sequential mode. Two stage parallel mode means that 8 PEs are used for the first stage of 2-D DCT computation and the other 8 PEs are used for the second stage of 2-D DCT computation. Therefore, 2 1-D DCT operations are pipelined to increase throughput rate. One stage sequential mode means that 8 PEs are used for both the first stage and the second

stage of 2-D DCT computation. The other 8 PEs can be configured with blank configurations to save power consumption or available for the implementation of other functions. PE is the basic computation unit which performs inner products using distributed arithmetic.

## 4.3 Bitstream Relocation

PR design is partitioned into static region and partial reconfigurable regions (PRRs). Multiple partial reconfigurable modules (PRM) can be mapped into each PRR. Each PRM can perform one function/task for the PRR. Each PRM will generate one partial bitstream. For example, if we have 2 PRRs and 3 PRMs for each PRR, it requires 6 partial bitstreams and 2 blank bitstreams for the PRRs. Blank bitstream is used for no switching activities for the PRR. By using bitstream relocation, partial bitstream with identical PRM can be relocated to different PRRs. We change the location of configuration data of the identical PRM. Using bitstream relocation can reduce both the storage size of the partial bitstreams and reconfiguration time due to external memory accesses.

In [FGG09], the authors proposed a bitstream relocation scheme for local clock domains. PRRs can drive local clock domains internally to change the clock frequency. They used Virtex-4 FPGA for PR implementations. Becker et al. introduced a method that enhances the relocatability of partial bitstreams for FPGA run-time reconfiguration [WC07]. Their approach circumvents the problem of having to find fully identical regions based on compatible subsets of resources. This enables flexible placement of relocatable modules. They used a

software defined radio for prototyping using Virtex-4 FPGA. The experimental results show that the number of partial bitstreams is reduced by 50% and the compile time is shortened by 43%. In [KP06], the authors developed the REPLICA2Pro (Relocation per online Configuration Alteration in Virtex-2/-Pro) filter to perform task relocation by manipulating the task's bitstream during regular allocation process. The task relocation is performed by manipulating the frame addresses (FAR) in the corresponding partial bitstream. Virtex-2/-Pro has different FAR format with Virtex-4 device. In Virtex-2/-Pro, Major Address (MJA) is used to determine the configuration column position. The filter architecture and the design flow are presented in the dissertation.

Virtex-4 configuration memory is arranged in frames [CT08]. Each frame has fixed length, i.e., 41 words. To perform configuration, 32-bit data packets are used for different configuration registers. The configuration data packets consist of 32-bit type 1 packet and followed by either 32-bit type 2 packet or 32-bit data body. The type 1 packet is used for register reads and writes, and type 2 packet is used to write long blocks. There are many different kinds of configuration registers. For bitstream relocation, Frame Address Register (FAR) and CRC registers are needed. Bitstream relocation is performed by manipulating FAR packet. FAR packet is composed of minor address, column address, row address, block type, and Top_B Bit. When performing bitstream relocation, we first search for the write FAR command, i.e., 0x30002001. Then, we change the FAR packet, i.e., adding distance for row address, column address or Top_B Bit depending on the locations. After changing the FAR packet, the CRC

register needs to be recalculated. In our experiment, we disable the CRC check since the bitstream corruption is not a main concern here.

## 4.4 Experimental Results

Xilinx ISE is used for synthesis in our design. Xilinx XPower is used for power estimation. Xilinx PlanAhead is used for floorplanning, and implementing the design. For bitstream relocation, some small modifications of the PlanAhead design flow need to be performed. First, "-g CRC: disable" needs to be added in Assembly Option to disable CRC check. Second, "Area_GROUP PRR_X Routing = CLOSED" needs to be added in the constraints file. This constraint is used not to allow resources from the static region to occupy unused resources in PRRs.

We used Xilinx ML410 Development Platform with Virtex-4 FX60 FPGA. In our design, we have 16 PRRs. The floorplan of our proposed reconfigurable architecture is shown in Fig. 4.3. Table 4.4 shows a part of the configurations of our reconfigurable architecture. Each PRR can be configured with certain type of PE configuration or blank configuration. Using blank configuration can reduce the power consumption since no switching activities are present. The DCT modes can be $8 \times 8$, $6 \times 6$, $4 \times 4$, $2 \times 2$ with parallel or sequential mode. In addition, $4 \times 4$ mode can have double throughput mode ($4 \times 4 \times 2$), and $2 \times 2$ mode can have quadruple throughput mode ($2 \times 2 \times 4$). Double throughput mode means that 2 blocks are processed concurrently. Quadruple throughput mode means that 4 blocks

are processed concurrently. The double or quadruple throughput modes can be changed to single throughput mode by replacing certain PE bitstreams with blank bitstreams. 12 different modes can be achieved by different combinations of PEs.

