Low Power VLSI Implementation Schemes For DCT-Based Image Compression

Shedden Masupe

A thesis submitted for the degree of Doctor of Philosophy.
The University of Edinburgh.
July 2001

EX BIBL. UNIV. EDIN.

Abstract

With the ever increasing need for portable electronic devices, there is a growing need for a new look at very large scale integration (VLSI) design in terms of power consumption. Most of the research and development efforts have focused on increasing the speed and complexity of single chip digital systems. In other words, focusing on systems that can perform most computations in given amount of time. However, area and time are not the only metrics that can measure implementation quality. Power consumption has now entered the field , so designers need to take another important parameter as a third dimension in order to enhance the device capabilities.

The Discrete Cosine Transform (DCT) is the basis for current video standards like H.261, JPEG and MPEG. Since the DCT involves matrix multiplication, it is a very computationally intensive operation. Matrix multiplication entails repetitive sum of products which are carried out numerous times during the DCT computation. Therefore, as a result of the multiplications, a significant amount of switching activity takes place during the DCT process. This thesis proposes a number of new implementation schemes that reduce the switching capacitance within a DCT processor for either JPEG or MPEG environment.

A number of new generic schemes for low power VLSI implementation of the DCT are presented in this thesis. The schemes target reducing the effective switched capacitance within the datapath section of a DCT processor. Switched capacitance is reduced through manipulation and exploitation of correlation in pixel and cosine coefficients during the computation of the DCT coefficients. The first scheme concurrently processes blocks of cosine coefficient and pixel values during the multiplication procedure, with the aim of reducing the total switched capacitance within the multiplier circuit. The coefficients are presented to the multiplier inputs as a sequence, ordered according to bit correlation between successive cosine coefficients. The ordering of the cosine coefficients is applied on the columns. Hence the scheme is referred to as *column-based* processing. Column-Based processing exhibits power reductions of up to 50% within the multiplier unit.

Another scheme, termed *order-based*, is based on the ordering of the cosine coefficients based on row segments. The scheme also utilises bit correlation between successive cosine coefficients. The effectiveness of this scheme is reflected in power savings of up to 24%. The final scheme is based on manipulating data representation of the cosine coefficients, through cosine word coding, in order to facilitate for a shift-only computational process. This eliminates the need for the multiplier unit, which poses a significant overhead in terms of power consumption, in the processing element. A maximum power saving of 41% was achieved with this implementation.

Declaration of originality

I hereby declare that the research recorded in this thesis and the thesis itself was composed and originated entirely by myself in the Department of Electronics and Electrical Engineering at The University of Edinburgh.

Shedden Masupe

Acknowledgements

I would like to thank my supervisor Dr T. Arslan, for his assistance and guidance throughout the project.

I would also like to thank all the guys in the SLI activity for their continuous assistance in the subject matter of various aspects of the project. In particular, Damon Thompson for his ever keen assistance with VHDL/Verilog programming and nifty digital design tips. I have to thank Dr Ahmet Erdogan for his inspirational assistance in the low power design techniques. Baba Goni, he had so much faith in me even at times that I doubted myself, I appreciate that. I thank Prof Alan F. Murray for his input in the thesis. The acknowledgements will be incomplete if I do not mention Barnabas Gatsheni, his probing questions about image processing solidified my understanding for the basic concepts. I have to thank Dr Joseph Chuma for his support throughout the programme.

My thanks go to my sponsors, the University of Botswana and Edinburgh University.

I still feel the urge to thank my parents, they did a good job.

Finally I would like to thank Tiny, Yoav and Ailsa. They were very patient and understanding to my absence at home.

Dedication

to my wife

Tiny

Contents

List of figures

List of tables

Acronyms and abbreviations

ARPA	Advanced Research Projects Agency
CIP	Cell Internal Power
CMOS	Complementary Metal Oxide Semiconductor
CLA	Carry Look Ahead
CSA	Carry Save Array
DA	Distributed Arithmetic
DC	Direct Current
DCT	Discrete Cosine Transform
DFT	Discrete Fourier Transform
DST	Discrete Sine Transform
FDCT	Forward Discrete Cosine Transform
FFT	Fast Fourier Transform
FSM	Finite State Machine
GOP	Group of Pictures
H.261	Recommendation of International Telegraph and Telephone Consultive Committee
HDTV	High Defination Television
IP	Intellectual Property
ISDN	Integrated Services Digital Network
JPEG	Joint Photographic Expert Group
KLT	Karhunen-Loeve Transform
LSI	Linear Shift Invariant
MAC	Multiply and Accumulate
MOPS	Mega Operations Per Second
MOS	Metal Oxide Semiconductor
MPEG	Moving Photographic Expert Group
MSB	Most Significant Bit
NMOS	Negative-Channel Metal Oxide Semiconductor
NSP	Net Switching Power
NTSC	National Television Standards Committee
Pixel	Picture elements
PMOS	Positive-Channel Metal Oxide Semiconductor
RAM	Random Access Memory
ROM	Read Only Memory
SAIF	Switching Activity Interchange Format
TDP	Total Dynamic Power
VHDL	Very-high-speed Hardware Description Language

VLC	Variable Length Coding
VLSI	Very Large Scale Intergration
WHT	Walsh-Hadamard Transform

Nomenclature

C_{eff}	effective capacitance
C_{load}	load capacitance
cis	cell internal power
f	switching frequency
I_{sc}	short-circuit current
I_{sw}	switching current
k	switching activity factor
L	transistor length
nsp	net switching power
P_{static}	static power dissipation
$P_{dynamic}$	dynamic power dissipation
P_{tot}	total power dissipation
P_{sw}	switching power
P_{sc}	short circuit power
$P_s(x)$	signal probability
$P_t(x)$	transition probability
tdp	total dynamic power
V_{dd}	supply voltage
V_{gs}	gate-source voltage
V_{in}	input voltage
V_{out}	output voltage
V_T	thermal voltage
V_{tn}	threshold voltage for NMOS transistor
V_{tp}	threshold voltage for PMOS transistor
W	transistor width

List of Publications

1. S. Masupe and T.Arslan. “Low Power DCT implementation approach for CMOS based DSP Processors,”IEE Electronics Letters, volume 34, pp.2392-2394, Dec. 1998.

2. S. Masupe and T.Arslan. “Low Power DCT implementation approach for VLSI based DSP Processors,” in International Symposium on Circuits and Systems, IEEE, 30 May - 2 June 1999.

3. T. Arslan, A. T. Erdogan, S. Masupe, C. Chan-Fu, D. Thompson, ”Low Power IP Design Methodology for Rapid Development of DSP-Intensive SoC Platform”, IP99 Europe, 2-3 November 1999, Edinburgh, UK. pp. 337-346

4. S. Masupe and T.Arslan. “Low Power VLSI implementation of the DCT on Single Multiplier DSP Processors,” VLSI Design: An International Journal of Custom-Chip Design, Simulation and Testing, Volume 11, Number 4, pp. 397-403, 2000.

5. S. Masupe and T.Arslan. “Low Power Order Based DCT Processing Algorithm,”in International Symposium on Circuits and Systems, IEEE, Sydney, Australia, volume 2, pp. 5-8, 7-9 May 2001.

Chapter 1 Introduction

1.1 Introduction

With the ever increasing need for portable electronic devices, there is a growing need for a new look at very large scale integration (VLSI) design in terms of power consumption. Most of the research and development efforts have focused on increasing the speed and complexity of single chip digital systems. In other words, focusing on systems that can perform most computations in given amount of time. However, area and time are not the only metrics that can measure implementation quality. Power consumption has now entered the field , so designers need to take another important parameter as a third dimension in order to enhance the device capabilities. Consumers would like their state of the art portable electronic devices to operate for long periods of time without loss of power.

Some of the portable consumer electronic devices include lap-top computers, cellular phones and pagers. Work has been done on battery technologies to increase the battery life. However, some devices like the upcoming portable multi-media terminal, if built using off-shelf components which are not designed for low power consumption, will require batteries of at least 3 kg in weight to operate for 10 hours without re-charge. Therefore, without low power design techniques, existing and upcoming portable devices will be very bulky due to battery packs or they will have a very short battery life if battery size is reduced[1]

A growing number of computer systems are incorporating multi-media capabilities for displaying and manipulating video data. This interest in Multi-media combined with the great popularity of portable computers and portable phones provides the impetus for creating a portable video on demand system, This requires a bandwidth far greater than the ordinary broadcast video, since a user can subscribe to different video programs at any time wherever they are. Therefore an enormous bandwidth is required for storage and transmission, data must be compressed in real time for the portable unit[1].

Digital video applications are some of the popular devices that have become part of the every day life. For example, ISDN video-phone, video-conference systems, digital broadcast HDTV and remote surveillance [2]. The Discrete Cosine Transform (DCT) is the basis for current video standards like H.261, JPEG and MPEG. Since the DCT involves matrix multiplication, it is a very computationally intensive operation. Matrix multiplication entails repetitive sum of products which are carried out numerous times during the DCT computation. Therefore, as a result of the multiplications, a significant amount of switching activity takes place during the DCT process. This thesis proposes a number of new implementation schemes that reduce the switching capacitance within a DCT processor for either JPEG or MPEG environment.

1.2 Thesis Contribution

A number of new generic schemes for low power VLSI implementation of the DCT are presented in this thesis. The schemes target reducing the effective switched capacitance within the datapath section forward DCT processor(FDCT). Switched capacitance is reduced through manipulation and exploitation of correlation in pixel and cosine coefficients during the computation of the DCT coefficients. The techniques are generic and can be extended to include other applications where matrix multiplication is required. Some DCT architectures which suit the schemes are proposed as proof of concept, and some power dissipation measures are given for the data path components.

The first scheme, *column-based*, reduces switched capacitance through manipulation and exploitation of correlation in pixel and cosine coefficients during the computation of the DCT coefficients. The manipulation is of the form of ordering the cosine coefficients per column, according to some ordering technique such as, ascending order or minimum hamming distance and processing the matrix multiplication using a column-column approach instead of the usual row-column approach. This scheme achieves a power reduction of up-to 50% within the multiplier section of the DCT implementation.

The second scheme, *order-based*, takes advantage of the rows in the cosine matrix. Ordering the cosine coefficients in each row according to some ordering technique such as, ascending order or minimum hamming distance, minimises the toggles at the inputs of the multiplier unit in the DCT processor. The above techniques are generic and can be extended to include other applications where matrix multiplication is required. The effectiveness of this scheme is reflected in power savings of up-to 24% within the MAC unit of the DCT implementation.

The third scheme proposes coding cosine coefficients such that only shift operations are used to process the DCT computation. In this scheme the need for a standard multiplication unit is eliminated by formating the data representation of the cosine elements such that the pixel values are processed using a shifter unit and an addition unit. Power savings resulting due to this scheme can go up-to 41%.

1.3 Thesis Layout

Chapter 2 introduces the basic concepts of low power CMOS design. It provides definitions and equations for static and dynamic power. Some of the techniques which can be used to minimise power dissipation in CMOS circuits are also presented. These techniques are more relevant to reducing power consumption which results from one of the major components of dynamic power dissipation, switching power.

Fundamentals of image processing and image compression are introduced in chapter 3. The chapter also goes on to cover some image transforms which are possible competitors of the DCT. Since the DCT is key to the project, a more detailed introduction of the transform is presented. This includes several algorithms and architectures which have been developed since the creation of the DCT.

To conclude the literature review, chapter 4 presents a summary of low-power DCT research covered before (or during the course of) this work.

The next four chapters introduce the three new schemes for low power implementation of the DCT. Chapter 5 presents the first scheme, which is termed *column-based*. This involves the power analysis of the multiplier section in the proposed scheme environment since this unit is well-known for its computation intensive nature. Column-Based scheme is further investigated in Chapter 6 by mapping the algorithm resulting from the scheme into an architecture suitable for VLSI implementation.

The second scheme is presented in chapter 7 and this takes advantage of the rows in the cosine matrix. This scheme is termed *order-based.* Ordering the cosine coefficients in each row according to some ordering technique such as, ascending order or minimum hamming distance, minimises the toggles at the inputs of the multiplier unit in the DCT processor.

Chapter 8 introduces the third and final scheme. This is a DCT implementation which codes the cosine coefficients in order to reduce computational complexity by using shifters and adders only in the processing.

Finally, chapter 9 concludes the thesis and puts forward some suggestions for future work to be done in order to continue this research.

Chapter 2
Low Power CMOS Design

2.1 Introduction

This chapter introduces the basic concepts of Low Power CMOS Design. It provides definitions and equations for static and dynamic power. Some of the techniques which can be used to minimise power dissipation in CMOS circuits are also presented. These techniques are more relevant to reducing power consumption which results from one of the major components of dynamic power dissipation, switching power. Some power estimation methodologies are presented as well.

2.2 Sources of Power Consumption

The dominant source of power consumption in digital CMOS circuits is the switching power, which is caused by periodic charging and discharging of nodal capacitances. There are two kinds of power dissipation in CMOS circuits, static and dynamic. The total power consumption can be categorised as shown in Equation 2.1.

$$P_{tot} = P_{static} + P_{dynamic} \tag{2.1}$$

The static power can be ignored since it can be minimised using current technologies. The Dynamic power however is still a major source of dissipation in CMOS circuits and it only occurs when a node voltage is switched.

2.2.1 Static Power Dissipation (P_{static})

Static power is the power dissipated by a gate when it is not in operation, that is when it is not switching [3]

Ideally, CMOS circuits dissipate no static (DC) power since in the steady state there is no direct path from V_{dd} to ground(see Figure 2.1). Of course, this scenario can never be realised in practice since in reality the MOS transistor is not an ideal switch. Hence there will always be some leakage currents and substrate injection currents which will give rise to a static component of the CMOS power dissipation.

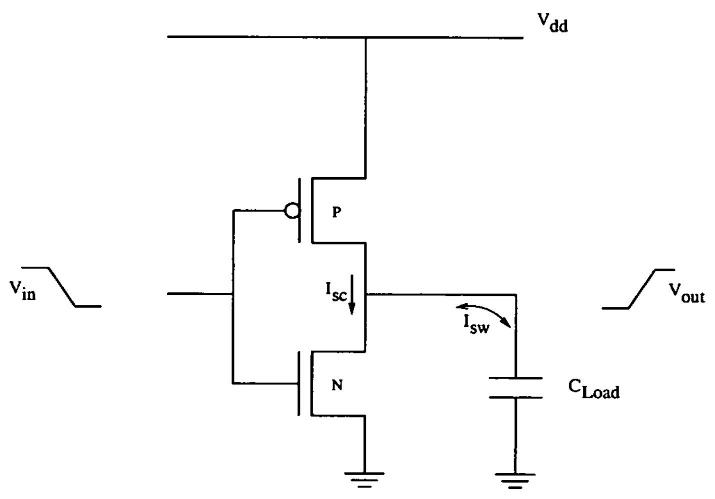


Figure 2.1: *Typical power consumption parameter profiles*

Leakage Current Power

There are two types of leakage currents,

1. reverse-bias diode leakage - at the transistor drains
2. sub-threshold leakage - through the channel of a device that is turned off.

The magnitudes of the leakage currents is determined mainly by the processing technology.

diode leakage

This occurs when a transistor is turned off. For the case of an inverter, when the PMOS transistor is turned off with a high input voltage, there will be a voltage drop of $-V_{dd}$ between the drain and the bulk after the V_{dd} to 0 transition at the output. This results in the diode leakage expressed as [4]:

$$I_d = I_s\left(e^{\frac{V}{V_T}} - 1\right) \tag{2.2}$$

where I_s is the reverse saturation current, V is the diode voltage and V_T is the thermal voltage. A typical value of the leakage current is $1fA$ per device junction. The value is too small to have any impact on the overall power consumption. For example, for a chip with a million devices, the total power dissipation contributed due to the leakage current will be *approx* 0.01μW [4, 5].

sub-threshold leakage

This occures due to current diffusion between the source and the drain when the gate-source voltage (V_{gs}) exceeds the weak inversion point, but still being below the threshold voltage (V_t) [4, 5].

$$I_{ds} = I_o \frac{W_{eff}}{W_o} 10^{\frac{(V_m - V_T)}{S}} \tag{2.3}$$

where

$$I_o = I_o'(1 - e^{\frac{V_{DS}}{V_t}}) \tag{2.4}$$

and V_T is the constant-current threshold voltage. W_o and I_o are the gate width and the drain current to define V_T. S is the subthreshold swing parameter. The effective channel width is referred to as W_{eff} [5].

The magnitude of the sub-threshold is a function of the process (device sizing) and supply voltage. The process parameter, V_t, predominantly affects the current. Reducing V_t has an exponential increasing effect on the sub-threshold current. The sub-threshold current is a proportional to $\frac{W}{L}$ (device size), and is an exponential function of the input voltage. Hence, the sub-threshold current can be minimised by the reduction of transistor size and the supply voltage [4].

2.2.2 Dynamic Power Dissipation ($P_{dynamic}$)

The dynamic power dissipation comprises of two main sources:-

1. Switching power (P_{sw}) due to the charging and discharging circuit capacitances

2. Short-circuit power (P_{sc}) current due to finite signal rise/fall times .

$$P_{dynamic} = P_{sw} + P_{sc} \tag{2.5}$$

where

$$P_{sw} = \frac{1}{2} \cdot k \cdot C_{load} \cdot V_{dd}^2 \cdot f \tag{2.6}$$

and where k is the number of energy consuming transitions. C_{load} is the physical capacitance. $V_{dd}$is the supply voltage. f is the data rate, it describes how often on average switching could occur. For synchronous systems, f might correspond with the clock frequency.

Short-circuit Power (P_{sc})

The dynamic part of power dissipation is a result of the transient switching behaviour of the CMOS circuit. There exists a point whereby both the NMOS and PMOS in Figure 2.1 will be on. During that moment, a short circuit exists between V_{dd} and the ground allowing currents to flow. Figure 2.2 illustrates the transition from 0 to V_{dd} and V_{dd} to 0. The input voltage V_{in}

obeys the following principle [4],

$$V_{tn} < V_{in} < V_{dd} - |V_{tp}| \tag{2.7}$$

where V_{tn} and V_{tp} are threshold voltages for NMOS and PMOS transistors respectively. A long input rise or fall time implies that the short-circuit current will flow longer.

Short circuit power dissipation is especially significant when the output rise/fall time is less than the input rise/fall time. This is the case when the load capacitance C_{load} (C_L), is small. In this situation, when the inverter gate makes a 0 to V_{dd} transition with a finite slope, the drain terminal of the PMOS transistor is immediately grounded and a current from the power supply to the ground flows through. On the one hand if C_L is very high, the output rise/fall time becomes much greater than the input rise/fall time. Hence during an input transition, whereby the input follows the principle in Equation 2.7, the output remains at V_{dd}. Therefore, there will be no voltage drop across the source and drain terminals of the PMOS transistor. This implies that there will be no current drawn [4]. See Figure 2.3.

In conclusion, short circuit power dissipation can be reduced by making certain that the output rise/fall time is larger than the input rise/fall time. There is, however, a problem with making out rise/fall time too large. It slows down the circuit and can potentially cause short circuit currents in the fan out gates.

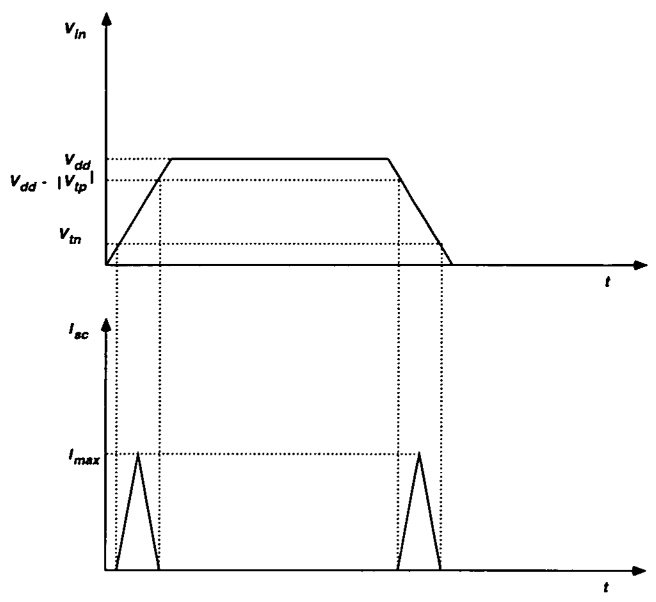


Figure 2.2: *Input voltage and short circuit current*

This short circuit dissipation can be kept below 10% of the total dynamic power dissipation with careful design. This is achieved by keeping the rise and fall times of all signals throughout the design within a fixed range [6, 7].

Switching Power (P_{sw})

The dominant component of the dynamic power is the switching power which is the result of charging and discharging parasitic capacitances in the circuit. The case is modelled in Figure 2.3 where the parasitics are lumped together at the output in the capacitor C_{load}.