The bitstreams are generated using PR Assemble process of PlanAhead [Jac08]. 48 partial bitstreams are generated for 16 PRRs including 16 blank bitstreams. However, since only 8 different PRMs exist in our design, only 9 partial bitstreams including 1 blank bitstream become necessary if bitstream relocation scheme is applied. The partial bitstream size and blank bitstream size for each PRM are 25 kb and 15.5 kb, respectively. Therefore, the partial bitstream sizes can be saved by 79.4% in total. Partial reconfigurable regions should be large enough to fit the biggest PE. Based on 400 MB/s reconfiguration rate, the reconfiguration time for each partial bitstream and blank bitstream are 64 us and 40 us, respectively. A full bitstream with fixed size (2.5 MB) is generated to initialize the FPGA when power is on. If static design without partial reconfigurable capability is used, 30 MB of bitstreams' size will be needed for all the 12 DCT modes. 6.6 ms is needed to change the configuration from one mode to another. By using dynamic partial reconfiguration, only 80 us is needed for the best case which reconfigures two blank bitstreams (e.g., from $8 \times 8 \times 1$ to $6 \times 6 \times 1$ in sequential mode) and 768 us is needed for the worst case which reconfigures six partial bitstreams (e.g., from $2 \times 2 \times 1$ to $8 \times 8 \times 1$ in sequential mode).

Comparison results of computation time, hardware resources, and power consumption for different DCT modes are also shown in Table 4.4. The comparisons of our proposed work

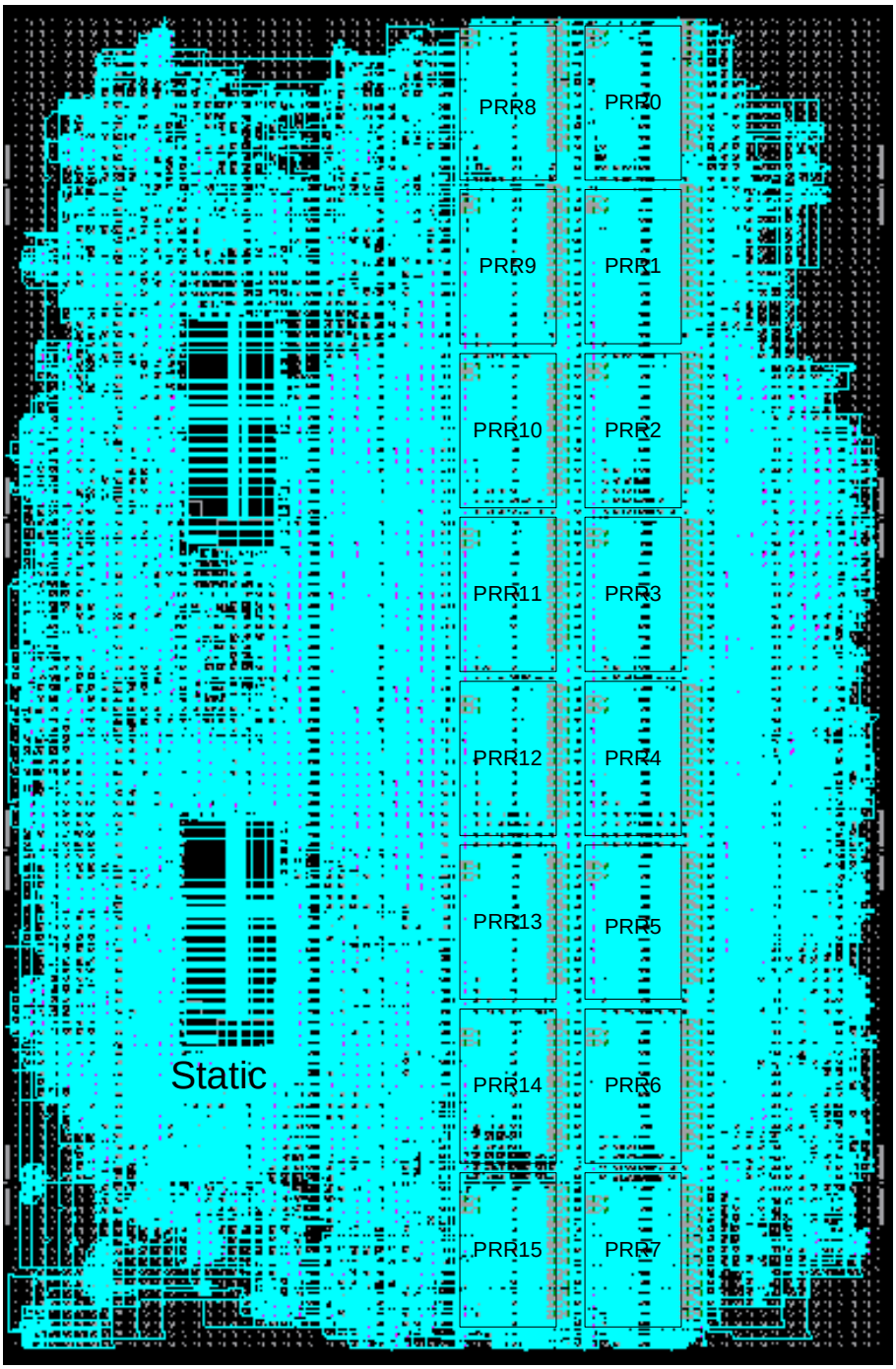



Figure 4.3: Floorplan of proposed reconfigurable architecture

with previously implemented works are shown in Table 4.5. Compared to previous works, our proposed work is highly flexible in terms of the power and throughput.

Table 4.4: PRR configurations

| | $8 \times 8 \times 1$ | | $6 \times 6 \times 1$ | | $4 \times 4 \times 1$ | | $4 \times 4 \times 2$ | | $2 \times 2 \times 1$ | | $2 \times 2 \times 4$ | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Parallel | Sequen. | Parallel | Sequen. | Parallel | Sequen. | Parallel | Sequen. | Parallel | Sequen. | Parallel | Sequen. |
| PRR0 | PE0 | PE0 | PE0 | PE0 | PE0 | PE0 | PE0 | PE0 | PE0 | PE0 | PE0 | PE0 |
| PRR1 | PE1 | PE1 | PE1 | PE1 | PE1 | PE1 | PE1 | PE1 | PE1 | PE1 | PE1 | PE1 |
| PRR2 | PE2 | PE2 | PE2 | PE2 | PE2 | PE2 | PE2 | PE2 | blank | blank | PE0 | PE0 |
| PRR3 | PE3 | PE3 | PE3 | PE3 | PE3 | PE3 | PE3 | PE3 | blank | blank | PE1 | PE1 |
| PRR4 | PE4 | PE4 | PE4 | PE4 | blank | blank | PE0 | PE0 | blank | blank | PE0 | PE0 |
| PRR5 | PE5 | PE5 | PE5 | PE5 | blank | blank | PE1 | PE1 | blank | blank | PE1 | PE1 |
| PRR6 | PE6 | PE6 | blank | blank | blank | blank | PE2 | PE2 | blank | blank | PE0 | PE0 |
| PRR7 | PE7 | PE7 | blank | blank | blank | blank | PE3 | PE3 | blank | blank | PE1 | PE1 |
| PRR8 | PE0 | blank | PE0 | blank | PE0 | blank | PE0 | blank | PE0 | blank | PE0 | blank |
| PRR9 | PE1 | blank | PE1 | blank | PE1 | blank | PE1 | blank | PE1 | blank | PE1 | blank |
| PRR10 | PE2 | blank | PE2 | blank | PE2 | blank | PE2 | blank | blank | blank | PE0 | blank |
| PRR11 | PE3 | blank | PE3 | blank | PE3 | blank | PE3 | blank | blank | blank | PE1 | blank |
| PRR12 | PE4 | blank | PE4 | blank | blank | blank | PE0 | blank | blank | blank | PE0 | blank |
| PRR13 | PE5 | blank | PE5 | blank | blank | blank | PE1 | blank | blank | blank | PE1 | blank |
| PRR14 | PE6 | blank | blank | blank | blank | blank | PE2 | blank | blank | blank | PE0 | blank |
| PRR15 | PE7 | blank | blank | blank | blank | blank | PE3 | blank | blank | blank | PE1 | blank |
| Comp. Time (Cycles/Block) | 64 | 128 | 64 | 128 | 64 | 96 | 32 | 48 | 64 | 80 | 16 | 20 |
| Hard. Resources (Gate Count) | 88463 | 57270 | 64972 | 41854 | 46710 | 30424 | 89873 | 58005 | 20765 | 12539 | 92761 | 59395 |
| Dynamic Power (mW) | 108 | 80 | 90 | 68 | 81 | 61 | 129 | 104 | 52 | 40 | 163 | 136 |
| Each PRR (768 LUTs, 768 FFs, 384 Slices) | Utilization: LUT 51.43% $\sim$ 72.14%, FF 8.85% $\sim$ 8.98%, Slice 63.02% $\sim$ 88.02% | | | | | | | | | | | |