The average switching power P_{sw} required to charge and discharge the capacitance C_L at a switching frequency of $f = \frac{1}{T}$ can be computed as:

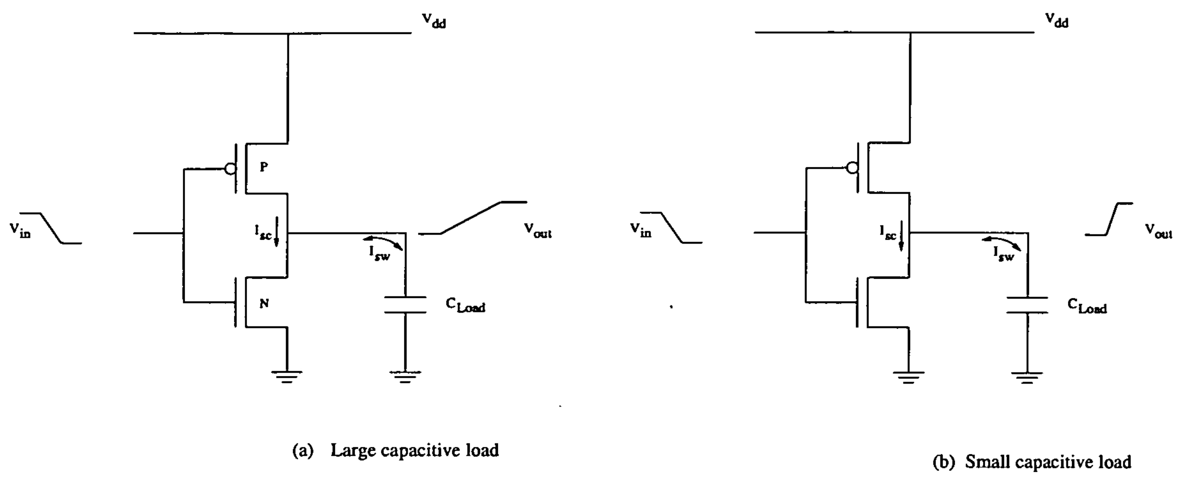


Figure 2.3: *Impact of load capacitance*

$$P_{sw} = \frac{1}{T}\int_0^T i_o(t)v_o(t)dt \tag{2.8}$$

The current at the output during the charging phase can be presented as

$$i_o = i_p = C_L\frac{dv_o}{dt} \tag{2.9}$$

whereas while discharging the current is given by

$$i_o = i_n = -C_L \frac{dv_o}{dt} \tag{2.10}$$

where i_n and i_p are NMOS and PMOS currents respectively [4].

Substituting Equations 2.9 and 2.10 into the average switching power equation 2.8, results in the average switching power for the inverter gate as:

$$\begin{aligned} P_{sw} &= \frac{1}{T}\left[\int_0^{V_{dd}} C_L v_o dv_o - \int_{V_{dd}}^0 C_L v_o dv_o\right] \\ &= \frac{C_L V_{dd}^2}{T} = C_L V_{dd}^2 f \end{aligned} \tag{2.11}$$

and the energy being drawn from the power supply is

$$E = \int_0^T P(t)dt = C_L V_{dd}^2 \tag{2.12}$$

It can also be shown that the energy being drawn by the load capacitance C_L is

$$\begin{aligned} E_L &= \int_0^{V_{dd}} C_L \frac{dv_o}{dt} v_o dt \\ &= C_L \int_0^{V_{dd}} v_o dv_o = \frac{1}{2} C_L V_{dd}^2 \end{aligned} \tag{2.13}$$

This implies that during a transition 0 to V_{dd}, one half of the energy drawn from the power supply is stored in the capacitor and the other half is used up by the PMOS pull-up network. The other transition V_{dd} to 0, results in the energy stored by the capacitor (Equation 2.13) being used by the NMOS pulled down network. Therefore, from the above analysis, it can be summarised that every time a capacitive node switches from ground to V_{dd}, an energy equivalent to Equation 2.13 is consumed [4].

This leads to the conclusion that CMOS power consumption depends on the switching activity of the signals involved. In this context, we define activity k, as the expected number of zero to one transitions per data cycle. If this is coupled with the average data-rate, f, which may be the clock frequency in a synchronous system, then the effective frequency of nodal charging is given by the product of the activity and the data rate: kf This leads to the equation for average CMOS power consumption shown in Equation 2.6 [4].

The resulting equation illustrates that the dynamic power is directly proportional to the switching activity, capacitive loading, and the square of the supply voltage [6]. This component of dynamic power, switching power, is the most dominant of the total power dissipation. It can amount to 80% of the total power consumption in circuit datapaths in modules like multipliers and adders.

2.3 Switching Power Reduction Techniques

2.3.1 Reducing Voltage

With its quadratic relationship to power, voltage reduction offers the most direct and dramatic means of minimising energy consumption. Without requiring any special circuits or technologies, a factor of two reduction in supply voltage (V_{dd}) yields a factor of four decrease in energy. This power reduction is a global effect, experienced not only in one sub-circuit or block of the chip, but throughout the entire design. Because of this quadratic relationship, designers are often willing to sacrifice increased physical capacitance or circuit activity for reduced voltage. Despite the obvious advantage, voltage reduction is detrimental to performance of the system [8].

As the supply voltage is lowered, circuit delays increase leading to reduced system performance. For $V_{dd} >> V_t$ delays increase linearly with decreasing voltage. In order to meet system performance requirements, these delay increases cannot go unchecked. Some techniques must be applied, either technological or architectural to compensate for this effect. This works well until V_{dd} approaches the threshold voltage at which point delay penalties simply become unmanageable. This tends to limit the advantageous range of the voltage supplies to a minimum of about $2V_t$.

Performance is not, however the only limiting factor. When going to non-standard voltage supplies, there is also the issue of compatibility and inter-operability. Most off-the-shelf components operate either on 5V supply or 3.3 V. Unless the entire system is being designed completely from scratch it is likely that some amount of communications will be required with components operating at a standard voltage. The severity of this problem is reduced by the availability of highly efficient DC-DC level converters, but still there is some cost involved in supporting several different voltages [6].

Another issue that arises with the reduction of voltage is that, more designs are now implemented as IP soft cores, this means that the operating voltage will be determined by the foundry that supplied the library cells, hence it can not be altered [9].

2.3.2 Reducing Physical Capacitance

This is yet another degree of freedom which can be utilised to reduce the dynamic power dissipation. In order to consider this possibility we must first understand what factors contribute to the physical capacitance of a circuit.

The physical capacitance in CMOS circuits comes from two basic sources, devices and interconnect. Previous technologies had more problems with device capacitance than interconnect parasitics. Since the technologies have scaled down a lot, interconnect parasitics contribute a lot to the overall physical capacitance and hence they need to be addressed.

From the previous discussion, it can be recognised that capacitances can be kept at a minimum by using less logic, smaller devices, fewer and shorter wires. Example techniques for reducing the active area include resource sharing, logic minimisation and gate sizing. Example techniques for reducing interconnect include register sharing, common sub-function extraction, placement and routing. As with voltage however, there are disadvantages to capacitive loading reduction. For example, reducing device sizes not only reduces physical capacitance, but also reduces the current drive of the transistors making the circuit operate more slowly. This loss of performance might hinder the reduction of V_{dd} to a value which would have otherwise been possible [6].

2.3.3 Reducing Switching Activity

Another candidate for dynamic power dissipation reduction in CMOS circuits is the reduction of switching activity. A chip can contain a huge amount of physical capacitance, but if it does not switch then no dynamic power will be consumed. The activity determines how often this switching occurs. As mentioned before, there are two components to the switching activity, the data rate (f), and the data activity (k). f describes how often on average switching could occur. In synchronous systems, f might correspond with the clock frequency. See Figure 2.4. The other component, k, corresponds to the expected number of energy consuming transitions that will be triggered by the arrival of each new piece of data. Hence while f determines the average periodicity of data arrivals, k determines how many transitions each arrival will spark. For circuits that do not experience *glitching*, k can be interpreted as the probability that an energy consuming transition will occur during a single data period. Even for these circuits, calculation of k is difficult as it depends not only on the switching activities of the circuit inputs and the logic function computed by the circuit, but also on the spatial and temporal correlations among the circuit inputs.

For certain logic styles, however, glitching can be an important source of signal activity and therefore, deserves some mention here. Glitching refers to the spurious and unwanted transitions that occur before a node settles down to its final steady-state value. Glitching often arises when paths with unbalanced propagation delays converge at the same point in the circuit. Calculation of this spurious activity in a circuit is very difficult and requires careful logic and/or circuit level characterisation of the gates in a library as well as detailed knowledge of the circuit structure. Since glitching can cause a node to make several power consuming transitions instead of one, it should be avoided whenever possible [6].

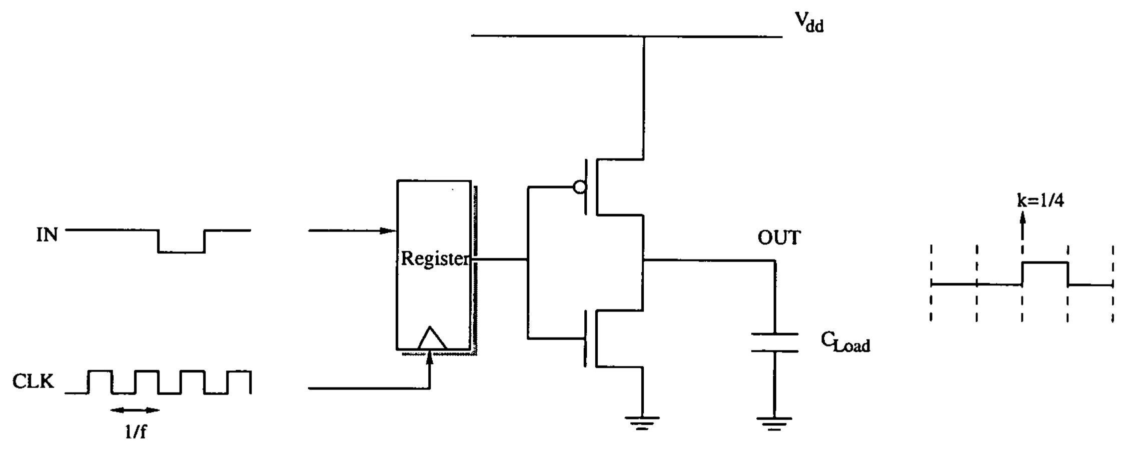


Figure 2.4: *Switching activity in synchronous systems*

The data activity k can be combined with the physical capacitance C_{load} to obtain an effective capacitance C_{eff}, which describes the average capacitance charged during each $1/f$ data period. This reflects the fact that neither the physical capacitance nor the activity alone determine dynamic power consumption. Instead, it is the effective capacitance, which combines the two, that truly determines the power consumed by a CMOS circuit. Therefore :

$$P_{sw} = \frac{1}{2} \cdot k \cdot C_{load} \cdot V_{dd}^2 \cdot f = C_{eff} \cdot V_{dd}^2 \cdot f \tag{2.14}$$

Evaluating the effective capacitance of a design is non-trivial as it requires a knowledge of both the physical aspects of the design(ie technology parameters, circuit structure, delay model) as well as the signal statistics(ie data activity and correlations). This explains why, lacking proper tools, power analysis is often deferred to the latest stages of the design process or is only obtained from measurements on the finished parts.

Some techniques for reducing switching activity include power-conscious state encoding and multi-level logic optimisation for FSM's. Another example will be certain data representations such as sign magnitude have an inherently lower activity than two's compliment. Since sign magnitude arithmetic is much more complex than two's compliment, however, there is a price to be paid for the reduced activity in terms of higher physical capacitance. This is yet another indication that low power design is a joint optimisation problem.

A summary of techniques that can be used to reduce effective switched capacitance is presented in Table 2.1.

Abstraction Level	Examples
System	Power Down, System Partitioning
Algorithm	Complexity, Concurrency Locality, Regularity Data representation
Architecture	Concurrency, Data Representation Signal correlations Instruction set selection
Circuit/Logic Physical Design	Transistor Sizing, Power Down Logic optimisation, layout Optimisation
Technology	Advanced Packaging

Table 2.1: *Power reduction techniques*

2.4 Power Estimation Techniques

This section presents the case of power estimation and it introduces some of the probabilistic and statistical measures used in power estimation.

Power estimation in general refers to the estimation of average power dissipation of a circuit. The most straight-forward method for power estimation is through simulation. That is, performing a circuit simulation of the design and monitoring the power supply current waveform. The average current is calculated and used to provide the average power. This method has an advantage of accuracy and generality. The technique can be used to estimate power of any circuit, regardless of the technology, design style, architecture etc. However, complete and specific information about the input signals is required. Hence the simulation based technique strongly depends on input patterns [7].

The pattern dependency poses a problem in the sense that often power estimation of a functional block is performed before the rest of the design is complete. In this case, little is known about the inputs to the functional block. Therefore, complete and specific information cannot be provided.

Some methods of power estimation have been proposed [7, 10–12]. These techniques simplify the problem by making three assumptions:

1. Assume that the power supply and ground voltage levels are fixed for the entire design.

   - this makes it easier to compute the power by estimating the current drawn by every sub-circuit assuming a fixed voltage.

2. Assume that the circuit is built up using logic gates and registers, see Figure 2.5

   - for this case, the power dissipation of the circuit can be broken down into two components

   - power consumed by registers
   - power consumed by the combinational block

3. Assume it is sufficient to consider only the charging/discharging current drawn by a logic gate

   - neglecting short-circuit current

Referring to Figure 2.5, whenever the clock triggers the registers, some of them will make transitions and hence draw power. Therefore the power consumed by registers depends on the clock. For the combinational block, the internal gates may make several transitions before settling down to their steady state values for that clock period. The additional transitions

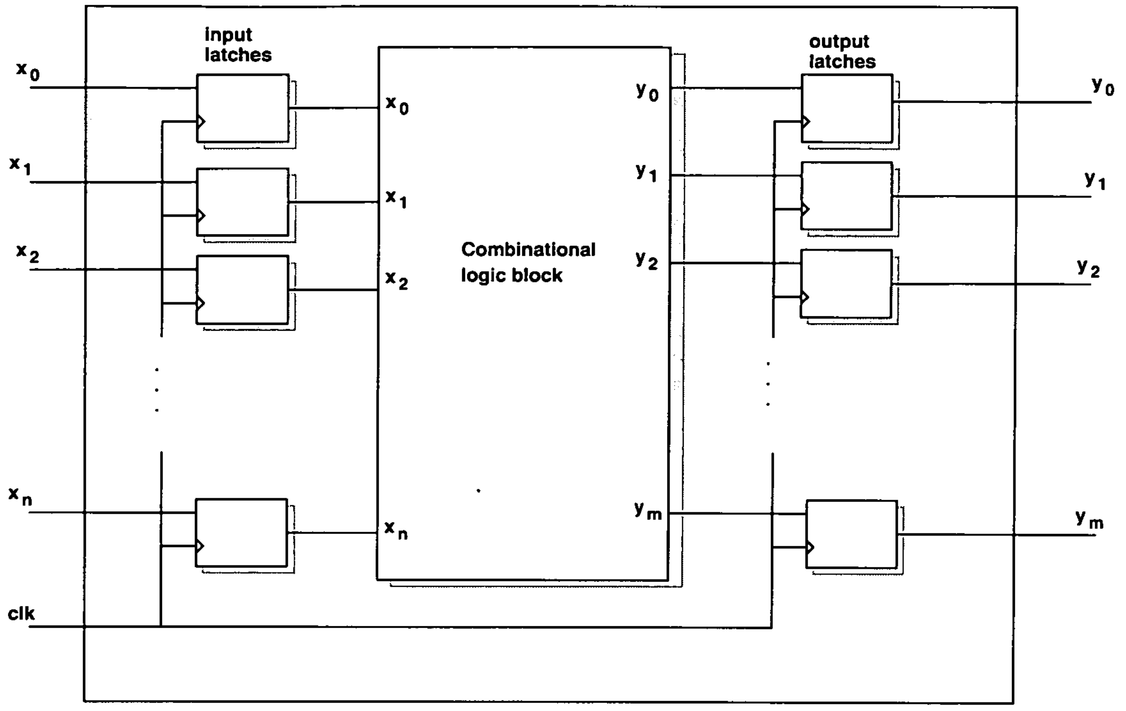


Figure 2.5: *A typical synchronous sequential design*

are called glitches. These are not necessarily design errors, but they are a problem for low-power design since additional power is being dissipated. This additional power can easily go up-to 20% of the total power dissipation in a circuit. Estimating the '*glitch power*' can be computationally expensive, this leads to most power estimation techniques ignoring it [7].

Instead of simulating the circuit for a large number of input patterns, and then averaging the results, the fraction of cycles in which an input signal makes a transition can be computed and used to estimate how often internal nodes make transitions. This fraction is a probability measure. Figure 2.6 shows both the conventional circuit simulation-based power estimation and the probability-based power estimation.

There are several ways of defining probability measures associated with the transitions made

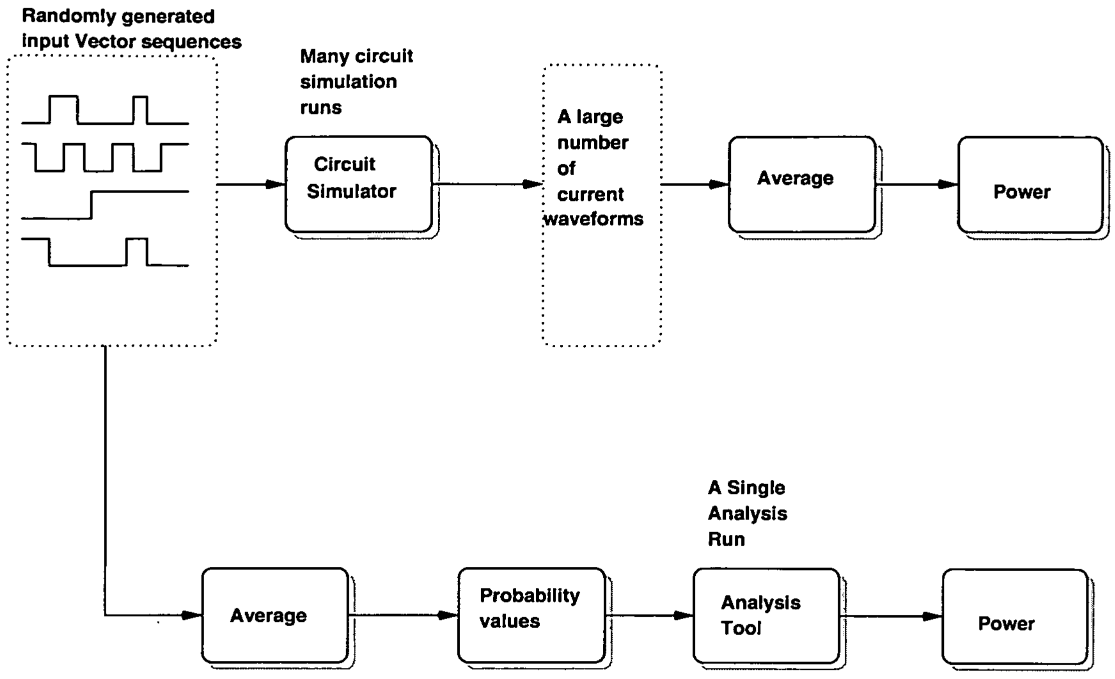


Figure 2.6: *A power estimation flow*

by a logical signal. This is the case for both the primary inputs of the combinational block and an internal node. These definitions are as follows:

signal probability $P_s(x)$

- at a node x, is the average fraction of the clock cycles in which the steady state value of x is a logic high

transition probability $P_t(x)$

- at a node x, is the average fraction of clock cycles in which the steady state value of x is different from its initial state

Both the above probability measures are not affected by the circuit internal delay. This implies that they are the same even if a zero-delay timing model is assumed. However, when zero-delay is assumed, the glitch power is excluded from the analysis.

When assuming zero-delay model and the transition probabilities are calculated, the power can be computed as [7]:

$$P_{av} = \frac{1}{2T_c} V_{dd}^2 \sum_{i=1}^{n} C_i P_t(x_i) \tag{2.15}$$

where T_c is the clock period, n is the total number of nodes in the circuit and C_i is the total capacitance at node x_i. Because this assumes at most a single transition per clock cycle, then it is the lower limit on the true average power [7].

In practice, it may occur that two signals are never high simultaneously. Computing this type of correlation can be very expensive, hence circuit input and internal nodes are usually assumed independent. This is referred as *spatial independence*. Another independence issue can be termed as *temporal independence*. This results in an assumption that the values of the same signal in two consecutive clock cycles are independent. Assuming temporal independence, the transition probability can be obtained from the signal probability as follows [7]:

$$P_t = 2P_s(x)P_s(\overline{x}) = 2P_s(x)[1 - P_s(x)] \tag{2.16}$$

transition density

- if a logical signal $x(t)$ makes $n_x(T)$ transitions in a time interval of length T, then

$$D_x \cong \lim_{T \to \infty} \frac{n_x(T)}{T} \tag{2.17}$$

The transition density provides a useful measure of switching activity in logic circuits. If the transition density of every node in a circuit can be computed, the overall power dissipation of the circuit can be calculated as :

$$P_{av} = \frac{1}{2} V_{dd}^2 \sum_{i=1}^{n} C_i D(x_i) \tag{2.18}$$

where $D(x_i)$ is the transition density at node x_i.