Table 4.5: Comparisons with other DCT implementations

| | [KK99] | [DB03] | [GHC04] | [GVB05] | [GK07] | Proposed |
|---|---|---|---|---|---|---|
| Hardware Resources (Gate Counts) | 23376 | 11550 | 52143 | 18590 | 69468 | 12539-88463 |
| Operating cycles (Cycles/$8 \times 8$ Block) | 96 | N/A | 64 | N/A | 526 | 16-128 |
| Transpose Memory | yes | yes | no | yes | no | yes |
| Technology | 0.5 $\mu m$ | 0.35 $\mu m$ | 0.25 $\mu m$ | FPGA (90 nm) | FPGA (90 nm) | FPGA (90 nm) |
| Functions | $8 \times 8$ DCT | $5 \times 5$ DCT | $8 \times 8$ DCT | $8 \times 8$ DCT | $8 \times 8$ DCT | 12 DCT Modes |
| Power (mW) | N/A | 129.2 | N/A | 281 | N/A | 40-163 |

We use H.263 reference software to simulate our mode decision scheme. Some representative video sequences (Bridge-far, Grandma, Foreman, Football) are used to test our quality priority prediction algorithm. In addition, we use some video sequences (Bridge-close, Mother and Daughter) which are not used for our ZQDCT distribution analysis and PSNR analysis to verify the quality priority prediction algorithm. We set frame rate to 30 frames/second. At the beginning, $8 \times 8$ DCT is computed as default mode. Then, we apply

Algorithm 2 to the incoming video sequences. The PSNR and bits per second (BPS) comparisons using quality priority prediction algorithm are shown in Table 4.6. With a small quality degradation, we can reduce the bitrate, save the power/area/computation time. We show the detailed PSNR and Power/Area/Throughput Comparisons for Football sequence in Figure 4.4 and 4.5.

Table 4.6: PSNR and BPS comparisons

| | | Bridge-far | Grandma | Foreman | Football | Bridge-close | Mother daughter |
|---|---|---|---|---|---|---|---|
| QP=5 | PSNR degradation (dB) | 0.2456 | 0 | 0 | 0.0031 | 0 | 0 |
| | BPS reduction (Kbits/sec) | 96.7988 | 0 | 0 | 0.0625 | 0 | 0 |
| QP=10 | PSNR degradation (dB) | 0.0801 | 0.0787 | 0.1645 | 0.345 | 0.6234 | 0.1611 |
| | BPS reduction (Kbits/sec) | 1.038 | 0.6297 | 5.6535 | 13.8574 | 12.934 | 3.318 |
| QP=15 | PSNR degradation (dB) | 0.0004 | 0.1331 | 0.6785 | 0.3214 | 0.5957 | 0.3265 |
| | BPS reduction (Kbits/sec) | 0.0005 | 0.5621 | 9.6748 | 7.5254 | 7.7863 | 2.571 |
| QP=20 | PSNR degradation (dB) | 0 | 0.057 | 0.311 | 0.6041 | 0.3319 | 0.151 |
| | BPS reduction (Kbits/sec) | 0 | 0.0983 | 3.2932 | 20.1914 | 2.6305 | 0.7436 |
| QP=25 | PSNR degradation (dB) | 0 | 0.1543 | 0.9618 | 0.6042 | 0.3457 | 0.6337 |
| | BPS reduction (Kbits/sec) | 0 | 0.2129 | 5.4056 | 10.9775 | 1.4754 | 1.489 |

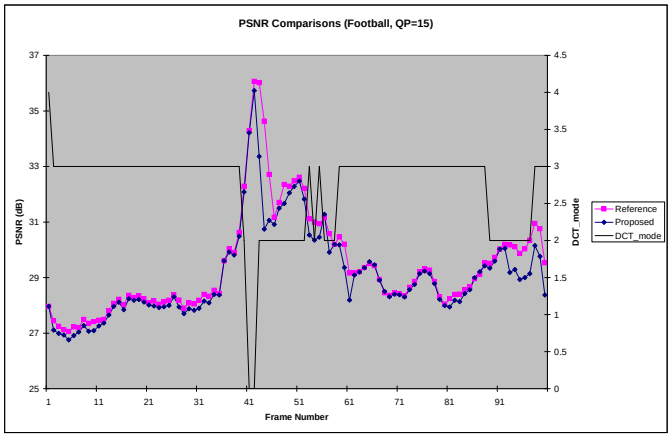



Figure 4.4: PSNR comparisons of Football sequence

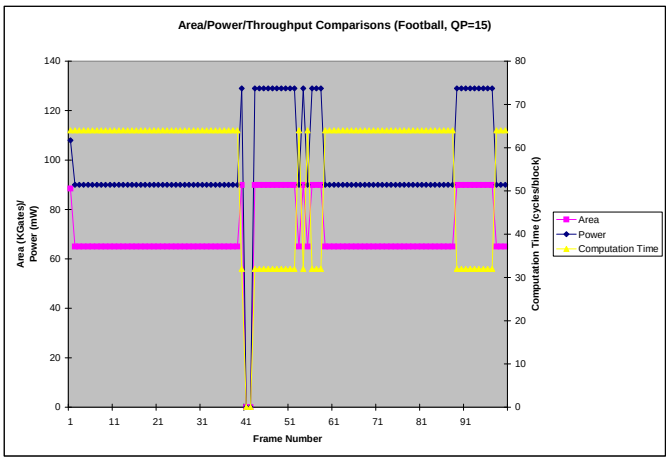



Figure 4.5: Power/Area/Throughput comparisons of Football sequence

# CHAPTER 5
# CONCLUSION

We propose a novel unified architecture that can perform DCT, IDCT, motion estimation & compensation in the DCT domain for video transcoding applications. To our best understanding, this is the first hardware architecture that can perform all these functions on the same hardware fabric. The architecture is based on a Wavefront Array Processor. DCT-MC is extremely important for DCT domain transcoder. Our implementation can reduce the computation complexity of DCT-MC by taking advantage of the sparseness property of DCT coefficients. DCT/IDCT/DCT-ME can be used for different types of video transcoding systems, such as displaying, logo insertion, and temporal transcoding.

Traditional IP-based implementations on FPGA device are often used to simplify the overall design process of complex digital systems. However, these pre-constructed circuits with details of required hardware resources, power consumption, and throughput provided by the vendors, have inherent limitation of achieving highly adaptive computing capabilities. While it is a challenging and complex task to explore both algorithmic and architectural design issues, adaptive hardware resource sharing and algorithmic mapping through spatial and time-multiplexing onto reconfigurable hardware show new possibility to provide significant benefits when compared to traditional system design methodologies.

We present an FPGA design for scalable DCT computations using dynamic partial reconfiguration in this work. Compressed bitstreams are used to reduce external memory accesses and storage sizes. BlockRAM is used as a cache to reduce the reconfiguration overhead. The whole design is implemented in Virtex-4 ML410 evaluation board. The PowerPC is included to control the reconfiguration of scalable DCT architecture to exploit trade-offs among different requirements set by the system. It also controls the prefetching and decompression of the partial bitstreams. The proposed work can be used effectively for video surveillance system requiring the storage of compressed bitstreams. Due to the different amounts of motion activities during the day and night time, which results in more or less number of DCT coefficients to be encoded after motion estimation process, the proposed self-reconfiguration scheme can maximize the benefits through its high adaptability. Experimental results show that our approach can reduce the external memory accesses by 69%, and achieve 400 MBytes/s reconfiguration rate.

Our exploration of scalable architecture of DCT and ME computations using FPGA dynamic partial reconfiguration. We used distributed arithmetic based architecture for DCT computations. Using dynamic partial reconfiguration, the processing elements of the DCT architecture can be changed on the fly including the number of the PEs and the internal logic of the PEs. Zonal coding can be achieved by changing the number of the PEs, and full or reduced precision DCT computations can be achieved by changing the internal logics of the PEs. Some of the bits from LSB of the DCT coefficients are often truncated to be zeros due to the following quantization process, especially for high compression ratio video encoding.

Therefore, low precision implementation with reduced ROM size can be beneficial in terms of hardware resources and power consumption since there will not be much differences in quantized DCT coefficients. The FPGA does not need to be stopped while changing the configuration, which is important for many image/video applications. We provided detailed implementation results and comparisons for different configurations of PEs using both partial reconfiguration process and non-partial reconfiguration process. The unused PEs can be used for motion estimation.

We combine a quality priority prediction algorithm to predict zero quantized DCT coefficients based on hybrid model and show highly adaptive architecture which can be continuously reconfigured to support different DCT computation modes during run-time. 12 different modes can be performed by the proposed reconfigurable architecture to achieve different goals of computation time, power, and hardware resources. Using the bitstream relocation can reduce the overall bitstream storage size by 79.4% in our design.