For a synchronous circuit, the relationship between transition density and transition probability is [7]:

$$D_x \geq \frac{P_t(x)}{T_c} \tag{2.19}$$

equilibrium probability

- the average fraction of time that the signal is high - if $x(t)$ is a logical signal, then its equilibrium probability is [7]:

$$P_x \cong \lim_{T \to \infty} \frac{1}{T} \int_{\frac{-T}{2}}^{\frac{T}{2}} x(t)dt \tag{2.20}$$

The equilibrium probability depends on the circuit internal delays since it describes the behaviour of the signal over time, not the steady state behaviour per clock. For steady state conditions, the equilibrium probability reduces to signal probability [7].

Other techniques, which use traditional simulation models and simulate the circuit for a limited number of randomly generated input vectors while monitoring power are named 'statistical-based'. The input vectors are generated from user specific probabilistic information about the circuit inputs.

The techniques introduced above, probabilistic and statistical techniques, are only applicable to combinational circuits. They require activity information at the register outputs to be specified by the user.

2.4.1 Probabilistic Techniques

There are several power estimation approaches which have been proposed to alleviate the input pattern dependency problem. These techniques use probabilities to solve the problem.

The techniques proposed are only applicable to combinational circuits and they require the user to specify the typical behaviour of the circuits at the combinational circuit inputs. Sample techniques evaluated are rated according to the following criteria [7]. See Table 2.2

1. glitch power handling
2. temporal correlation handling
3. complexity of the input specification
4. individual gate power provision
5. spatial correlation handling
6. speed

Approach	Handle glitch Power	Handle temporal correlation	Input Specification	Individual gate power	Handle spatial correlation	Speed
signal probability	No	No	Simple	Yes	No	Fast
CREST	Yes	Yes	Moderate	Yes	No	Fast
DENSIM	Yes	Yes	Simple	Yes	No	Fast
BDD	Yes	Yes	Simple	Yes	Yes	slow
Correlation Coefficients	Yes	Yes	Moderate	Yes	Yes	Moderate

Table 2.2: *Probabilistic techniques*

All the techniques in Table 2.2 use simplified delay models for the circuit components and require user-specified information about the input behaviour. Therefore their accuracy is limited by the quality of the delay models and the input specification.

using signal probability

The easiest way to propagate signal probabilities throughout every node in a circuit is to work with a gate-level description of the circuit. For example, if $y = AND(x_1, x_2)$, then using basic probability theory we get $P_s(y) = P_s(x_1)P_s(x_2)$, assuming that x_1 and x_2 are spatially independent. Using the same approach, other simple expressions can be derived for other gate types. After calculating the signal probabilities of every node in the circuit, the power can be computed using Equations 2.15 and 2.16 [7].

If a circuit is built from boolean components that are not a part of a predefined gate library, the signal probability can be computed using a *Binary Decision Diagram* (BDD) to represent the boolean functions. Figure 2.7 shows an example where the boolean function $y = x_1x_2 + x_3$ using a BDD. As an example, assume that $x_1 = 1, x_2 = 0, x_3 = 1$, to evaluate y begin at the top node and branch to the right since $x_1 = 1$. Then branch to the left ($x_2 = 0$) and finally to the right ($x_3 = 1$) to arrive at terminal node 1. The resulting value for y is 1 [12, 13].

For the general case, if $y = f(x_1, ..., x_n)$ is a boolean function, and the inputs x_i are independent, then the signal probability of f can be obtained in linear time as follows: let $f_{x_1} = f(1, x_2, ..., x_n)$ and $f_{\overline{x_1}} = f(0, x_2, ...x_n)$ be cofactors of f with respect to x_1, then

$$P(y) = P(x_1)P(f_{x_1}) + P(\overline{x_1})P(f_{\overline{x_1}}) \tag{2.21}$$

where cofactors are defined by the Shannon decomposition of boolean functions [13].

Equation 2.21 shows how the BDD can be used to evaluate $P(y)$.

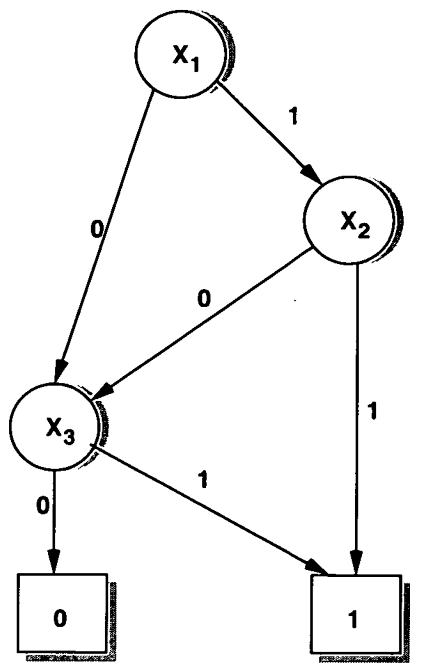


Figure 2.7: *A typical BDD representation*

probabilistic simulation

The typical input signal behaviour of a circuit is provided in a form of waveforms for this approach. Probability waveforms are sequences of values indicating the probability that a signal is high for certain time periods and the probability that it makes a 0 to 1 transition at specific time points. This allows the computation of the average current waveforms drawn by individual gates in the design in a single simulation run. The average current waveforms are then used to compute average power dissipated by each gate, which can in-turn be used to calculate the total average power. An example of the signal probability waveform is given in Figure 2.8 [14].

A program by [15] called CREST uses this approach to estimate power dissipation.

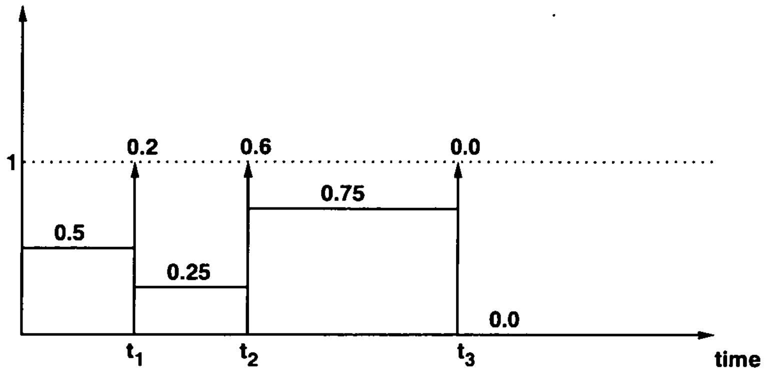


Figure 2.8: *An example signal probability waveform*

Transition Density

A program was presented in [16] which propagates the transition density values from the inputs throughout the circuit. This was called DENSIM. To visualise the propagation of the transition density, recall that if y is a boolean function that depends on x, then the Boolean difference of y with respect to x is

$$\frac{d_y}{d_x} \cong y|_{x=1} \oplus y|_{=0} \tag{2.22}$$

If the inputs (x_i) to the Boolean module are spatially independent, then the transition density of its outputs is given by [16]:

$$D(y) = \sum_{x=1}^{n} P\left(\frac{d_y}{d_{x_i}}\right) D(x_i) \tag{2.23}$$

Using a BDD

A BDD is used to represent the successive Boolean functions at every node in terms of the primary inputs. These functions do not represent the intermediate values that the node takes before reaching a steady state condition. However, the circuit delay information can be used to construct boolean functions for some intermediate values, assuming that the delay of every gate is a specified fixed constant [7].

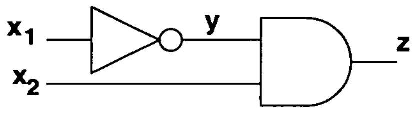


Figure 2.9: *A simplified test case model*

In Figure 2.9, let x_1 and x_2 in two consecutive clock periods be denoted by $x_1(1)$, $x_1(2)$ and $x_2(1)$, $x_2(2)$. Assuming equivalent delays between the inverter and the AND gate, a typical timing diagram can be shown as in Figure 2.10, where it can be seen that node z may make two transitions before settling down [7].

The intermediate and the steady state values can be expressed as follows:

node	expressions
y	$y(1) = \overline{x_1(1)}$
	$y(2) = \overline{x_1(2)}$
z	$z(1) = \overline{x_1(1)}x_2(1)$,
	$z(2) = \overline{x_1(1)}x_2(2)$,
	$z(3) = \overline{x_1(2)}x_2(2)$

Table 2.3: *intermediate and steady state expressions*

A BDD can be built for these functions which makes it possible to accurately compute the intermediate state probabilities.

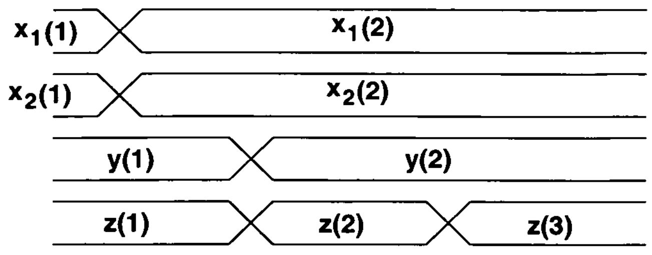


Figure 2.10: *A typical timing diagram for the test case model*

This method can be quite slow since a BDD for the entire circuit will have to be built. In some cases the resulting BDD may be too large [7].

Correlation Coefficients

For this approach, correlation coefficients between steady state signal values are used as approximations to the correlation coefficients between the intermediate signal values [7].

2.4.2 Statistical Techniques

For the statistical technique, a simulation of the circuit is conducted repeatedly while monitoring the power being consumed. A logic or timing simulator can be used for this. The average power will be the final result. The main concern of their method is how the input patterns are selected such that the measured power converges to the true average power. Usually the input vectors are randomly generated based on some method, e.g Monte Carlo. Table 2.4 compares two such methods [7].

Total Power

The method in [17] estimates the total average power of the circuit by applying randomly gen-

Approach	Handle glitch Power	Handle temporal correlation	Input Specification	Individual gate power	Handle spatial correlation	Speed
McPower	Yes	Yes	Simple	No	only internally	Fast
MED	Yes	Yes	Simple	Yes	only internally	Moderate

Table 2.4: *Statistical techniques*

erated input patterns and monitoring the energy dissipated per clock cycle using a simulator. The program developed was named *'McPower'*

The disadvantage of this method is that it does not provide the power consumed by individual gates or a small group of gates.

Individual Gate Power

To deal with the disadvatange exhibited by the above method, [18] proposed a method that not only provides the total power, but also the individual-gate power estimate. They named their program *'MED'*. Despite its improved accuracy, this method suffers from slow speed.

2.5 Summary

This chapter presented the basic of low power design and the underlying factors behind the power estimation tools. Sources of power consumption were presented with the help of the power consumption equation. Using the equation, the degrees of freedom for reducing power consumption were evaluated, giving advantages and disadvantages of each technique. The focus of this research is based on reducing the switching activity of a circuit. A summary of techniques for reducing power consumption was presented in table 2.1. Some power estimation techniques were described briefly, this gives the background of the techniques behind the power estimation tools used in the research.

Chapter 3
Algorithms and Architectures for Image Transforms

3.1 Introduction

This chapter introduces the basic concepts of Image Processing and Image Compression. The image compression covered is the one that results due to image transformation. Some image transforms are presented which compete with the DCT.

An image processing system, in general, consists of a source of image data, a processing element and a destination for the processed results. Figure 3.1 shows a typical image processing system.

The source of image data can be any of the following: a camera, a scanner, a mathematical equation, statistical data etc. That is, anything able to generate or acquire data that has a two-dimensional structure is considered to be a valid source of image data. Furthermore, the data may change as a function of time.

The processing element is usually a microprocessor. The microprocessor may be implemented in several ways. For example, the brain can be considered as some kind of a microprocessor that is able to perform image processing. Another type of microprocessor that can handle image processing is the digital computer.

For the purpose of image processing, digitised video can be treated as a sequence of frames (images) with each frame represented as an array of picture elements (pixels), See Figure 3.2. For colour video, a pixel is represented by three primary components - Red(R), Green(G), and Blue(B). For effective coding, the three colour components are converted to another coordinate system, called YUV, where Y denotes the luminance (brightness) and U and V, called the chrominance, denote the strength and vividness of the colour [19, 20]. This conversion is described below:

$$\begin{aligned} Y &= 0.3R + 0.6G + 0.1B \\ U &= B - Y \\ V &= R - Y \end{aligned} \tag{3.1}$$

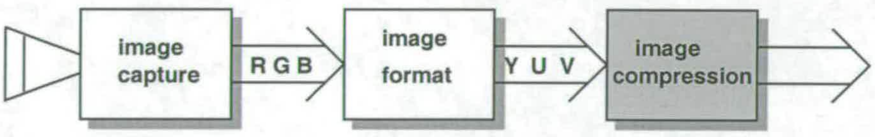


Figure 3.1: *from capture to compression*

A 2D continuous image $a(x, y)$ is divided into N rows and M columns, see Figure 3.2. The intersection of a row and a column is normally termed a pixel. The value assigned to the integer coordinates $[m, n]$ with $\{m = 0, 1, 2, \ldots, M-1\}$ and $\{n = 0, 1, 2, \ldots, N-1\}$ is $a[m, n]$.

The image shown in Figure 3.2 has been divided into $N = 19$ rows and $M = 26$ columns. The value assigned to every pixel is the average brightness in the pixel rounded to the nearest integer value. The process of representing the amplitude of the 2D signal at a given coordinate

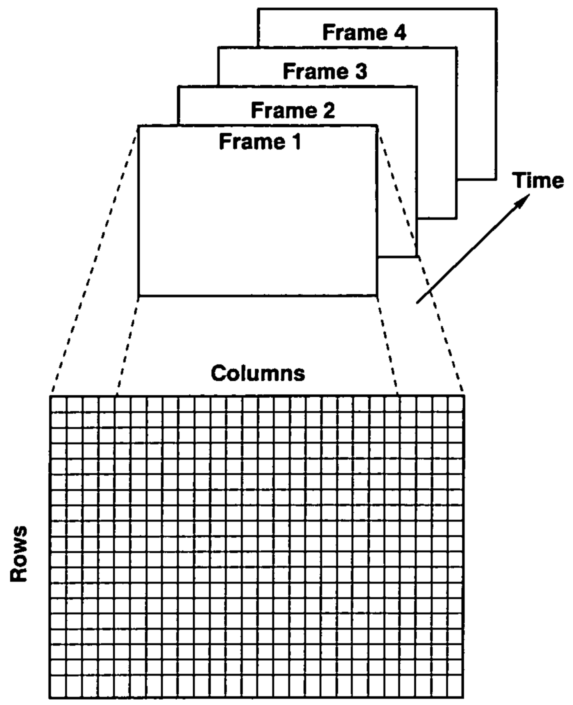


Figure 3.2: *Video sequence example*

as an integer value with L different gray levels is usually referred to as amplitude quantisation or simply quantisation.

There are standard values for the various parameters encountered in digital image processing. These values can be dictated by video standards, by algorithmic requirements, or by the desire to keep digital circuitry simple. Table 3.1 gives some of the commonly encountered values [21].

Parameter	Symbol	Typical Value
Rows	N	256, 512, 525, 625, 1024, 1035
Columns	M	256, 512, 768, 1024, 1320
Gray Levels	L	2, 64, 256, 1024, 4096, 16384

Table 3.1: *Common Values of digital image parameters*

The number of distinct gray levels is usually a power of 2, that is, $L = 2^n$ where n is the number of bits in the binary representation of the brightness levels. When $n > 1$, the image is referred to as a gray-level image, whereas when $n = 1$, the image is a binary or bi-level image . In a bi-level image there are just two gray levels which can be referred to, for example, as "black" and "white" or "0" and "1".

3.2 Image Compression Fundamentals

Image compression operations reduce the data content of a digital image and represent the image in a more compact form, usually before storage or transmission. Grey scale, colour or bi-level images can be compressed and different types of compression may be used for different applications such as in medical imaging, the internet, finger-printing/security, seismology and astronomy.

Digital images can be compressed by eliminating some redundant information. There are three basic types of redundancy that can be exploited by image compression:

1. Spatial Redundancy

   - in natural images, the values of neighbouring pixels are strongly correlated

2. Spectral Redundancy

   - some images are composed of more than one spectral band, hence the spectral values for the same pixel location are sometimes correlated

3. Temporal Redundancy

- adjacent frames in video sequences often show very little change

Transform coding, which uses some reversible linear transform to decorrelate the image data, removes both spatial and spectral redundancies. Temporal redundancy is handled by techniques that only encode the differences between adjacent frames in an image sequence. Motion prediction is an example of such techniques.

There are two types of image compression, 1)lossy and 2)lossless.

1. **Lossy compression**

   This type of compression results in the decompressed image being similar but not the same as the original image. This is because some of the original data has been completely discarded and/or changed. Because of its 'lossy' nature, this technique offers high compression ratios in the order of 12:1 and beyond, depending on how much data one is willing to loose.

2. **Lossless compression**

   Lossless compression on the other hand retains the exact data of the original image bit for bit. Lossless compression ratios are much lower, achieving rates of approximately 3:1.

Image compression is normally a two-way process which involves both compression and decompression. This process may not be symmetrical, that is, the time taken and the computing power for one process may differ from the other given the type of compression algorithm used.

3.2.1 Image Transforms

If a purely sinusoidal signal has to be transmitted over some media, the signal can be sampled and each data point be transmitted sequentially. The number of points depend on how accurate the reconstructed signal should be, more points result in a better reconstructed signal. To construct a deterministic sinusoid, *magnitude, phase, frequency, starting time* and the fact that it is *sinusoid* are required. Therefore only five pieces of information are required to reconstruct the exact sinusoid. From an information theoretic point of view, a sampled sinusoid is highly correlated whereas the five pieces mentioned above are not correlated. Transformation is an attempt to take N sampled points in the transmission and turn them into a few uncorrelated information pieces [22].

Transforms are used widely in image processing for functions such as image filtering and image data compression. Only a few of the commonly used compression transforms are presented in this section.

Basics

Let an image d be represented as an MxN matrix of integer numbers

$$\mathbf{d} = \begin{bmatrix} d(0,0) & d(0,1) & \dots & d(0,N-1) \\ & & \vdots & \\ d(M-1,0) & d(M-1,1) & \dots & d(M-1,N-1) \end{bmatrix} \tag{3.2}$$

A general transform for image d is

$$D = PdQ \tag{3.3}$$

where P and Q are transformation matrices on the rows and columns respectively for a 2-D transform.

The general transform equation 3.3 can be re written as

$$D(u,v) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} P(u,m)d(m,n)Q(n,v) \tag{3.4}$$

where $u = 0, 1, \ldots, M-1$; and $v = 0, 1, \ldots, N-1$

If P and Q are non-singular(non-zero determinants), then the inverse of matrix of Equation 3.3 exists.

$$d = P^{-1}DQ^{-1} \tag{3.5}$$

If both P and Q are symmetric ($P = P^t$ and $Q = Q^t$), and orthogonal($P^tP = 1$ and $Q^tQ = 1$) then

$$\begin{aligned} D &= PdQ \\ d &= PDQ \end{aligned} \tag{3.6}$$

and the transform is termed as an orthogonal transform [20].

Discrete Fourier Transform

The discrete Fourier transform is analogous to the continuous Fourier transform and may be efficiently computed using the Fast Fourier transform algorithm. The properties of linearity, shift of position, modulation, convolution, multiplication and correlation are similar to the continuous one. The difference between them is the discrete periodic nature of the image and its transform [23, 24].

Let Φ_{jj} be a transform matrix of size jxj

$$\Phi_{jj}(k,l) = \frac{1}{j}e^{(-i\frac{2\pi}{j}kl)} \tag{3.7}$$

where $k, l = 0, 1, \ldots, j-1$

The discrete Fourier transform can be defined according to the following equation

$$F = \Phi_{MM} f \Phi_{NN} \tag{3.8}$$

$$F(u,v) = \frac{1}{MN}\sum_{m=0}^{M-1}\sum_{n=0}^{N-1} f(m,n)e\left(-2\pi i(\frac{mu}{M} + \frac{nv}{N})\right) \tag{3.9}$$

where $u = 0, 1, \dots, M-1$ and $v = 0, 1, \dots, N-1$

The inverse transform matrix is given by

$$\Phi_{jj}^{-1}(k,l) = e^{(\frac{2\pi i}{j}kl)} \tag{3.10}$$

and the inverse Fourier transform by

$$f(m,n) = \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} F(u,v) e\left(2\pi i(\frac{mu}{M} + \frac{nv}{N})\right) \tag{3.11}$$

where $m = 0, 1, \dots, M-1$ and $n = 0, 1, \dots, N-1$

Therefore, the kernel of the discrete Fourier transform is given by

$$e\left(2\pi i(\frac{mu}{M} + \frac{nv}{N})\right) \tag{3.12}$$

When considering implementation of the discrete Fourier transform, equation 3.9 can be modified to

$$F(u,v) = \frac{1}{M} \sum_{m=0}^{M-1} \left[\frac{1}{N} \sum_{n=0}^{N-1} e^{(\frac{-2\pi inv}{N})} f(m,n) \right] e^{(\frac{-2\pi imu}{M})} \tag{3.13}$$

The term in the square brackets, which is actually a 1D Fourier transform of the mth line, can be computed using the standard Fast Fourier Transform procedures. Each line is substituted with its Fourier transform, and the 1D discrete Fourier transform of each column is computed

[20].

Although the DFT offers a good energy compaction, it suffers from increased computational complexity. Both real and imaginary components of the transform have to be computed as it can be seen in Equation 3.13.