# LIST OF REFERENCES


[AG05] P. A. A. Assuncao and M. Ghanbar. "A frequency-domain video transcoder for dynamic bitrate reduction of MPEG-2 bit streams." *IEEE Transactions on Circuits and Systems for Video Technology*, **8**(8), December 2005.

[AKG07] A. B. Atitallah, P. Kadionik, F. Ghozzi, and P. Nouel. "An FPGA implementation of HW/SW codesign architecture for H.263 video coding." *International Journal of Electronics and Communications*, **61**, 2007.

[AWS05] I. Ahmad, X. Wei, Y. Sun, and Y. Q. Zhang. "Video transcoding: an overview of various techniques and research issues." *IEEE Transactions on Multimedia*, **7**(5), March 2005.

[Bar96] Haskell Barry. *An Introduction to MPEG-2.* Chapman & Hall, 115 Fifth Avenue, New York, NY, 1996.

[Bay08] S. Bayar. "Self-reconfiguration on Spartan-III FPGAs with compressed partial bitstreams via a parallel configuration access port (cPCAP) core." In *PH.D. Research in Microelectronics and Electronics*, pp. 137–140, 2008.

[BPK08] L. Braun, K. Paulsson, H. Kromer, M. Hubner, and J. Becker. "Data path driven waveform-like reconfiguration." In *Proceedings of International Conference on Field Programmable Logic and Applications*, pp. 607–610, Sept. 2008.

[CC99] S. Choi and S. Chae. "Hierarchical motion estimation in Hadamard transform domain." *Electronics Letters*, **35**(25), December 1999.

[CCH06] C. Y. Chen, S. Y. Chien, Y. W. Huang, T. C. Chen, T. C. Wang, and L. G. Chen. "Analysis and architecture design of variable block-size motion estimation for H.264/AVC." *IEEE Transactions on Circuits and Systems*, **53**(2), February 2006.

[CJC00] H. C. Chang, J. Y. Jiu, L. L. Chen, and L. G. Chen. "A Low Power $8 \times 8$ Direct 2-D DCT Chip Design." *Journal of VLSI Signal Processing*, **26**:319–332, 2000.

[CMZ07] C. Claus, F.H. Muller, J. Zeppenfeld, and W. Stechele. "A new framework to accelerate Virtex-II Pro dynamic partial self-reconfiguration." In *Proceedings of IEEE International Parallel and Distributed Processing Symposium*, pp. 1–7, March 2007.

[CP93] E. Chan and S. Panchanathan. "Motion estimation architecture for video compression." *IEEE Transactions on Consumer Electronics*, **39**(3):292–297, 1993.

[CSF77] W. H. Chen, C. Smith, and S. Fralick. "A fast computation algorithm for the discrete cosine transform." *IEEE Transactions on Communications*, **25**:1004–1009, 1977.

[CT08] C. Carmichael and C.W. Tseng. "Correcting single-event upsets in Virtex-4 platform FPGA configuration memory." *Xilinx Application Notes*, March 2008.

[CZM07] Christopher Claus, Johannes Zeppenfeld, Florian Müller, and Walter Stechele. "Using partial-run-time reconfigurable hardware to accelerate video processing in driver assistance system." In *DATE '07: Proceedings of the Conference on Design, automation and test in Europe*, pp. 498–503, San Jose, CA, USA, 2007. EDA Consortium.

[CZS08] C. Claus, B. Zhang, W. Stechele, L. Braun, M. Hubner, and J. Becker. "A multi-platform controller allowing for maximum dynamic partial reconfiguration throughput." In *Proceedings of International Conference on Field Programmable Logic and Applications*, pp. 535–538, 2008.

[DB03] T. Darwish and M. Bayoumi. "Energy aware distributed arithmetic DCT architectures." In *Proceedings of the IEEE Workshop Signal Processing Systems*, pp. 351–356, March 2003.

[Dip] Michael Dipperstein. "LZSS (LZ77) discussion and implementation." http://michael.dipperstein.com/lzss/.

[DKF05] A. Dehanhardt, M. B. Kulaczewski, L. Friebe, S. Moch, P. Pirsch, H.J. Stolberg, and C. Reuter. "A multi-core SOC design for advanced image and video compression." In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, pp. 18–23, March 2005.

[FGG09] A. Flynn, A. Gordon-Ross, and A.D. George. "Bitstream relocation with local clock domains for partially reconfigurable FPGAs." In *Proceedings of the IEEE/ACM Design, Automation and Test in Europe*, April 2009.

[GGA92] K. Guttag, R. J. Gove, and J. R. Van Aken. "A single chip multiprocessor for multimedia: The MVP." *IEEE Computer Graphics and Applications*, 1992.

[GHC04] D. Gong, Y. He, and Z. Cao. "New cost-effective VLSI implementation of a 2-D discrete cosine transform and its inverse." *IEEE Transactions on Circuits and Systems for Video Technology*, **14**(4), April 2004.

[GHP08] D. Gohringer, M. Hubner, T. Perschke, and J. Becker. "New dimensions for multiprocessor architectures: On demand heterogeneity, infrastructure and performance through reconfigurability - the RAMPSoC approach." In *Proceedings of International Conference on Field Programmable Logic and Applications*, pp. 495–498, 2008.

[GK07] S. Gharge and S. Krishnan. "Simulation and implementation of Discrete Cosine Transform for MPEG-4." In *Proceedings of the International Conference on Computational Intelligence and Multimedia Applications*, pp. 137–141, 2007.

[GSJ02] M. J. Garrido, C. Sanz, M. Jimenez, and J. M. Menesses. "An FPGA implementation of a flexible architecture for H.263 video coding." *IEEE Transactions on Consumer Electronics*, **48**, November 2002.

[GVB05] S. Ghosh, S. Venigalla, and M. Bayoumi. "Design and implementation of a 2D-DCT architecture using coefficient distributed arithmetic." In *Proceedings of the IEEE Computer Society Annual Symposium on VLSI*, pp. 162–166, May 2005.

[HB96] Haskell and Barry. *An introduction to MPEG-2*. Chapman&Hall, 115 Fifth Avenue, New York, NY, 1996.

[HG98] F. Hartung and B. Girod. "Watermarking of uncompressed and compressed video." *Signal Processing*, **66**(4), May 1998.

[HL04] C. H. Hsieh and T. P. Lin. "VLSI architecture for block-matching motion estimation algorithm." *IEEE Trans. Circuits Syst. Video Technol.*, **2**(2), 2004.

[ITU96] ITU-T Recommendation H.263. *Video coding for low bit rate communication(Annex A: Inverse transform accuracy specification)*, 1996.

[Jac08] Brian Jackson. "Partial Reconfiguration Design with PlanAhead." *Xilinx Inc*, March 2008.

[Khu01] A. Khu. *Xilinx FPGA configuration data compression and decompression*. Xilinx, Sept. 2001.

[KK99] K. Kim and J. Koh. "An area efficient DCT architecture for MPEG-2 video encoder." *IEEE Transactions on Consumer Electronics*, **45**, February 1999.

[KL98] U. Koc and K. J. Ray Liu. "Interpolation-free subpixel motion estimation techniques in DCT domain." *IEEE Transactions on Circuits and Systems for Video Technology*, **8**(4), August 1998.

[KMO04] A. Kinane, V. Muresan, N. O'Connor, N. Murphy, and S. Marlow. "Energy-efficient hardware architecture for variable N-point 1D DCT." In *Proceedings of International Workshop on Power and Timing Modeling, Optimization and Simulation*, pp. 780–788, May 2004.

[KP06] H. Kalte and M. Porrmann. "REPLICA2Pro: Task relocation by bitstream manipulation in Virtex-II/Pro FPGAs." In *Proceedings of the 3rd conference on Computing frontiers*, pp. 403–412, 2006.

[LBM06] P. Lysaght, B. Blodget, J. Mason, J. Young, and B. Bridgford. "Invited paper: Enhanced architectures, design methodologies and CAD tools for dynamic reconfiguration of Xilinx FPGAs." In *FPL*, pp. 1–6, 2006.

[LVI06] J. Lee, N. Vijaykrishnan, M. J. Irwin, and W. Wolf. "Motion vector refinement for high performance transcoding." *IEEE Transactions on Circuits and Systems for Video Technology*, **16**(2), Feburary 2006.

[Max04] Clive Maxfield. *The Design Warrior's Guide to FPGAs*. Elsevier, 2004.

[MTA07] M. Majer, J. Teich, A. Ahmadinia, and C. Bobda. "The Erlangen Slot Machine: A dynamically reconfigurable FPGA-based computer." *J. VLSI Signal Process. Syst.*, **47**:15–31, 2007.

[OBL04] J. Ostermann, J. Bormans, P. List, D. Marpe, M. Narroschke, F. Pereira, T. Stockhammer, and T. Wedi. "Video coding with H.264/AVC: tools, performance, and complexity." *Circuits and Systems Magazine, IEEE*, **4**(1):7 – 28, first 2004.

[PC05] Y. Huang H. Fang C. Huang P. Tseng, Y. Chang and L. Chen. "Advances in hardware architectures for image and video coding - a survey." *Proceedings of the IEEE*, **93**:184–197, 2005.