Hadamard Transform

The forward Hadamard kernel is defined as

$$g(x,u) = \frac{1}{N}(-1)^{\sum_{i=0}^{n-1} b_i(x)b_i(u)} \tag{3.14}$$

where the summation in the exponent is performed in modulo 2 arithmetic. Therefore a one dimensional forward Hadamard transform is described by

$$H(u) = \frac{1}{N}\sum_{x=0}^{N-1} f(x)(-1)^{\sum_{i=0}^{n-1} b_i(x)b_i(u)} \tag{3.15}$$

where $f(x)$ is the image samples, $N = 2^u$ and $u = 0, 1, 2, \ldots, N-1$

A Hadamard matrix H is an nxn matrix with all entries +1 or -1(see Equation 3.16), such that all rows are orthogonal and all columns are orthogonal [20].

The usual development starts with a defined 2x2 Hadamard matrix H_{22}, Equation 3.16. Each step consists of multiplying each element in H_{22} by the previous matrix [20].

$$\mathbf{H_{22}} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{3.16}$$

$$\mathbf{H_{2j2j}} = \begin{bmatrix} H_{jj} & H_{jj} \\ H_{jj} & -H_{jj} \end{bmatrix} \tag{3.17}$$

$$H_{jj}^{-1} = \frac{1}{j} H_{jj} \tag{3.18}$$

$$\begin{aligned} F &= H_{MM} f H_{NN} \\ f &= \frac{1}{MN} H_{MM} F H_{NN} \end{aligned} \tag{3.19}$$

The Hadamard transform has no multiplications. This might seem like an advantage but it turns out that it is very hard to analyse it.

Wavelet Transform

Wavelets are mathematical functions that divide data into different frequency components. Each component is then studied with a resolution matched to its scale. Wavelets are suited to modelling phenomena whose signals are not continuous. Wavelet compression algorithms have achieved ratios of around 300:1 for still images [25] .

The compression technique used in wavelets is using low-pass and high-pass filters to separate an image into images with low or high spatial frequencies respectively. Low frequency images being those with gradual brightness change. This is the case for images like flat or rounded background areas. Such images appear soft and blurry. As for the high frequency band, the images are sharp and crisp edged. To reconstruct the original image, the frequency bands imaged are added together. This results in a near perfect image if the processing is perfect [19, 26].

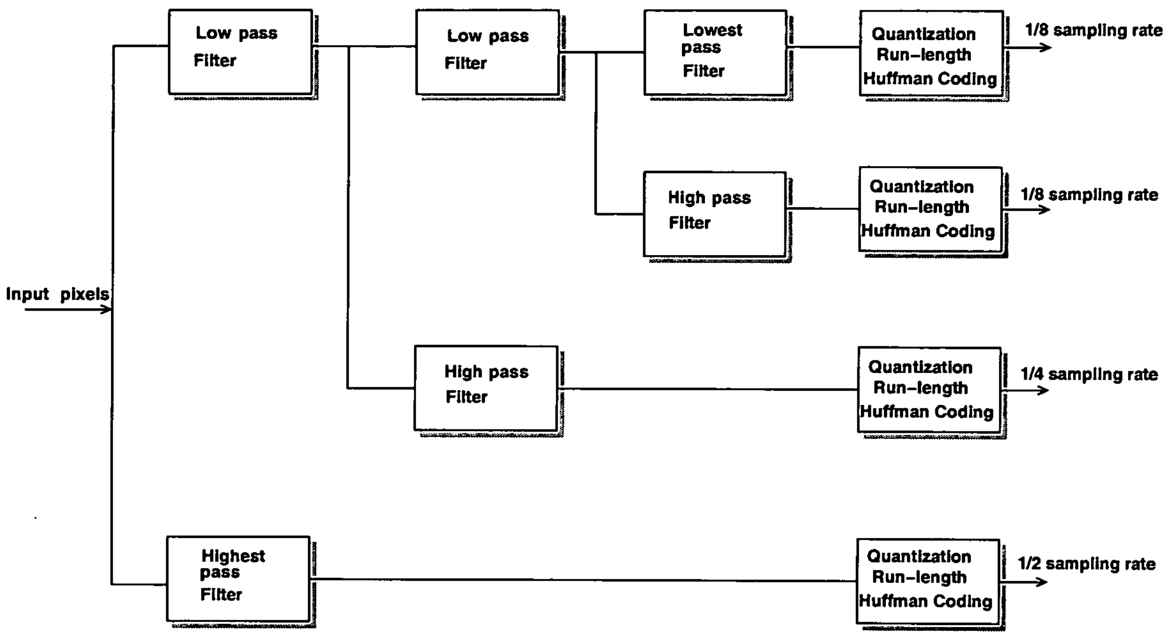


Figure 3.3: *Wavelet compression*

In figure 3.3, a pixel data stream from an input image is divided into several sub-bands by a tree of low and high pass filters. Each filter allows a specific band of frequencies to pass. There filters can be either digital or analogue.

Wavelet compression is a lossy process. The image quality is always compromised to some extent. Higher compression rates results in high image distortion. This distortion is different from the blocking effects arising from the DCT. Wavelet application areas include signal and image compression [27], communications [25].

Despite all the advantages exhibited by the wavelet transform, it has an inherent weakness. This segmentation of an image into different frequency ranges requires that the resulting images be stored in the interim before going through the quantisation stage. Therefore starting of with a single image and ending up with several images costs in terms of storage space.

Karhunen-Loeve Transform

The Karhunen-Loeve Transform (KLT) is based on statistical properties of an image. Its main applications are in image compression and rotation. A sample image $f_i(x, y)$ can be expressed in the form of an N^2-dimensional vector x_i

$$\mathbf{x}_i = \begin{bmatrix} x_{i1} \\ x_{i2} \\ \vdots \\ x_{ij} \\ \vdots \\ x_{iN^2} \end{bmatrix} \tag{3.20}$$

where x_{ij} denotes the jth component vector $\mathbf{x}_i$.

The covariance matrix of the x vectors is defined as

$$\mathbf{C_x} = E\{(\mathbf{x} - \mathbf{m_x})(\mathbf{x} - \mathbf{m_x})^{\mathbf{T}}\} \tag{3.21}$$

where $\mathbf{m_x} = E\{\mathbf{x}\}$ is the mean vector and E is the expected value. Equation 3.21 can be approximated from image sample using the relations

$$\mathbf{m_x} \cong \frac{1}{M}\sum_{i=1}^{M}\mathbf{x}_i \tag{3.22}$$

and

$$\mathbf{C_x} \cong \frac{1}{M}\sum_{i=1}^{M}(\mathbf{x_i}-\mathbf{m_x})(\mathbf{x_i}-\mathbf{m_x})^T \tag{3.23}$$

or

$$\mathbf{C_x} \cong \frac{1}{M}\left[\sum_{i=1}^{M}\mathbf{x_i}\mathbf{x_i^T}\right] - \mathbf{m_x}\mathbf{m_x}^T \tag{3.24}$$

The mean vector is N^2-Dimensional and $\mathbf{C_x}$ is an $N^2 x N^2$ matrix.

Let $\mathbf{e}_i$ and λ_i, $i = 1, 2, \ldots, N^2$, be the eigenvectors and corresponding eigenvalues of $\mathbf{C_x}$. The eigenvalues are arranged in decreasing order ($\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_{N^2}$) for convenience. A transformation matrix, $\mathbf{A}$, whose rows are eigenvectors of $\mathbf{C_x}$ is

$$\mathbf{A} = \begin{bmatrix} e_{11} & e_{12} & \ldots & e_{1N^2} \\ e_{21} & e_{22} & \ldots & e_{2N^2} \\ \vdots & & & \\ e_{N^21} & e_{N^22} & \ldots & e_{N^2N^2} \end{bmatrix} \tag{3.25}$$

where e_{ij} is the jth component of the ith eigenvector.

The discrete Karhunen-Loeve transform is simply a multiplication of the centralised image vector $(\mathbf{x} - \mathbf{m_x})$, by $\mathbf{A}$ to obtain a new image vector $\mathbf{y}$:

$$\mathbf{y} = \mathbf{A}(\mathbf{x} - \mathbf{m_x}) \tag{3.26}$$

The KLT offers optimal energy compaction in the Mean Square error sense. That is $\epsilon(k) = E[(x - \hat{x})^T(x - \hat{x})]$ is a minimum, where $\hat{x}$ is a representation of the truncated x in k terms. E is the mathematical expectation operator. It also projects the data onto a basis that results in complete decorrelation. Therefore, the KLT can reduce dimensionality if the data is of high dimensionality [20, 28].

As can be seen from the above presentation, the transform coefficients of the KLT vary from image to image, hence they have to be computed on the fly. This can be very computationally intensive for realtime and low power applications. Therefore, it is better to use the next best efficient image transform.

Discrete Cosine Transform

The Discrete cosine transform and its inverse (IDCT) are the transforms for practical image processing. Its energy compaction abilities are high. Another advantage over other transforms is the existence of fast implementation algorithms. Because the DCT is the main subject of this project, it will be discussed in detail in the next section. It is not the purpose of this thesis to examine the IDCT, however the techniques proposed can be used to process the IDCT.

3.3 The Discrete Cosine Transform

Ever since it was discovered in 1974 by [29], the DCT has attracted attention from engineering, scientific and research communities. This is indeed the least surprising because of its energy packing capabilities which approach the statistically optimum transform, the KLT. A number of fast DCT algorithms have been developed which also contributed to its hype [22, 30, 31].

The following equation is a mathematical definition of an NxN DCT.

$$C(k,m) = \frac{2G_kG_m}{N}\sum_{i=0}^{N-1}\sum_{j=0}^{N-1} d(i,j)cos(\frac{(2i+1)k\pi}{2N})cos(\frac{(2j+1)m\pi}{2N}) \qquad (3.27)$$

where $G_k = G_m = 1$ and $G_0 = 1/\sqrt{2}$

in a matrix form, Equation 3.27 can be written as:

$$[C] = [E][D][E]^t \qquad (3.28)$$

U. BL. UNIV. EDINBURGH

where

$[E]$ is the cosine matrix and $[D]$ is the pixel matrix

An interesting quality of the DCT is that it is separable and orthogonal. The orthogonality property implies that the energy of a signal is preserved under the transformation. The DCT's separability principle implies that a multidimensional DCT can be implemented by a series of one dimensional (1D) transforms. The advantage of this property is that fast algorithms developed for 1D DCT can be directly extended to multidimensional transforms (Figure 3.4). It should be observed that other forms of transformation do posses this separability quality. Some examples are DFT, WHT, Haar etc [22].

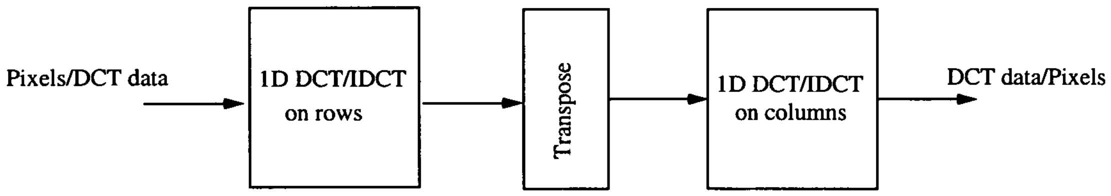


Figure 3.4: *Implementation of 2D DCT by 1D DCTs*

3.3.1 DCT Algorithms and Architectures

The development of efficient algorithms began soon after it inception. It was natural for the initial attempts to focus on the computation of the DCT by using the FFT algorithms. Although it was not developed as a discretised version of the FFT, the DCT's relation to the DFT were exploited in the initial developments of its computational algorithms [22].

There are two main approaches for implementing the DCT. Firstly, there is the flowgraph based architecture, which has an advantage of minimising the total number of arithmetic operations and also is the basis of the fast algorithms [32]. Secondly, there is the scalar product based architectures, these have gained favour with designers for their regularity in nature and can exploit the use of distributed arithmetic [33]. Several architectures are discussed in literature, see [34–36]. This section introduces only fundamental architectures.

3.3.1.1 The Chen Algorithm

Consider the matrix E

$$\mathbf{E} = \begin{bmatrix} d & d & d & d & d & d & d & d \\ a & c & e & g & -g & -e & -c & -a \\ b & f & -f & -b & -b & -f & f & b \\ c & -g & -a & -e & e & a & g & -c \\ d & -d & -d & d & d & -d & -d & d \\ e & -a & g & c & -c & -g & a & -e \\ f & -b & b & -f & -f & b & -b & f \\ g & -e & c & -a & a & -c & e & -g \end{bmatrix} \tag{3.29}$$

where

$$a = cos\frac{\pi}{16}, b = cos\frac{\pi}{8}, c = cos\frac{3\pi}{16}, d = cos\frac{\pi}{4}, e = cos\frac{3\pi}{16}, f = cos\frac{\pi}{8}, g = cos\frac{\pi}{16} \tag{3.30}$$

The symmetry of matrix E can be exploited to represent the 1-D DCT as follows

$$\begin{bmatrix} y_0 \\ y_2 \\ y_4 \\ y_6 \end{bmatrix} = \begin{bmatrix} d & d & d & d \\ b & f & -f & -b \\ d & -d & -d & d \\ f & -b & b & -f \end{bmatrix} \begin{bmatrix} x_0 + x_7 \\ x_1 + x_6 \\ x_2 + x_5 \\ x_3 + x_4 \end{bmatrix}$$

$$\begin{bmatrix} y_1 \\ y_3 \\ y_5 \\ y_7 \end{bmatrix} = \begin{bmatrix} a & c & e & g \\ c & -g & -a & -e \\ e & -a & g & c \\ g & -e & c & -a \end{bmatrix} \begin{bmatrix} x_0 - x_7 \\ x_1 - x_6 \\ x_2 - x_5 \\ x_3 - x_4 \end{bmatrix} \tag{3.31}$$

The exploitation in 3.31 results in the (NXN) multiplication matrix being replaced two $(N/2)X(N/2)$ matrices which can be computed in parallel together with the sums and differences on the right hand side of 3.31.

Implementations in [37],[38], [39], [40] and [41] are all derivatives of this algorithm. To further reduce complexity, [37] observed that the first matrix processing in Equation 3.31 involves multiplication with only three cosine coefficients (b, d and f). Therefore, the multiplications can be reduced from 32 to 28 in total.

The Algorithm presented in [42] is derived from 3.31, however it only requires 16 multiplications with 2 Additions on the critical data path. Figure 3.5 depicts the data flow of this algorithm. The arrow ($\longrightarrow$) in the figure represents subtraction. An example architecture that can be used to implement this algorithm is depicted in Figure 3.6.

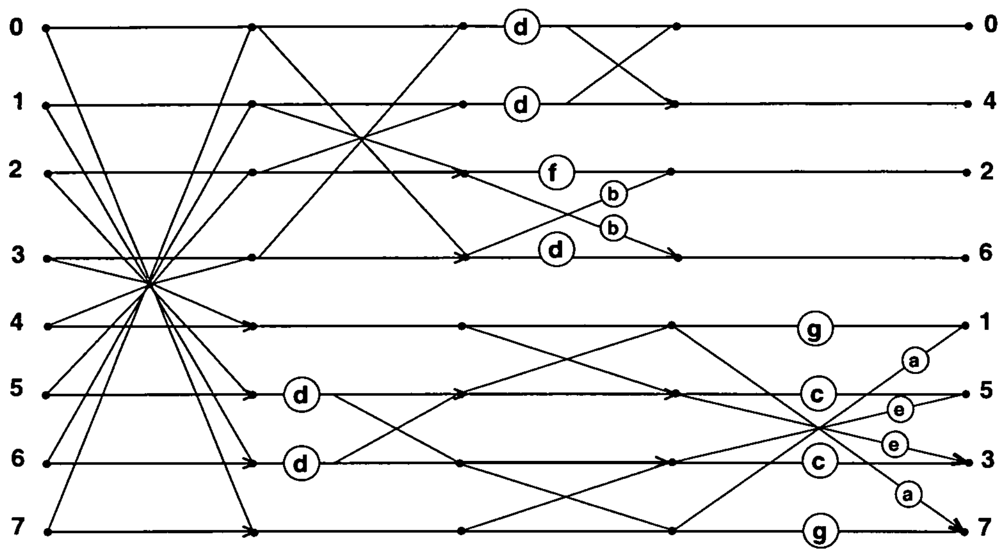


Figure 3.5: *Data flow graph for the Chen Algorithm*

3.3.1.2 The Arai Algorithm

Arai et al in [43] developed an algorithm that requires only 5 multiplications. This scheme is shown in the flowgraph depicted in 3.7. This is only possible because the outputs are scaled. To get the true values of the DCT coefficients, the output requires a further multiplication. For some systems, this further multiplication can be incorporated in subsequent stages. For example, the perceptual weights in the video encoding algorithm. This processing technique is termed as scaled-DCT.

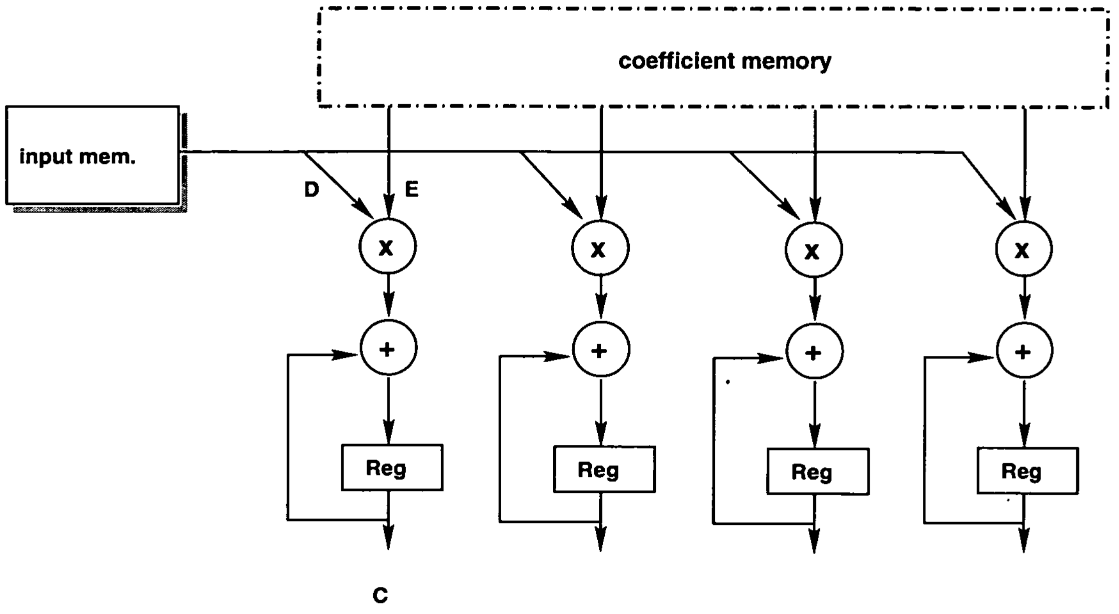


Figure 3.6: *DCT architecture based on four MAC processing units*

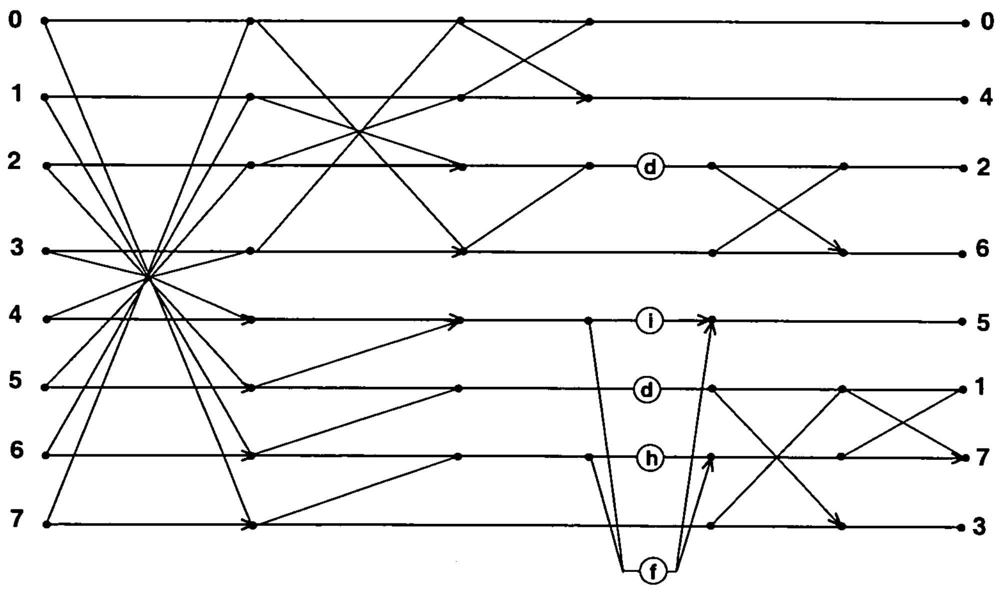


Figure 3.7: *Data flow graph for the Arai Algorithm*

3.3.1.3 The Chiu Algorithm

A different approach to the DCT implementation was proposed by Chiu et al in [44, 45]. They propose a direct implementation whereby the DCT is computed recursively. The values of the DCT are updated with each new sample. Each DCT utilises the results from a previous DCT, ie the next DCT is obtained by adding the difference between it and the previous DCT. This method requires the Discrete Sine Transform (DST), therefore a computation of both the DCT and the DST is necessary. Deriving the recursive expression from 3.32, we get 3.33