[PS99] I.M. Pao and M.T. Sun. "Modeling DCT coefficients for fast video encoding." *IEEE Transactions on Circuits and Systems for Video Technology*, **9**(4), 1999.

[QSL06] T. Qian, J. Sun, D. Li, X. Yang, and J. Wang. "Transform domain transcoding from MPEG-2 to H. 264 with interpolation drift-error compensation." *IEEE Transactions on Circuits and Systems for Video Technology*, **16**(4), April 2006.

[RR03] I. Richardson and J. E. Richardson. *H.264 and MPEG-4 Video Compression: Video coding for next-generation multimedia*. Wiley, John & Sons, Incorporated, 2003.

[SBB06] P. Sedcole, B. Blodget, T. Becker, J. Anderson, and P. Lysaght. "Modular dynamic reconfiguration in Virtex FPGAs." *IEE Proc. Computers and Digital Techniques*, **153**(3):157–164, May 2006.

[SK96] R. Swann and N. Kingsbury. "Transcoding of MPEG-II for enhanced resilience to transmission errors." In *Proceedings of the IEEE International Conference on Image Processing*, pp. 813–816, 1996.

[SMW07] 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*, **17**(9):1103–1120, 2007.

[SPC02] A. Shams, W. Pan, A. Chidanandan, and M. A. Bayouni. "A low power high performance distributed DCT architecture." In *Proceedings of the IEEE Computer Society Annual Symposium on VLSI*, pp. 21–27, April 2002.

[SRR06] F. Sun, S. Ravi, A. Raghunathan, and N. K. Jha. "Application-specific heterogeneous multiprocessor synthesis using extensible processors." *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, **25**:1589–1602, 2006.

[VCS03] A. Vetro, C. Christopoulos, and H. Sun. "Video transcoding architectures and techniques: an overview." *IEEE Signal Processing Magazine*, 2003.

[VKH00] R. J. van der Vleuten, R. P. Kleihorst, and C. Hentschel. "Low-complexity scalable DCT image compression." In *Proceedings of International Conference on Image Processing*, volume 3, pp. 837–840, Sept. 2000.

[WC07] T. Becker amn W. Luk and P.Y.K. Cheung. "Enhancing relocatability of partial bitstreams for run-time reconfiguration." In *Proceedings of the the IEEE International Symposium on Field-Programmable Custom Computing Machines*, pp. 35–44, April 2007.

[Whi89] Stanley A. White. "Applications of distributed arithmetic to digital signal processing: A tutorial review." *IEEE ASSP Magazine*, 1989.

[WKK07] H. Wang, S. Kwong, and C.W. Kok. "Efficient predictive model of zero quantized DCT coefficients for fast video encoding." *Image and Vision Computing*, **25**, 2007.

[WSB03] T. Wiegand, G.J. Sullivan, G. Bjontegaard, and A. Luthra. "Overview of the H.264/AVC video coding standard." *Circuits and Systems for Video Technology, IEEE Transactions on*, **13**(7):560 –576, july 2003.

[XC00] 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*, **35**(5), May 2000.

[Xil05] Xilinx Inc., San Jose, CA. *XAPP138 - Virtex FPGA series configuration and readback*, 2005.

[Xil06] Xilinx Inc., San Jose, CA. *Early access partial reconfiguration user guide*, 2006.

[XLS05] J. Xin, C. W. Lin, and M. T. Sun. "Digital video transcoding." *Proceedings of the IEEE*, **93**(1), 2005.

[YH95] H. Yeo and Y. H. Hu. "A novel modular systolic array architecture for full-search block matching motion estimation." *IEEE Transactions on Circuits and Systems for Video Technology*, **5**(5), October 1995.

[YSL99] J. Youn, M. T. Sun, and C. W. Lin. "Motion vector refinement for high performance transcoding." *IEEE Transactions on Multimedia*, **1**, March 1999.

[YSX99] J. Youn, M. T. Sun, and J. Xin. "Video transcoder architectures for bit rate scaling of H.263 bit streams." In *Proceedings of the Seventh ACM International Conference on Multimedia*, pp. 243–250, 1999.

[ZL77] J. Ziv and A. Lempel. "A universal algorithm for sequential data compression." *IEEE Transactions on Information Theory*, **23**(3):337–343, May 1977.

[ZYB98] W. Zhu, K. H. Yang, and M. J. Beacken. "CIF-to-QCIF video bitstream down-conversion in the DCT domain." *Bell Labs Technical Journal*, **3**(3), July 1998.