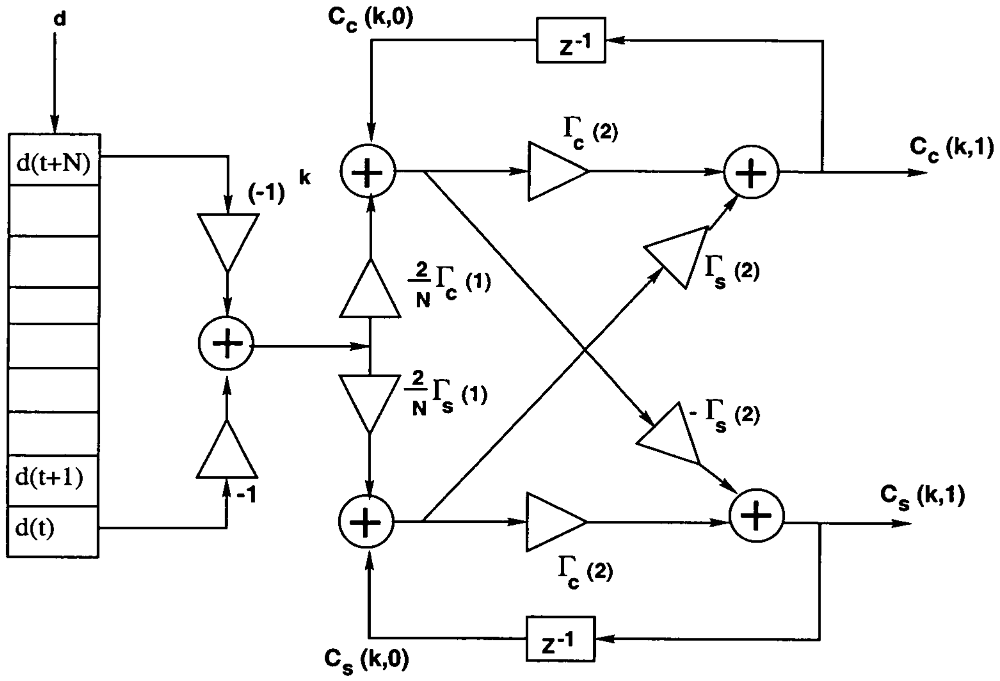


Figure 3.8: *Lattice structure*

$$C_c(k,t) = \frac{2G(k)}{N} \sum_{n=t}^{t+N-1} d(n) cos(\frac{(2(n-t)+1)k\pi}{2N}) \tag{3.32}$$

where $k = 0, ..., N-1$

$$\begin{aligned} C_c(k,t+1) &= (C_c(k,t) + \frac{2}{N}\left[-d(t) + (-1)^k d(t+N)\right] cos(\frac{k\pi}{2N}))cos(\frac{k\pi}{N}) \\ &+ (C_s(k,t) + \frac{2}{N}\left[-d(t) + (-1)^k d(t+N)\right] sin(\frac{k\pi}{2N}))sin(\frac{k\pi}{N}) \end{aligned} \quad (3.33)$$

where $k \neq 0$; $k = 0$ is a special case. $C_s(k,t+1)$ is generated in the same manner as Equation 3.33. The difference is that the first cosine and sine terms in Equation 3.33 are swapped. $C_c(k,t+1)$ and $C_s(k,t+1)$ are obtained from $C_c(k,t)$ and $C_s(k,t)$ by subtracting the effect of $d(t)$ and adding the effect of $d(t+N)$. This process is termed as time recursive DCT.

Chiu et al proposed a fully pipelined architecture to compute the 2-D DCT from a frame-recursive point of view. Two real-time lattice structures for successive frame and block 2-D have been developed. These structures are fully pipelined with throughput rate N clock cycles for an NxN successive input data frame. The resulting 2-D DCT architectures are modular, regular, and locally connected and require only two 1-D DCT blocks that are extended directly from the 1-D DCT structure without transposition [44].

The 2-D DCT can be generated by using the lattice structure as in Figure 3.8. Depending on the coefficients stored, the appropriate transform coefficient, Y_k is computed. Every clock unit, a new data input enters and gets processed. At the end of N cycles, the computations for that set of N data elements are completed. By employing N such structures, all the transform coefficients are computed in parallel and made available at the end of N cycles. This parallel computation of N coefficients represents the 1-D DCT.

Each kernel requires six multipliers for the computation of the forward transform alone. Clearly, there are too many multipliers for the efficient VLSI implementation. The advantage of this structure is its better numerical properties, being a normal form implementation as opposed to the IIR structure.

3.3.1.4 Distributed Arithmetic (DA) Implementation

Distributed arithmetic differs from conventional arithmetic in the order in which operations are performed.

Consider a one dimensional N-point DCT defined as follows

$$C = \sum_{i=0}^{N-1} e_i d_i \tag{3.34}$$

where e_k's are W-bit constants and d_i's are coded in B bits using two's compliment representation, i.e

$$d_i = -d_{i,0} + \sum_{j=1}^{B-1} d_{i,j} 2^{-j} \tag{3.35}$$

this leads to :

$$\begin{aligned} C &= \sum_{i=0}^{N-1} e_i(-d_{i,0} + \sum_{j=1}^{B-1} d_{i,j}2^{-j}) \\ &= -\sum_{i=0}^{N-1} e_i d_{i,0} + \sum_{j=1}^{B-1}(\sum_{i=0}^{N-1} e_i d_{i,j}2^{-j}) \\ &= \sum_{j=0}^{B-1} E_j 2^{-j} \end{aligned} \tag{3.36}$$

where

$$E_{j\neq 0} = \sum_{i=0}^{N-1} e_i d_{i,j} \tag{3.37}$$

and

$$E_0 = -\sum_{i=0}^{N-1} e_i d_{i,0} \tag{3.38}$$

The change of summing the order in i and j characterises the DA scheme whereby the initial multiplications are *distributed* to another computation pattern. Since the term E_j has only 2^N possible values(that depend on the $d_{i,j}$ values), it is possible to store these 2^N values in a ROM. An input set of N bits $\{d_{0,j}, d_{1,j}, ..., d_{N-1,j}\}$ is used as an address, allowing the retrieval of E_j values. These intermediate results are accumulated in B clock periods, for producing one C value. A typical architecture for DA computation is depicted in Figure 3.9. The multipliers are replaced by a ROM followed by an accumulator [46, 47].

The DA takes advantage of ROM's in which the partial products are stored. This ROM's need to be reset every internal clock. This is prone to increased power consumption since the resets are implemented using a P-channel pull-up transistor. A direct path between VDD and ground exists while the pull-ups are active [48].

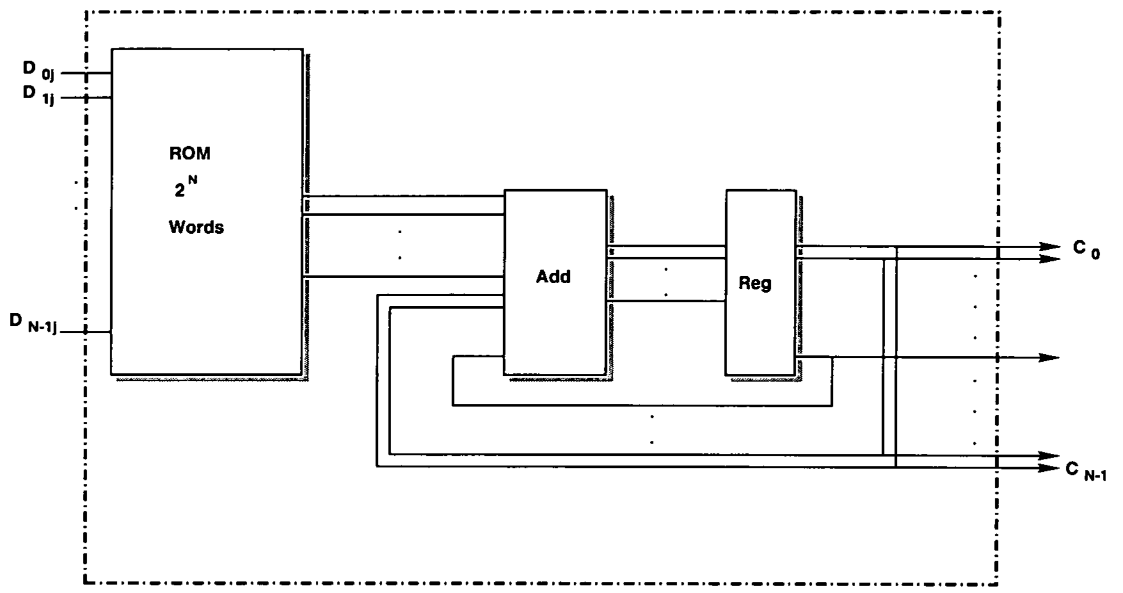


Figure 3.9: *Architecture of N input scalar product using DA*

3.3.1.5 Bit-Serial Implementation

In cases where cost is an issue, where parallel implementations are too expensive in terms of integration, bit serial implementation becomes a viable option. These are simply a direct mapping of a data-flow graph. Some bit-serial DCT implementations are presented in [49], [50], [51] and [52]. Balsara et al in [53] presented an overview of bit serial multipliers. A detailed comparison of bit-serial and bit parallel implementations is presented in [54]. Another advantage of bit-serial is that wiring overheads are almost negligible because only

1-2 wires need to be routed per interconnection [55]. However, the availability of more routing layers, the impact of global interconnects on bit-parallel implementations may not be so detrimental. In the context of power, bit-parallel implementations have the advantage of being able to exploit data correlation for power reduction [56].

	Multiplications additions, and ROM space	Number of Multiplications	Number of Additions	Number of ROM words	Number of RAM words for Transposition
Per NxN Block	Direct Computation	N^4	N^4	-	-
	With Separability	$2N^3$	$2N^3$	-	N^2
	Fast Algorithm	$N^2 \log_2 N$	$N^2 \log_2 N$	-	N^2
	Pure DA	-	$32N$	$2N2^N$	N^2
	Mixed DA/FG	-	$34N$	$N2^{\frac{L}{2}}$	N^2
Per Pixel	Direct Computation	N^2	N^2	-	-
	With Separability	$2N$	$2N$	-	N^2
	Fast Algorithm	$\log_2 N$	$2\log_2 N$	-	N^2
	Pure DA	-	$\frac{32}{N}$	$2N2^2$	N^2
	Mixed DA/FG	$2N^3$	$\frac{34}{N}$	$N2^{\frac{L}{2}}$	N^2

Table 3.2: *Computation and Storage requirements for some DCT Algorithms*

3.4 Summary

This chapter presented the basics of image compression. It also highlighted relevant important features of some well known image transforms and how they relate or compare with the DCT. It is still fact that in terms of energy compaction, the DCT performs close to the optimal KLT.

There are several algorithms developed for the DCT ever since its invention. This goes to show its continuing popularity amongst the image compression research and development. Some algorithms are more suited for VLSI implementation than others, particularly the implementation using the separability principle of the DCT. This is so because of the regular structure that results from implementation. Table 3.2 presents a summary of different implementation, which include computational complexity and ROM/RAM requirements.

Chapter 4
Low Power DCT Research

4.1 Introduction

Some of the work done by other researchers in order to address the problem of low power DCT implementation is briefly summarised in this chapter. In addition it will also highlight the need for DCTs and in particular, the low power DCT implementation.

Most research work considering low power implementation of the DCT have targeted reducing the computational complexity of the design or modifying it for operation under a lower supply voltage [57, 58]. Both these techniques have a limited effect on power reduction. Another major contribution to power consumption is due to the effective switched capacitance [56, 59]. Only a few researchers have targeted reducing power of a DCT implementation through a reduction in the amount of switched capacitance [60–62].

4.2 Work so far

Switched capacitance reduction has been achieved through techniques such as the adaptive bitwidth reduction [61–63] and lookup table partitioning [60]. Xanthopoulos et al. [62] demonstrated that bitwidth of arithmetic operations can be minimised dynamically in the presence of data spatial correlation. This technique utilised the fact that image pixels are locally highly correlated and they show a certain number of common most significant bits. These MSB's are only relevant for the DC component and can be rejected for all high frequency coefficients. Hence the bitwidth is reduced, which in turn implies reduction of switching activity.

For the lookup table partioning technique, Cho et al. [60] proposed a data driven variable length decoder which exploits the signal statistics of variable length codes to reduce power. The idea here is to minimise the average codeword length by exploiting the statistics of the data. That is, shorter codewords are assigned to recurrent data while longer codewords are assigned to sparsely occurring data. Therefore a minimum average codeword is experienced, implying a reduction of switched capacitance.

Another circuit has been proposed in [63] which reduces signal transitions by adaptively adjusting the number of the MSB's to varying content of the input data. The scheme compares the input pixels and dynamically shuts the MSB's values that have not changed.

Chen et al in [64, 65] presented a low power architecture for video coding systems using a combination of low power techniques such as multirate, look-ahead and pipelining. Multirate in this context is defined as dividing the data stream into odd and even sequences and then running the resulting modules at half the speed of the input data.

Targeting an asynchronous multiplier for the DCT, Kim et al [66] take advantage of the typically large fraction of zero and small valued data in the DCT application. Their design skips multiplication by zero and dynamically deactivates bit slices that contain only sign extension bits.

The techniques here include removal of circuit blocks that computes the DCT coefficients which will be quantised to zeroes, reordering of operations in constant-multipliers to reduce transition probability, and re-designing cells for low-voltage operation. Power consumption for an 8x8 2D DCT is estimated at 16.95mW [67].

The use of pipelining and parallelism for reducing power consumption is considered in [68] for fast discrete cosine transforms. ROM-Based algorithms are also presented. In conclusion, multiplier based implementations consumed less power than ROM based implementations. But the ROM based Implementation was faster than the multiplier Implementation.

Chen et. al. in [58] presented a low power DCT chip. Their technique for power reduction was based on reducing the complexity of the DCT by using the direct implementation of the 2-D DCT. They also used a parallel distributed arithmetic approach with reduced power supply voltage [58, 69].

In [70], Masselos et al presented an approach for low power realisation of DSP algorithms that are based on inner product computation. Their proposed methodology is based on an architectural transformation that reorders the sequence of evaluation of the partial products forming the inner products.

Although the methodology summarised above appears to be comparable to the schemes proposed in this thesis, there are several key differences. The first difference worthy of noting is that the methodology proposed by [70] is not generic, only the minimum Hamming distance can be used with the methodology. This implies that no other ordering techniques can be used even if it offers a better solution in terms of reducing switching activity. In [70], it is unclear how the data is being dealt with. This comes into consideration since when a matrix multiplication is to be processed, the row of the cosine matrix has to be multiplied with a column of the data matrix. If the location of the entry in the row of the cosine matrix is changed, appropriate measures have to be taken in the respective column entry so that the correct entries are used. And finally, [70] proposes an architecture whereby data is transfered from the data memory into a set of foreground registers. This foreground registers are configured in such a way that they form a queue. That is when the second entry is being read from the data memory, it pushes the first entry into the second register. This implementation suggests that every time a new entry is being read from the data memory, all the registers are being written to since data is being shifted into them. From our experience, this technique is very costly in terms of switching activity.

This thesis presents a number of low power approaches for implementing the DCT. The low power approach used is based on reducing the switching activity of circuit. The techniques are generic and can be extended to include other applications where matrix multiplication is required. Some DCT architectures which suit the schemes are proposed as proof of the concept, and some power dissipation measures are given for the data path components.

4.3 A Case for Low-Power DCT

Until recently, dedicated signal processing hardware was a necessity for the DCT computation in real time video applications. MPEG2 compression for a 720 x480 frame size at 30 frames per second(fps) 4:2:0 requires approximately 132 mega operations per second worst case assuming that a fast computation algorithm is employed. Microprocessors (300MHz) with instruction sets enhanced with multimedia extensions (ie Intel MMX) are capable of performing multiple instructions per cycle by partitioning a single 64-bit integer datapath into a two 32-bit, four 16-bit or eight 8-bit independent integer datapaths operating in parallel. Such instruction sets are ideal for the implementation of signal processing tasks that involve multiple independent low precision integer operations. A 300 MHz Intel Pentium II can process 1.2 giga operations per second on 16-bit media data types. Therefore, it can be observed that the DCT approximately occupies 11% of the processing resources available in commodity microprocessors today in a real time video application. The other 89% of the machine resources may be allocated to other video processing tasks such as motion compensation and entropy decoding or graphics computations for updating the display[61].

Some more figures reflecting the amount of data involved in video data are listed below.

- Five minutes of video

  - 512x512 pixel colour image at 25 frames/sec requires 5.625 Gbytes

- Five still images

  - 5 still images of size 512x512, 3 bytes for colour(RGB), requires 3.35Gbytes

This clearly calls for some form of compression in order to store data.

4.4 Summary

The chapter presented some of the work covered prior (or during) to this work in order to address the problem of low power DCT implementation. The common technique witnessed seems to be adaptively minimising the bitwidth of input pixel based on the statistics of image content. This research proposes a different approach, unique to the DCT implementation, whereby switched capacitance is reduced through manipulation and exploitation of correlation in cosine coefficients during the computation of the DCT coefficients.

The need for low power DCT was also highlighted in this chapter. Some figures are given to enhance the appreciation of how computation-costly the DCT can be.

Chapter 5 Column-Based DCT Scheme

5.1 Introduction

An exploration of the first scheme is presented by the use of the multiplier unit example. This involves the power analysis of the multiplier section of the proposed scheme environment. The multiplier unit is well-known for its computation intensive nature. This scheme is termed *Column-Based.*

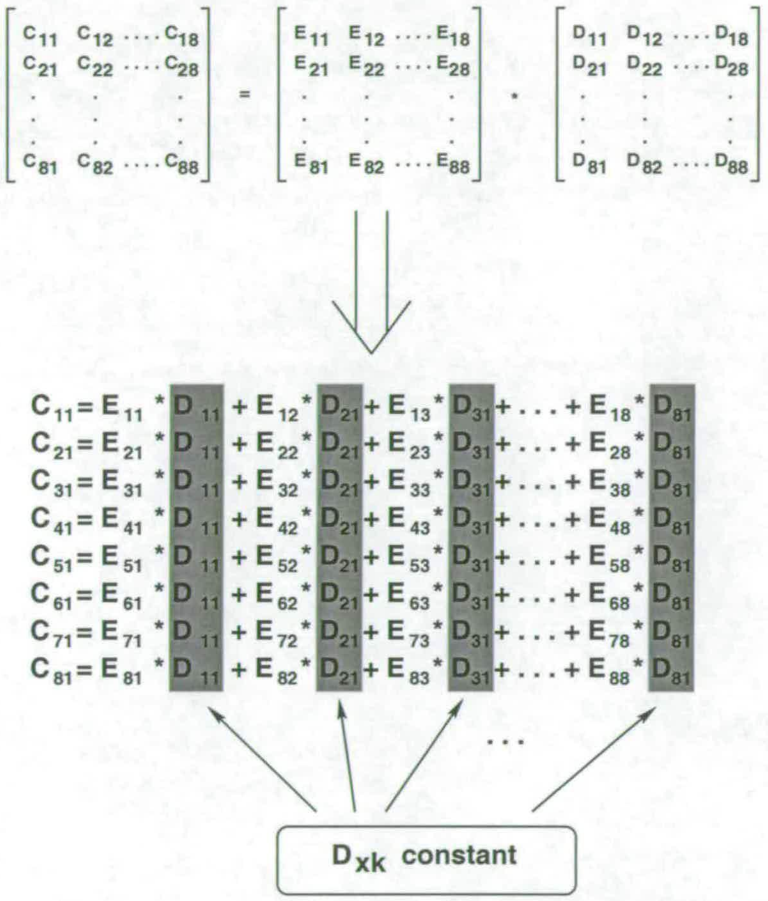
5.2 Column Based Processing

The computational bottleneck for the implementation of the DCT is the multiplication of the cosine coefficients matrix **[E]**, by the pixel matrix **[D]**, in order to obtain the DCT coefficients **[C]**, i.e.

$$[\mathrm{C}] = [\mathrm{E}] * [\mathrm{D}] \tag{5.1}$$

where each element C_{ik} in [C] matrix of order *n* is given by:

$$C_{ik} = \sum_{x=1}^{n} E_{ix} D_{xk} \tag{5.2}$$



C11 C12 .... C18
C21 C22 .... C28
C81 C82 .... C88
=
E11 E12 .... E18
E21 E22 .... E28
E81 E82 .... E88
D11 D12 .... D18
D21 D22 .... D28
D81 D82 .... D88
C11 = E11 * D11 + E12 * D21 + E13 * D31 + ... + E18 * D81
C21 = E21 * D11 + E22 * D21 + E23 * D31 + ... + E28 * D81
C31 = E31 * D11 + E32 * D21 + E33 * D31 + ... + E38 * D81
C41 = E41 * D11 + E42 * D21 + E43 * D31 + ... + E48 * D81
C51 = E51 * D11 + E52 * D21 + E53 * D31 + ... + E58 * D81
C61 = E61 * D11 + E62 * D21 + E63 * D31 + ... + E68 * D81
C71 = E71 * D11 + E72 * D21 + E73 * D31 + ... + E78 * D81
C81 = E81 * D11 + E82 * D21 + E83 * D31 + ... + E88 * D81
Dxk constant

Figure 5.1: *Matrix multiplication process*

A number of experiments were carried out with some cosine matrix and various pixel matrices. Careful analysis of the multiplication procedure revealed two distinct categories of cosine elements: (1) elements that are powers of 2, and (2) those which are non-powers of 2 numerics. For this reason, the algorithm processes the elements in (1) by performing a simple shift operation. It is well known that the switched capacitance of a shift is significantly less than that of a multiplication[58].

Examination of the multiplication procedure embodied in Equation 5.2 can reveal more scope for power optimisation. This can be demonstrated by expanding Equation 5.2 with n=8, as shown in Figure 5.1. As could be deduced from the figure, when the computation is performed on the basis of evaluating coefficients successively, i.e. C_{11}-C_{81}, then new pixel and coefficient values are processed at both inputs of the multiplier. If the multiplication procedure is performed on a column-after-column basis (i.e. $E_{i1}*D_{11}$, $E_{i2}*D_{21}$, and so on) then the value of D_{xk} will be constant (for 8 multiplications) at the corresponding multiplier input. This implies less switching activity at the multiplier inputs and memory buses.

For each column being processed, the multiplication procedure outlined above has the product form (m*constant), where m are the cosine matrix elements in the column being processed. The *constant* remains valid for only these values of m, if the column changes, a new *constant* is activated.

The multiplication results are unchanged irrespective of the order in which the *constant* is multiplied by the sequence of cosine coefficients in the column. For this reason, the algorithm performs the multiplication of elements in (2) in a column-by-column fashion and orders the elements, in the column being processed, according to some criteria. In this case, the criteria being minimum switching activity (hamming distance) between consecutive coefficients multiplied by the *constant*. This guarantees that similar elements are subsequently applied to the inputs of the multiplier causing minimum switching activity within the internal circuit of the

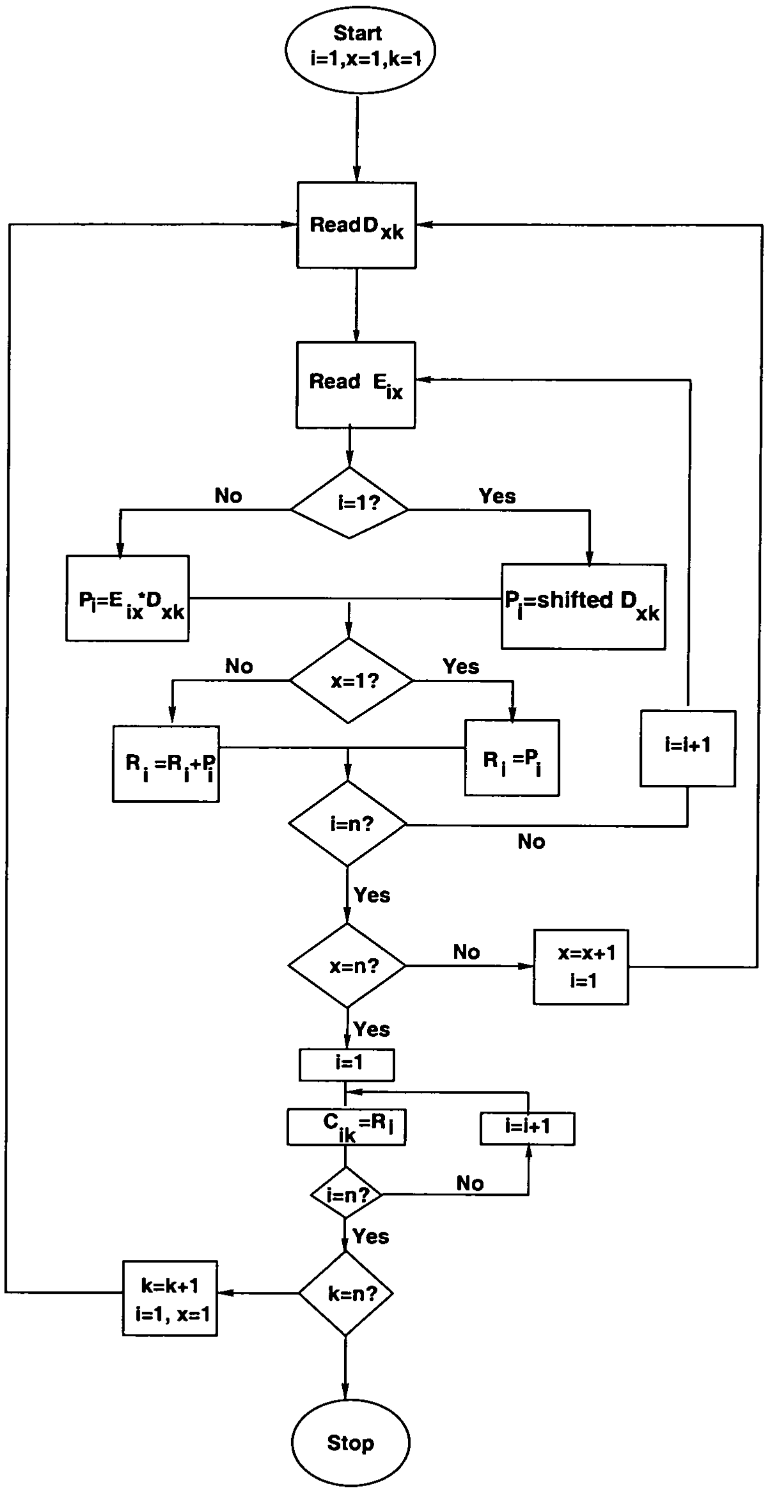


Figure 5.2: *Flowchart of the algorithm*

multiplier.

Hamming distance is defined as the number of digit positions in which the corresponding digits of two binary words of the same length are different. For example, the Hamming distance between 01101100 and 10001000 is 4. The ordering of binary words according to minimum Hamming distance uses this definition to arrange any two consecutive numbers such that the Hamming distance is less than any other arrangement.

Although the investigations were carried out using the example of ordering elements according to minimum hamming distance, in practice, any ordering algorithm can be used and *the amount of power saving is determined by the power of the algorithm used.* The steps in Figure 5.2 outline the algorithm, which commences with the following initialisation steps: (1) Process entries E_{1x} with shift operation. (2) Order remaining coefficients according to some ordering algorithm. (3) Save entries E_{ix} in (E) memory, see Figure 5.3, with elements of the same column being adjacent to each other, followed by the next column, and so on.

To illustrate the above algorithm, consider an example where;

$$\mathbf{E} = \begin{bmatrix} 2.00 & 2.00 & 2.00 & 2.00 \\ 1.85 & 0.77 & -0.77 & -1.85 \\ 1.41 & -1.41 & -1.41 & 1.41 \\ 0.77 & -1.85 & 1.85 & -0.77 \end{bmatrix}. \tag{5.3}$$

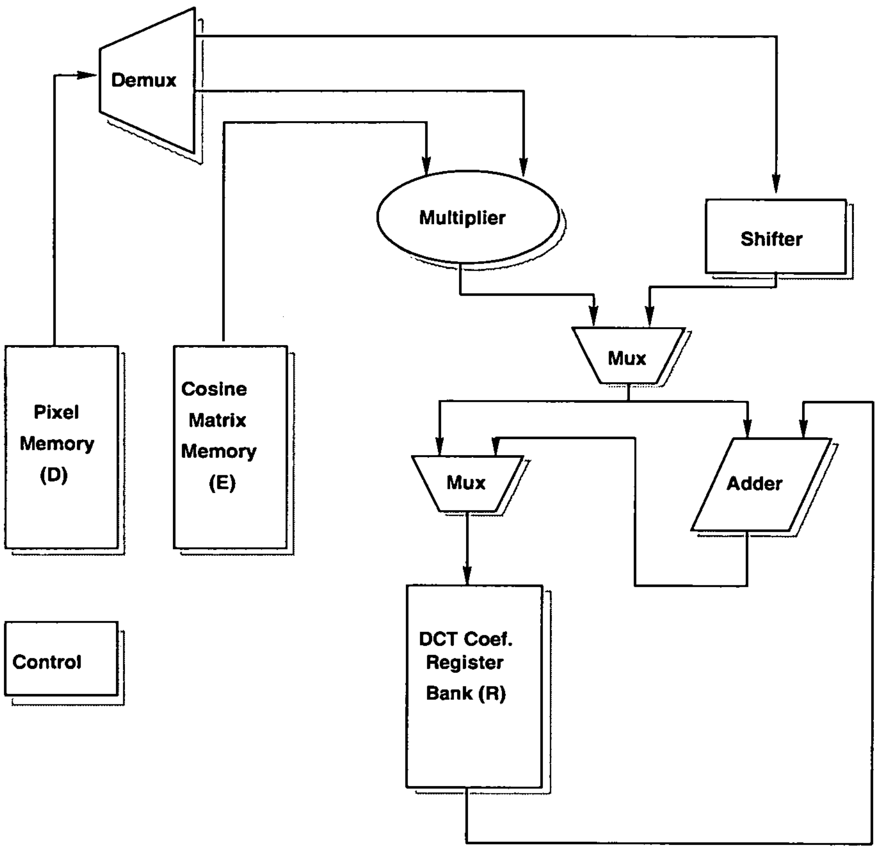


Figure 5.3: *Simplified architecture of the processor*

and

$$\mathbf{D} = \begin{bmatrix} 35 & 32 & 31 & 34 \\ 32 & 37 & 31 & 31 \\ 29 & 28 & 27 & 31 \\ 26 & 28 & 31 & 34 \end{bmatrix} \tag{5.4}$$

After the first iteration (i=1, x=1, and k=1), the first column, E_{i1} , is ordered according to minimum hamming distance to produce the ordered sequence (2.00, 0.77, 1.41, 1.85). The original locations of these elements are stored. Next the above sequence is multiplied by the first entry in D_{x1}, 35. The entries in registers R_i will be (70, 64.75, 49.35, 26.95), where the first entry (70) is saved in R_1 and the second entry is saved in R_2 and so on. Similarly at the end of iteration 2 (i=2, x=1, and k=1), R_i will contain (134, 89.39, 4.23, -32.25). The last iteration (i=4, x=1, and k=1) for this particular column, results in R_i containing the first column of the DCT coefficient matrix [C], i.e. R_i = C_{x1} $= (244, 18.96, 0, 1.35)^T$.This procedure is carried out until x =k = n, at which case R_i will contain C_{44}

As it stands, the algorithm can be implemented on traditional DSPs without any loss in throughput. However for high throughput applications, a modified processor architecture is required. The architecture, a simplified version of which is shown in Figure 5.3, requires an internal register bank in order to store the partial products, (R), which eventually result in the DCT coefficients, [C]. In addition, a memory unit is allocated for both the cosine and the pixel matrix elements, E_{ix} and D_{xk} respectively. A shifter is included to cope with the additional shift operations. The multiplier and the adder units are included to perform the normal multiply-add DSP operations. Since the outputs of both the multiplier and the shifter have to share the same input of the register bank, a multiplexer is needed to resolve which one output can use the register bank input. Another multiplexer is required to handle outputs from multiplier/shifter and adder units since some of the inputs to the register bank proceed directly to the bank without passing through the adder.

5.3 Simulation Results and Analysis

Figure 5.4 illustrates the framework utilised for the evaluation of the scheme with a number of 512x512 pixel example benchmark images. These, which include Lena, Man and checked images, are shown in Appendix C. The cosine coefficient matrix used was obtained using the MATLAB signal processing toolbox [71].

The evaluation environment is based upon the JPEG standards, where images are processed in blocks of 8x8. For this reason our results are obtained with n=8, i.e. an 8x8 cosine matrix is used with 512x512 pixel images being processed in blocks of 8x8. Some MPEG implementations subscribe to this processing. An 8x8-bit array Multiplier was constructed, using the Cadence VLSI suite with ES2 0.7 CMOS Technology, and processed down to layout level[72].

A C-program based *test-fixture* mapping system was developed to generate input simulation files for the Verilog-XL^{TM} digital simulator [73]. This involved forming the appropriate image-pixel/cosine-coefficient pairs, in the order imposed by the multiplication algorithm, so that they can be applied to the inputs of the hardware multiplier.

Three types of input simulation files are generated, representing the use of one of the following: (1) Traditional cosine DCT multiplication (*Traditional*). (2) Column-based multiplication algorithm without ordering (*Column-based*). (3) Column-based multiplication algorithm with ordering according to minimum hamming distance (*Hamming*). In each simulation, the number of signal transitions (switching activity), at the output of each gate, is monitored. Capacitive information for each gate is extracted from the layout of the multiplier circuit. Both of these are used to obtain a figure for the total switched capacitance of the multiplier.

Table 5.1 illustrates the results obtained with the different bench mark images. Clearly, power saving is achieved in all cases with a maximum of 50% with the checked image using *Hamming*, which reduces the amount of switched capacitance at both multiplier inputs. An average power saving of around 35% is achieved over all benchmark images.

The results illustrate that although the amount of power saving is proportional to the frequency of pixel value variation across the image, significant power saving is achieved with all image types. This includes those with sharp variation in pixel value, such as the checked image, and those with continuous spectrum of pixel values, such as Lena, Man and Peppers images (See Appendix C). This in turn implies the suitability of the scheme for wide ranges of images including coloured images.

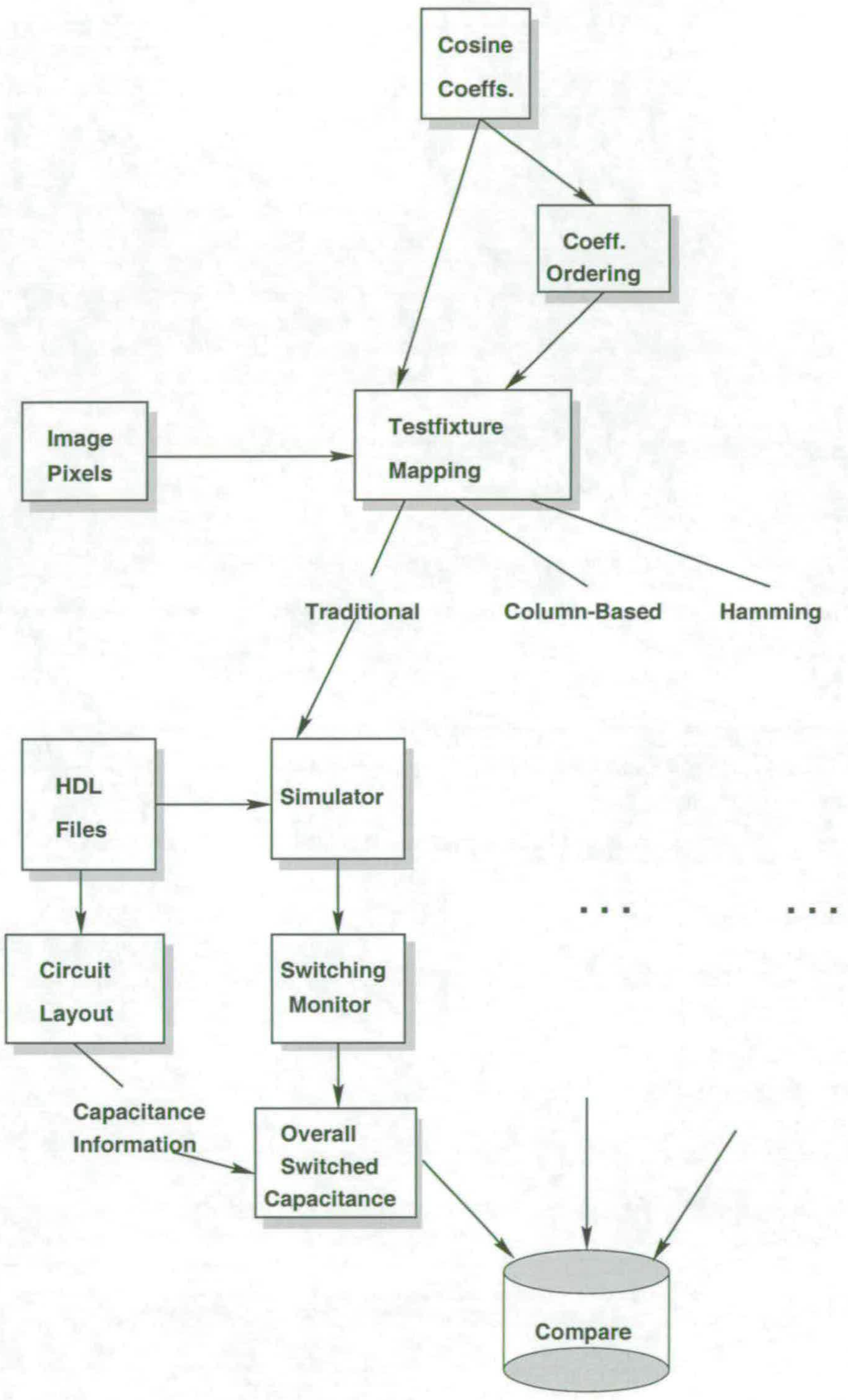


Figure 5.4: *Framework for algorithm evaluation*

Image	Ordering	Switched Capacitance (pF) $*10^3$	Switched Capacitance reduction (%)
Horizontal Stripes	Traditional	13.77	-
	Column-Based	8.45	38.64
	Hamming	7.06	48.64
Vertical Stripes	Traditional	7.70	-
	Column-Based	7.55	1.97
	Hamming	6.41	16.84
Blocks	Traditional	9.25	-
	Column-Based	7.65	17.27
	Hamming	6.47	29.99
Checked	Traditional	13.81	-
	Column-Based	8.39	39.25
	Hamming	6.83	50.50
Lena	Traditional	39.01	-
	Column-Based	29.51	24.35
	Hamming	26.23	32.77
Man	Traditional	40.05	-
	Column-Based	30.08	24.91
	Hamming	26.63	33.50
Peppers	Traditional	39.57	-
	Column-Based	29.75	24.84
	Hamming	26.49	33.05

Table 5.1: *Typical power savings*

5.4 Conclusions

The chapter presented an effective scheme for the low power implementation of the DCT within an JPEG/MPEG based environment. The scheme reduces power through the utilisation of shift operations, where possible, together with sequencing coefficient and pixel multiplications in an order dictated by the bit correlation of their successive values. This results in up-to 50% power reduction with a number of standard benchmark images. The investigations revealed that the scheme provides a generic framework for power saving in that significant power savings is achieved with a range of standard image types. These include those with sharp, continuous, and low pixel variation frequency. The scheme can easily be extended to the implementation of the 2D DCT. A full implementation of the scheme presented here is the subject of the next chapter which will also address possible overheads resulting from ordering of the cosine coefficients.

Chapter 6
Column-Based DCT Processor

6.1 Introduction

Based on the foundation laid by the previous chapter, this chapter maps the column-based scheme into a DCT processor. The resulting architecture is then analysed for power consumption with the results being compared with the power consumption of a conventional DCT processor.

6.2 Column-Based Scheme Implementation

This section will discuss the implementation of the column-based scheme. A brief summary of various components of the architecture will be given. If a component is built from other sub-components, the sub-components will subsequently be discussed under the component heading.

The proposed architecture for the column-based scheme is illustrated in Figure 6.1. The reference architecture for the original implementation used for comparison with the column-based scheme is shown in Figure 6.2. The original implementation is a direct mapping of the 1-D DCT equation in matrix form.

The emphasis of the implementation is on the data path of the proposed architecture. The

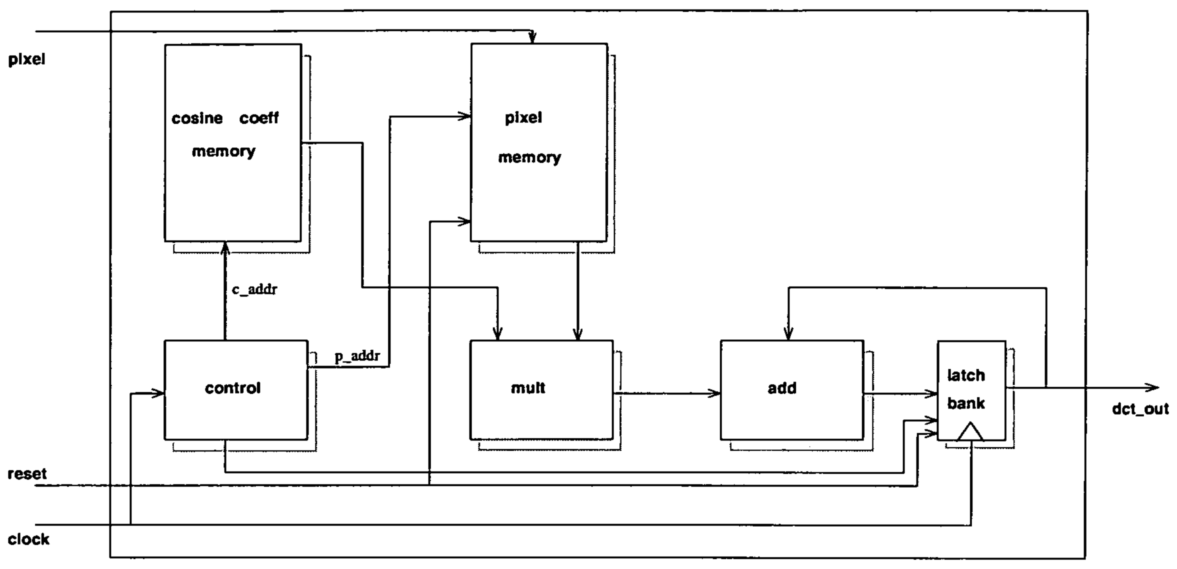


Figure 6.1: *DCT processor modified for Column based processing*

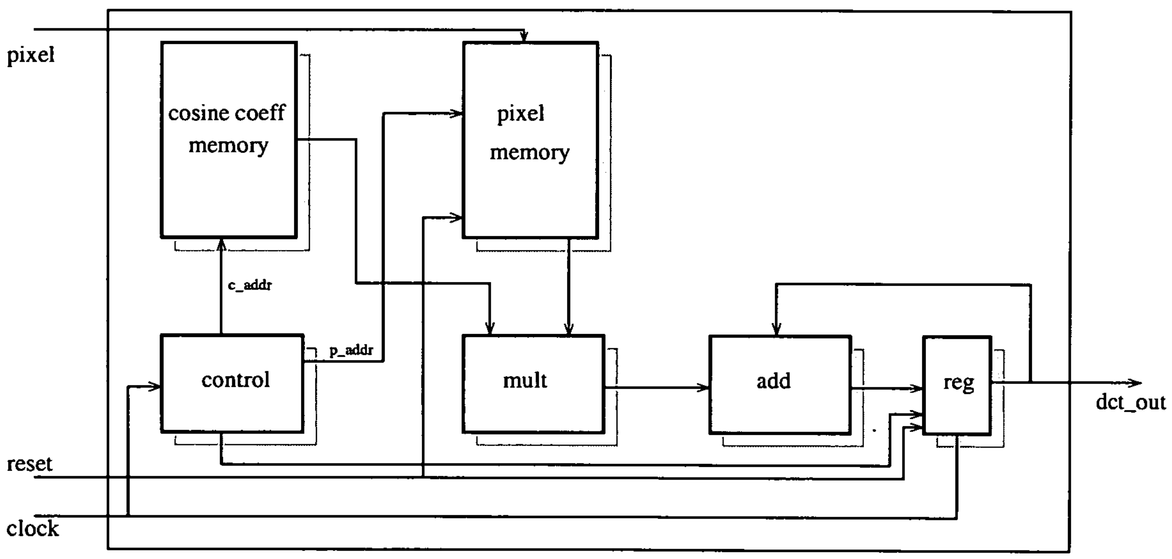


Figure 6.2: *Typical original DCT processor architecture*

data path for the design is considered to be the Multiply-Accumulate (MAC) unit shown in Figure 6.3. The column based MAC unit consists of the multiplier, an adder, and a latch bank. (see Figure 6.3). It should be noted that the MAC unit in this design differs from a conventional MAC unit in the sense that this one has a latch bank consisting of 8 latches whereas a normal MAC unit will have a simple register in the place of the latch bank. For the multiplier unit, two multiplier implementations were used, the csa, and Wallace architectures. These are listed in Table 6.1. The adders used are also listed in Table 6.1.

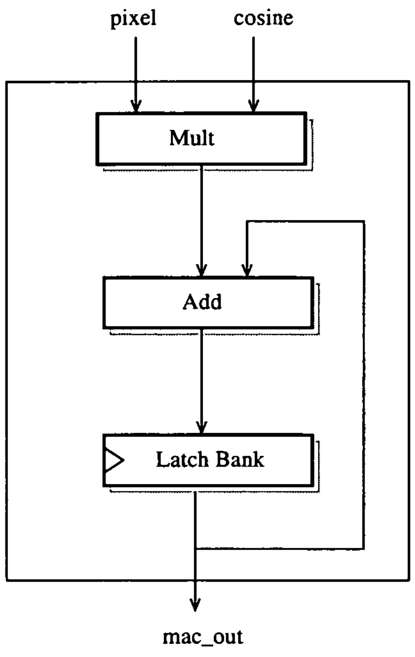


Figure 6.3: *Implementation of column based mac unit*

Adders	Multipliers
Brent-Kung (**bk**)	Carry-save array (**csa**)
Carry look-ahead (**cla**)	Booth-coded Wallace tree (**wall**)

Table 6.1: *some of the DesignWare components used*

6.2.1 Multipliers

The multiplier unit in the DCT processor is the most power hungry component. The previous section explored the scheme based only on the power consumption of the multiplier unit. There are several multiplier implementations reported in literature [74], but for the purpose of this project we used two implementations. It should be noted that since the project is based on comparing conventional DCT with the new schemes, it should not matter which component implementations are used, as long a the same modules are used in both designs.

The multipliers examined in the project are briefly introduced below.

Carry-save Array multiplier

The Carry-Save Array (csa) multiplier architecture used for the design is capable of processing signed numbers (two's compliment). The Baugh-Wooley algorithm [75] was developed to perform regular direct two's complement multiplication. It does not need prior converting of multiplicands/multipliers to unsigned numbers and then back to signed after processing.

Consider two numbers A and B as follows :

$$A = (a_{n-1}...a_0) = -a_{n-1}2^{n-1} + \sum_{i=0}^{n-2} a_i 2^i \tag{6.1}$$

$$B = (b_{n-1} \ldots b_0) = -b_{n-1}2^{n-1} + \sum_{i=0}^{n-2} b_i 2^i \tag{6.2}$$

The product A.B is given by the following equation, see Equation 6.3 :

$$A.B = a_{n-1}b_{n-1}2^{2n-2} + \sum_{i=0}^{n-2}\sum_{j=0}^{n-2} a_i b_j 2^{i+j} - a_{n-1}\sum_{i=0}^{n-2} b_i 2^{n+i-1} - b_{n-1}\sum_{i=0}^{n-2} a_i 2^{n+i-1} \tag{6.3}$$

If using this form, it can be seen that subtraction operations are needed as well as additions. However, negative terms may be written as shown in Equation 6.4.

$$-a_{n-1}\sum_{i=0}^{n-2} b_i 2^{n+i-1} = a_{n-1}(-2^{2n-2} + 2^{n-1} + \sum_{i=0}^{n-2} \overline{b_i}\, 2^{n+i-1}) \tag{6.4}$$

For this approach, A.B becomes

$$\begin{aligned} A.B \;=\; & a_{n-1}b_{n-1}2^{2n-2} + \sum_{i=0}^{n-2}\sum_{j=0}^{n-2} a_i b_j 2^{i+j} + a_{n-1}(-2^{2n-2} + 2^{n-1} + \sum_{i=0}^{n-2} \overline{b_i} 2^{n+i-1}) \\ & + b_{n-1}(-2^{2n-2} + 2^{n-1} + \sum_{i=0}^{n-2} \overline{a_i} 2^{n+i-1}) \end{aligned} \tag{6.5}$$

Equation 6.5 can be put in a more convenient form with the recognition of

$$-(a_{n-1}+b_{n-1})2^{2n-2} = -2^{2n-1} + (\overline{a_{n-1}} + \overline{b_{n-1}})2^{2n-2} \tag{6.6}$$

Therefore A.B becomes

$$\begin{aligned} A.B &= -2^{2n-1} + (\overline{a_{n-1}} + \overline{b_{n-1}} + a_{n-1}b_{n-1})2^{2n-2} \\ &+ \sum_{i=0}^{n-2}\sum_{j=0}^{n-2} a_i b_j 2^{i+j} + (a_{n-1}+b_{n-1})2^{n-1} \\ &+ b_{n-1}\sum_{i=0}^{n-2}\overline{a_i}2^{n+i-1} + a_{n-1}\sum_{i=0}^{n-2}\overline{b_i}2^{n+i-1} \end{aligned} \tag{6.7}$$

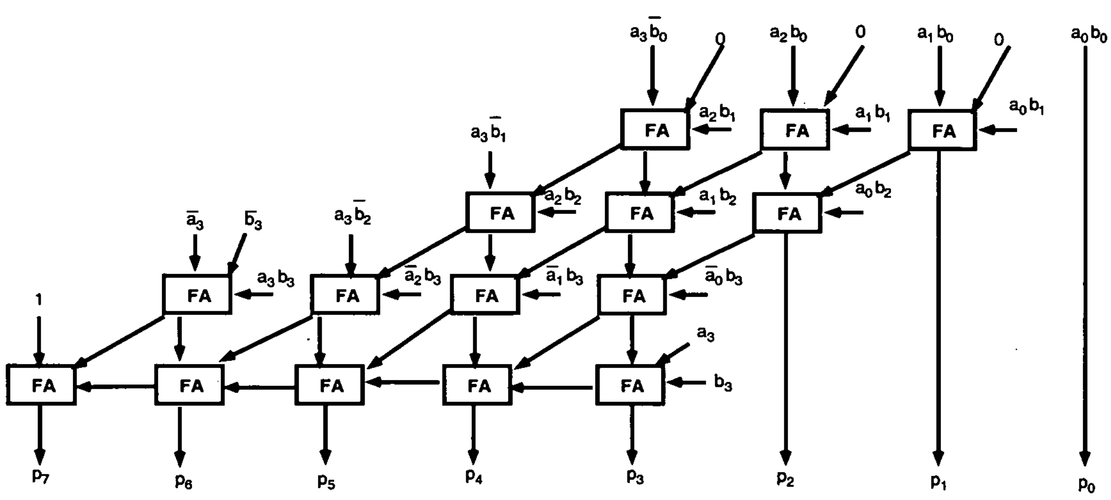
Using the above equations, an *n x n* multiplier can be implemented using only adder units. This is shown in Figure 6.4 for an 4x4 two's compliment multiplier. Figure 6.4(a) gives an example of the multiplication for a 4x4 multiplication whereas 6.4(b) depicts an implantation of the multiplier.

Booth-coded Wallace tree(wall)

This is a parallel implementation of a multiplier unit which combines two well-known approaches that enhance the speed of multiplication. These approaches are the Booth encoding technique and the Wallace tree reduction. It is reported in [76, 77] that the Booth encoding

$$\begin{array}{ccccccccc}
 & & & & & a_3\bar{b}_0 & a_2b_0 & a_1b_0 & a_0b_0 \\
 & & & & a_3\bar{b}_1 & a_2b_1 & a_1b_1 & a_0b_1 & \\
 & & & a_3\bar{b}_2 & a_2b_2 & a_1b_2 & a_0b_2 & & \\
 & & a_3b_3 & \bar{a}_2b_3 & \bar{a}_1b_3 & \bar{a}_0b_3 & & & \\
 & & \bar{b}_3 & & & b_3 & & & \\
 + & 1 & \bar{a}_3 & & & a_3 & & & \\
\hline
 & p_7 & p_6 & p_5 & p_4 & p_3 & p_2 & p_1 & p_0
\end{array}$$

(a) Example of Baugh-Wooley multiplication



(b) Baugh-Wooley multiplier implementation

Figure 6.4: *A signed Carry-save Array Multiplier example and implementation*

reduces the number of partial products by 50% while the Wallace tree increases the speed of summing the partial products[74, 77–79]. The steps involved in the scheme are listed below: See Figure 6.5

1. Booth encoding
2. Wallace tree reduction
3. final adder unit.

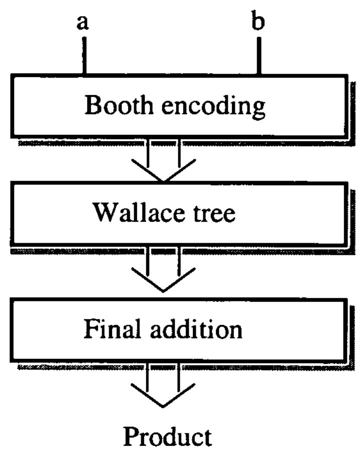


Figure 6.5: *General architecture for Booth-coded Wallace Multiplier*

1) **Booth Encoding**

Let the multiplier B be represented as

$$B = \sum_{i=0}^{\frac{n}{2}-1} (b_{2k-1} + b_{2k} - 2b_{2k+1})2^{2k} \qquad (6.8)$$

where $b_{-1} = 0$

Basically, three multiplier bits (b_{k+1}, b_k, b_{k-1}) are encoded into nine bits which are used to select multiples of the multiplicand value set (-2,-1,0,1,2) determined by Equation 6.8[78] [80]. The details of the Booth encoding/recoding scheme are illustrated in Table 6.2.

b_{k+1}	b_k	b_{k-1}	Encoded Digit	Operation on (X)
0	0	0	0	0.X
0	0	1	+1	+1.X
0	1	0	+1	+1.X
0	1	1	+2	+2.X
1	0	0	-2	-2.X
1	0	1	-1	-1.X
1	1	0	-1	-1.X
1	1	1	0	0.X

Table 6.2: *Booth Algorithm encoding scheme*

2) Wallace Tree Reduction

A Wallace tree is an implementation of an adder tree reduction designed for minimum propagation delay. Rather than completely adding the partial products in pairs like the ripple adder tree does, the Wallace tree sums up all the bits of the same weights in a merged tree, See Figure 6.6. Usually full adders are used, so that 3 equally weighted bits are combined to produce two bits: one (*the carry*) with weight of n+1 and the other (*the sum*) with weight n. Each layer of the tree therefore reduces the number of vectors by a factor of 3:2. There exists another reduction scheme which obtains a 4:2 reduction using a different adder style that adds little delay in an ASIC implementation. The tree has as many layers as is necessary to reduce the number of vectors to two (a carry and a sum)[80–82].

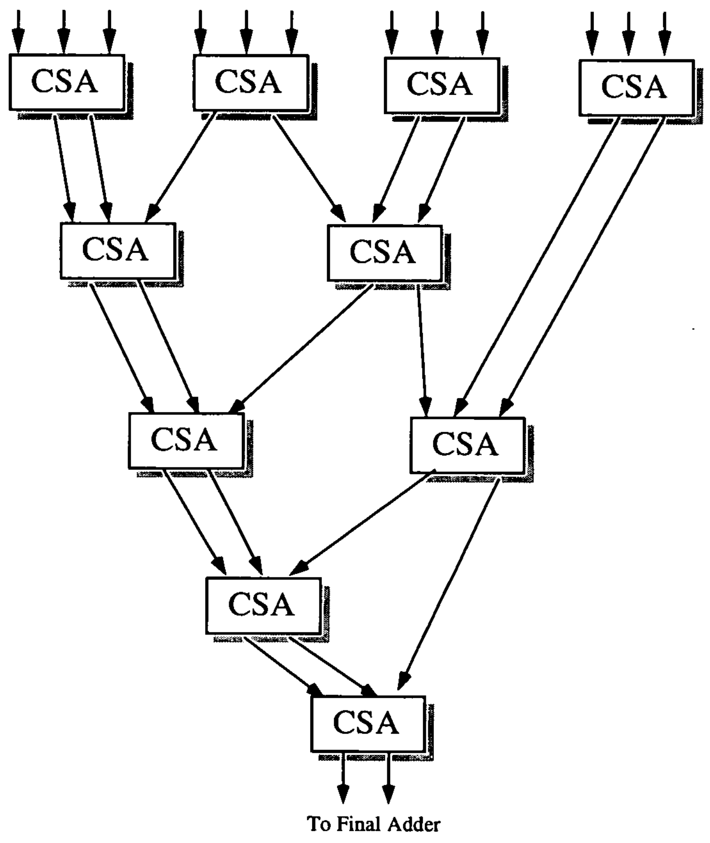


Figure 6.6: *Typical Wallace tree reduction*

Tracing the bits in the tree , an observation is made that the Wallace tree is a tree of carry-save adders. A carry save adder consists of full adders like the more familiar ripple adders, but the carry output from each bit is brought out to form the second result vector rather than being wired to the next most significant bit. The carry vector is saved to be combined with the sum later[77, 83].

3) **Final Adder Unit**

After the Wallace tree reduction, there is a need for a final addition stage. The ripple adder is used for this purpose. A typical ripple adder is depicted in Figure 6.7.

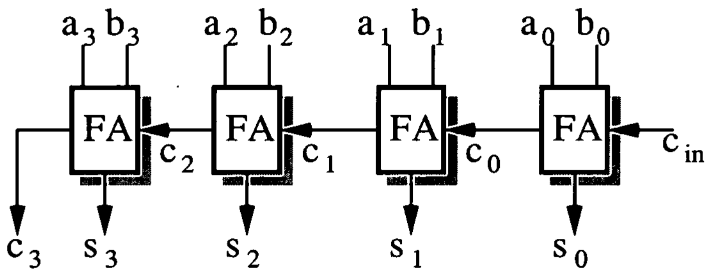


Figure 6.7: *Final Addition Using ripple adder*

6.2.2 Adders

A brief overview of the adders used in the designs follows.

Carry look-ahead (cla)

1. The disadvantage of Ripple Carry Addition is that each carry is dependent on the previous carry, which results in a long delay. Carry-Lookahead Addition produces the carries in parallel.

2. The CarryOut logic equation is

$$CarryOut = (a \cdot b) \oplus (a \oplus CarryIn) \oplus (b \cdot CarryIn) \tag{6.9}$$

This can be rewritten as

$$\begin{aligned} c[i+1] &= (a[i] \cdot b[i]) \oplus (a[i] \cdot c[i]) \oplus (b[i] \cdot c[i]) \\ &= (a[i] \cdot b[i]) \oplus (c[i] \cdot (a[i] \oplus b[i])) \end{aligned} \tag{6.10}$$

where c[i] is the carry into the i-th full adder.

3. We define

$$g[i] = a[i] \cdot b[i] \tag{6.11}$$

$$p[i] = a[i] \oplus b[i] \tag{6.12}$$

and rewrite the carry equation as

$$c[i+1] = g[i] \oplus (c[i] \cdot p[i]) \tag{6.13}$$

4. For a 4-bit Carry-Lookahead Adder the carry equations are

$$\begin{aligned}
c[1] &= g[0] \oplus (c[0] \cdot p[0]) \\
c[2] &= g[1] \oplus (c[1] \cdot p[1]) \\
&= g[1] \oplus (p[1] \cdot g[0]) \oplus (p[1] \cdot p[0] \cdot c[0]) \\
c[3] &= g[2] \oplus (c[2] \cdot p[2]) \\
&= g[2] \oplus (p[2] \cdot g[1]) \oplus (p[2] \cdot p[1] \cdot g[0]) \oplus (p[2] \cdot p[1] \cdot p[0] \cdot c[0]) \qquad (6.14)
\end{aligned}$$

The 4th carry can be produced using a regular full adder.

Brent-Kung(bk)

The Brent-Kung adder is identified by its regular layout [84]. This was first presented by Brent and Kung in [85]. One of the distinct characteristics of the Brent-Kung Adder is that computation time increases logarithmically as the number of bits increases linearly.

$$t_{add} \propto \log_2 N \qquad (6.15)$$

Define an operator o such that

$$(g, p) \circ (g', p') = (g + p \cdot g', p \cdot p') \qquad (6.16)$$

where g and g' are *carry generate* terms formed any two input bits [85].

p and p' are *carry propagate* terms.

Also

$$(G_i, P_i) = \begin{cases} (g_i, p_i) & if i = 1 \\ (g_i, p_i) \quad (G_{i-1}, P_{i-1}) & if 2 \leq i \leq n \end{cases} \tag{6.17}$$

Equation 6.17 together with the fact that $\circ$ is associative, is the fundamental key that supports the Brent-Kung approach. The associative property allows the computation of (G_i, P_i)'s in any order. Hence G_i can be considered as a block carry generate signal and P_i as a block carry propagate signal. The problem of computing (G_i, P_i), for $i = 0$ to 16 can now be done in the order defined a binary tree as in Figure 6.8. The main components of the carry generation tree are the $\circ$ operator denoted in the diagram by white circles.

6.2.3 Latch Bank

The latch bank module consists of a demultiplexor, gating module, 8 latches, and a multiplexor. See Figure 6.9. The components of this module are briefly described as follows.

Demux

The demux block is simply a 8 to 1 demultiplexor used to decode the write address. A single data line can be connected to a number of output lines by appropriate choice of signal on the select lines as shown in Figure 6.10. If a demultiplexor contains s select lines, then the number of possible output lines n to which data can be routed is $n = 2^s$. A demultiplexor

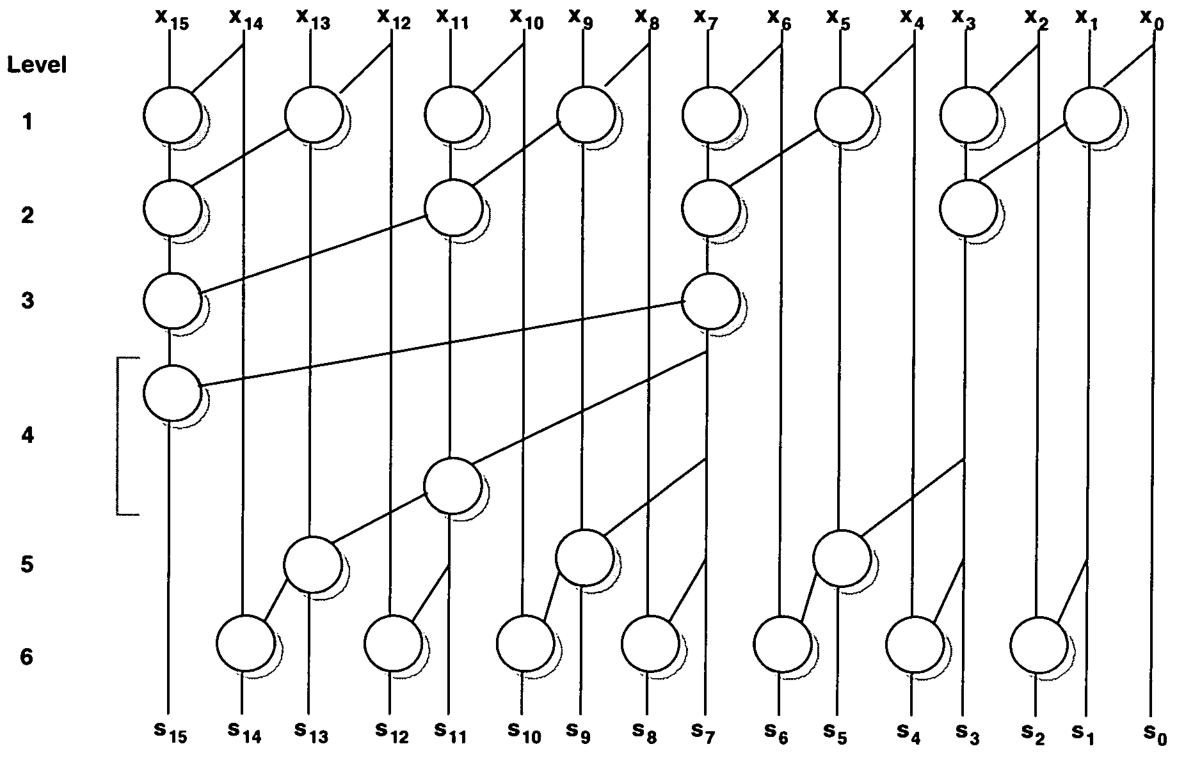

Level
1
2
3
4
5
6
x_{15} x_{14} x_{13} x_{12} x_{11} x_{10} x_9 x_8 x_7 x_6 x_5 x_4 x_3 x_2 x_1 x_0
s_{15} s_{14} s_{13} s_{12} s_{11} s_{10} s_9 s_8 s_7 s_6 s_5 s_4 s_3 s_2 s_1 s_0

Figure 6.8: *Brent-Kung Parallel Prefix Network*

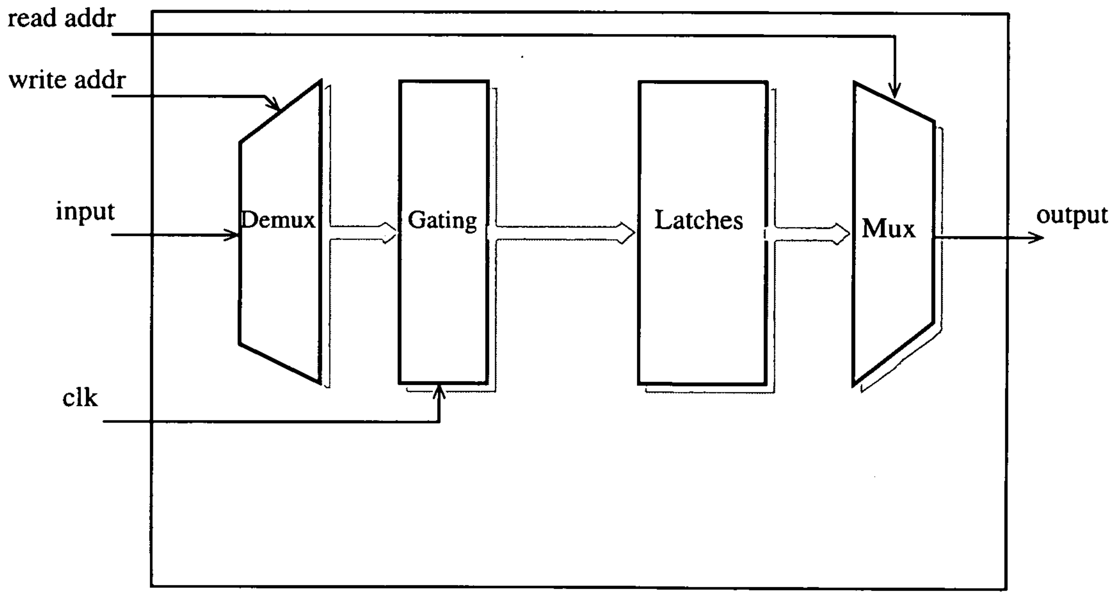


Figure 6.9: *Latch bank module*

that contains an enable signal can serve as a decoder[86].

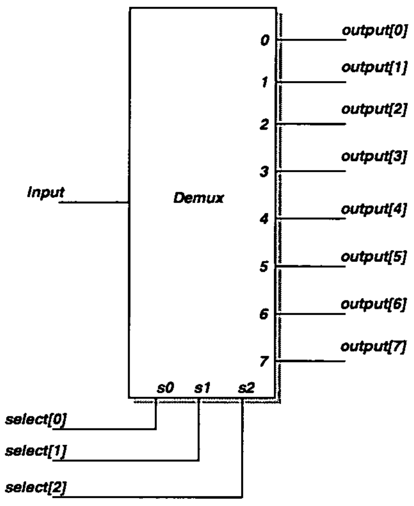


Figure 6.10: *Demultiplexor*

Gating

The next module in the bank is the gating logic. Gating the clock is a powerful technique used in low power design to disable unused modules of a circuit. Gating can create power savings by both preventing unnecessary activity in the logic modules as well as eliminating power dissipation in the clock distribution[87, 88].

This unit gates the clock such that only the latch which needs to be written is activated.. This technique helps reduce the power consumption of the device by making sure that only the one latch toggles at any given time. The example Figure 6.11 relates to the output of the Demultiplexor for the first latch (*at address 0*). Gating signal *clk* with the address enable *in_0*, this will produce latch enable *enable_0* for the first latch only when both *clk* and *in_0* are asserted. Hence all other latches will not be activated within the current *clk* period.

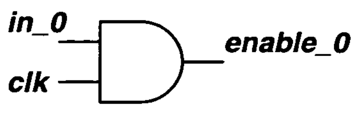


Figure 6.11: *Gating example for first Latch*

The Gating module consists of 8 sets of gating logic as in Figure 6.11. This implies that there is gating logic for each Latch in the Bank.

Latches

A latch is a level sensitive device. The output of a latch (*Q*) is transparent to the input (*D*) as long as enable signal is activated. If the enable signal is disabled the output (*Q*) remains constant and any change that occurs in the input is ignored. See Figure 6.12 for the latch diagram.

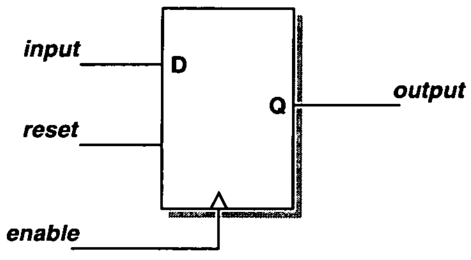


Figure 6.12: *Latch*

For this design a *reset* signal is required in the storage element because there is a need to clear the contents and start a new computation when a fresh column from the pixel matrix **[D]** is being processed.

The advantage of a latch based storage over register based storage is that they are considerably smaller (gate count) than registers. Therefore they consume less power. But a designer has to be aware of the pit-falls of latch based designs. They can be prone to glitches and hold time violations. The glitches can be avoided by using a glitch free enable. For the hold time violation, one should ensure that data is held for long enough as the latch is closed. Figure 6.13 illustrates a latch array. This is an array of 8 latches that will be required to implement the storage element. Since the output of the multiplier is 16 bits in total, each latch in the array should be capable of storing 16 bits. It should be noted that the precision of the data path is 16 bits. This implies that as the wordlength of the result increases beyond 16 bits, it is truncated. This truncation will affect the quality of the resulting image.

Mux

The function of the Mux is to multiplex the outputs of the 8 latches on to a single data bus which is fed back to the adder module and/or output out of the DCT processor module. A multiplexor routes 1 of many inputs to its output under the control of 1 or more select signals. See Figure 6.14.[89].

6.3 Synthesis and Simulation Results

The data path for the DCT (MAC unit) and its test-bench were implemented in Verilog. The input images that were used for the simulations were preprocessed to chop them into blocks of 8x8 matrices. The images tested can be found in Appendix C. The designs were synthesised with Synopsys's *Design Compiler* using the Alcatel 0.35μm CMOS library.

The simulation environment is shown in figure 6.15. An image is segmented into blocks

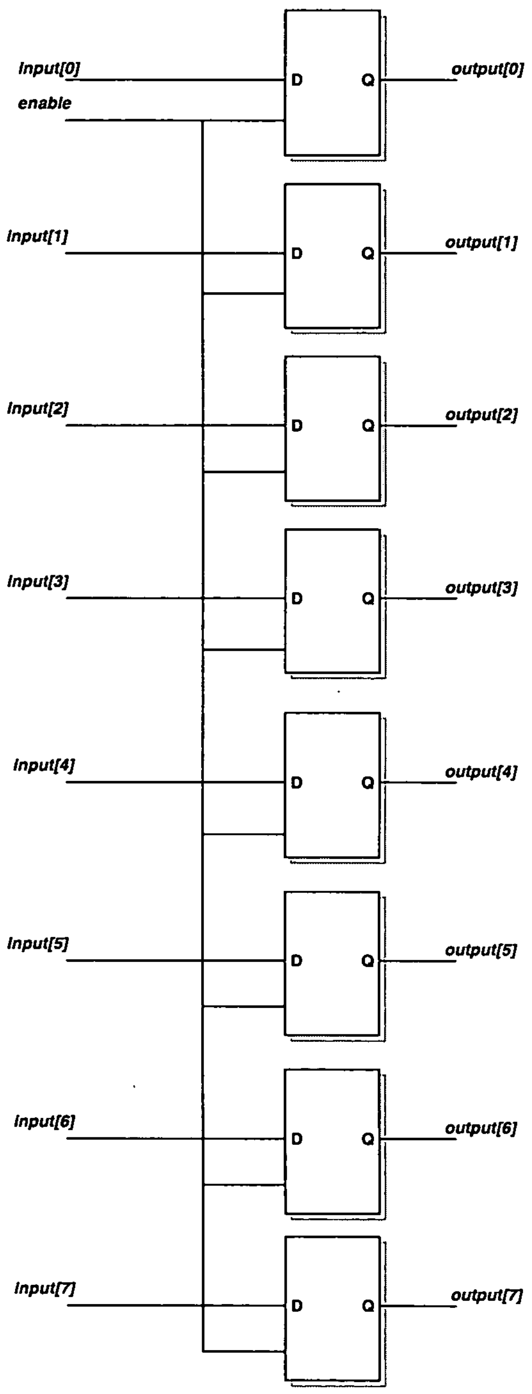


Figure 6.13: *Latch array*

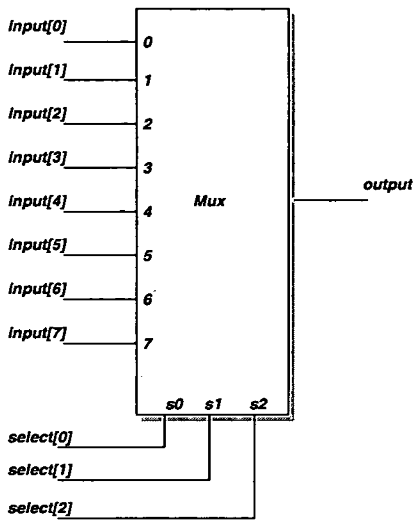


Figure 6.14: *Multiplexor*

of 8x8 using a perl script and this is input into the testbench environment. The temporary storage of the test image is 64x8. On the other input channel, the cosine coefficients, another array of 64x8 size is used. The elements in this memory module are sorted out according to some algorithm, (either hamming, ascending, or any other algorithm) and then passed to the testbench for processing. It should be noted that any algorithm that minimises the bit toggling between any consecutive inputs can be used with this scheme.

During simulation, the toggle information is logged into a file as bits are being processed. This is referred to as the *SAIF/toggle* file. The information is then annotated into Synopsys DesignPower for the estimation of the power consumption. Appendix D summarises the DesignPower strategy for power analysis.

Tables 6.3 and 6.4 presents the results for the Column-based DCT scheme and conventional DCT respectively.

Image	Implementation	Power(mW)			mult	add	latch bank
		cip	nsp	tdp	(cip)	(cip)	(cip)
lena	ascending	5.91	6.79	12.69	1.17	2.58	2.15
	hamming	5.5	6.33	11.82	1.03	2.40	2.05
wallace	ascending	7.30	8.41	15.71	1.91	3.17	2.20
	hamming	7.18	8.25	15.43	1.81	3.16	2.20
checked	ascending	6.42	7.42	13.84	1.58	2.68	2.15
	hamming	6.24	7.17	13.42	1.36	2.66	2.20

Table 6.3: *Column Based dct results for bk with csa*

Image	Power(mW)			mult	add	reg
	cip	nsp	tdp	(cip)	(cip)	(cip)
lena	4.89	5.59	10.47	2.35	1.67	0.87
wallace	5.03	5.73	10.75	2.43	1.72	0.88
checked	5.41	6.45	11.87	3.03	1.63	0.76

Table 6.4: *Conventional dct results for csa with bk*

Image	Implementation	Power (mW)	Power savings (%)
lena	conventional	2.35	-
	hamming	1.03	56.17
	ascending	1.17	50.21
wallace	conventional	2.43	-
	hamming	1.81	25.51
	ascending	1.91	21.40
checked	conventional	3.03	-
	hamming	1.36	55.11
	ascending	1.58	47.85

Table 6.5: *comparison of the multiplier units*

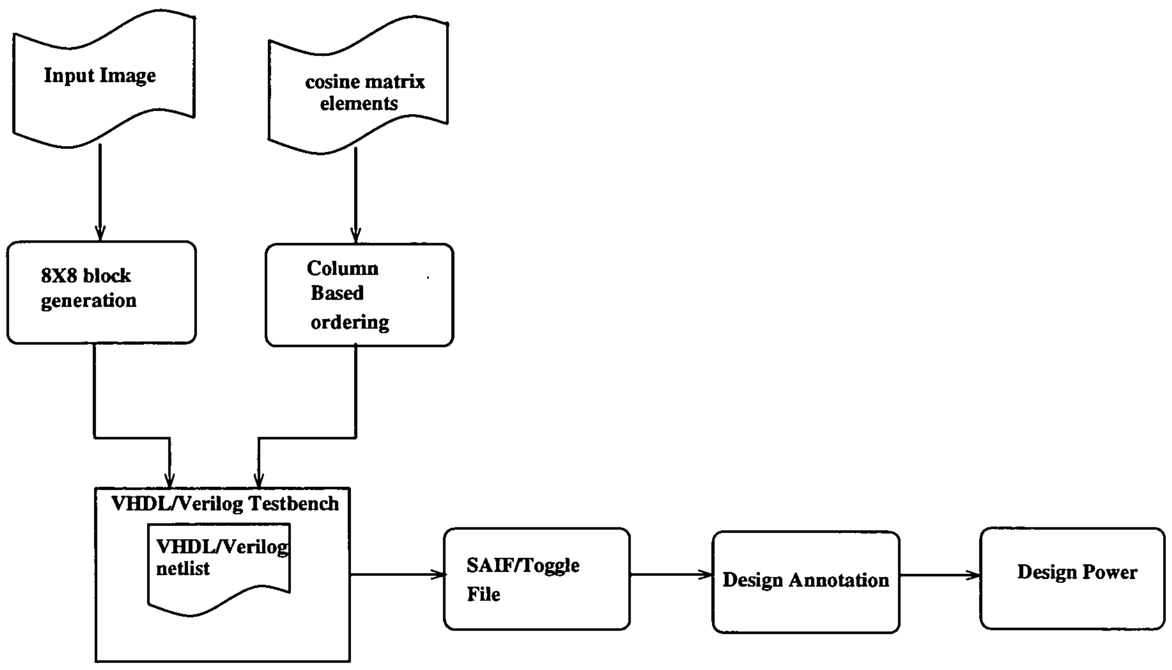


Figure 6.15: *Simulation environment*

Table 6.5 presents the power dissipation within the multiplier units. The power consumption of the multiplier unit when using the scheme is compared to that of the conventional implementation. A percentage of the difference relative to the conventional implementation is presented. This reflects that although there is an overall overhead in the design, the multiplier section is still saving some energy compared to the conventional implementation. The resulting overhead is due to extra logic required to implement this scheme. A set of 8 registers/latches is required compared to only one in the conventional implementation. The overhead is not of too much concern since for the implementation of a 2D DCT, an intermediate memory unit is required for the transposition process. This transposition memory can be reused as a temporary storage for the partial sums. Another source of extra circuitry comes from addressing the re-ordered cosine coefficients.

Image	Implementation	Power(mW)			mult	add	latch bank
		cip	nsp	tdp	(cip)	(cip)	(cip)
lena	wall+bk	5.59	6.28	11.88	0.97	2.24	2.21
	wall+cla	5.15	5.84	11.00	0.97	1.96	2.23
	csa+cla	4.53	5.08	9.61	0.95	1.85	1.73
wallace	wall+bk	6.16	6.87	13.03	1.26	2.52	2.38
	wall+cla	5.75	6.40	12.15	1.27	2.12	2.36
	csa+cla	6.05	6.82	12.87	1.90	2.52	1.63
checked	wall+bk	5.95	6.63	12.58	1.31	2.26	2.37
	wall+cla	5.44	6.23	11.68	1.31	1.89	2.34
	csa+cla	5.39	6.11	11.50	1.30	2.17	1.92

Table 6.6: *Column Based dct results for other implementations with hamming sort*

Image	Implementation	Power(mW)			mult	add	latch bank
		cip	nsp	tdp	(cip)	(cip)	(cip)
lena	wall+bk	6.26	7.03	13.26	1.18	2.72	2.74
	wall+cla	5.66	6.37	12.00	1.18	2.15	2.33
	csa+cla	5.08	5.71	10.79	1.11	2.13	1.83
wallace	wall+bk	6.52	7.32	13.84	1.52	2.62	2.39
	wall+cla	6.09	6.81	12.91	1.52	2.19	2.37
	csa+cla	6.29	7.13	13.42	1.79	2.58	1.93
checked	wall+bk	6.24	6.98	13.23	1.55	2.35	2.33
	wall+cla	5.82	6.49	12.32	1.55	1.96	2.31
	csa+cla	5.46	6.19	11.64	1.44	2.14	1.88

Table 6.7: *Column Based dct results for other implementations with ascending sort*

For other multiplier and adder combination implementations, the results are presented in Tables 6.6 and 6.7. The effects of varying the word length of the inputs to the datapath are presented in the Table 6.8. As can be seen from the results, the multiplier unit within the MAC saves power despite the increased wordlength.

Implematation	Wordlength	Power(mW)			mult	add	latch bank
		cip	nsp	tdp	(cip)	(cip)	or reg(cip)
Column-based	8 bits	5.50	6.33	11.82	1.03	2.40	2.05
	16 bits	9.63	8.61	18.24	3.97	2.56	3.10
	24 bits	12.94	13.73	26.68	5.34	3.83	3.77
Conventional	8 bits	4.89	5.59	10.47	2.35	1.67	0.87
	16 bits	9.18	11.77	20.85	5.50	2.34	1.35
	24 bits	17.28	15.37	32.64	9.87	4.45	2.95

Table 6.8: *Wordlength variations: for csa and bk implementations with lena image*

6.4 Conclusions

This chapter presented an implementation of the low power scheme for DCT processing introduced in chapter 5. The scheme reduces power through the utilisation of sequencing coefficient and pixel multiplications in an order dictated by the bit correlation of their successive values. This results in up-to 56% power reduction within the multiplier unit for a number of standard benchmark images.

Since the scheme requires saving of partial summations prior to outputting the DCT coefficients, a set of 8 registers is required compared to the conventional implementation which requires only one. This register bank has a significant overhead on the overall power saving such that the advantage revealed within the multiplier section is diminished. Theoretically, only two registers/latches are expected to be switching when the scheme is in operation. One read register and one write register, so the expected power overhead should be at least twice the consumption of the single register in the conventional implementation. This however is not completely unacceptable since in cases where a 2D DCT implementation is required, the intermediate memory for transposition can be used to temporarily store these partial sums and no extra circuitry will be required. Another source of overheads is the addressing of the ordered cosine coefficients. This requires extra circuitry in the control logic and also the original location of the cosine coefficient entry to saved. Hence a wider storage unit is required.

Different combinations of multiplier and adder units were analysed for power consumption. In all combinations, the column-based implementation saved power within the multiplier unit.

Chapter 7
Cosine Ordering Scheme

7.1 Introduction

The scheme presented in this chapter takes advantage of the rows in the cosine matrix. Ordering the cosine coefficients in each row according to some ordering technique such as, ascending order or minimum hamming distance, minimises the toggles at the inputs of the multiplier unit in the DCT processor. Reducing the number of toggles at the inputs of the multiplier reduces power consumption in the unit [4, 90]. Since the multiplier unit is the most computationally expensive unit in the data path of a DCT processor, it provides the most advantage if it is targeted with a power conscious approach.

The scheme utilises the cosine coefficients ordering as in [90], however, the ordering is done in the rows of the cosine matrix rather than the columns. The advantage of ordering the coefficients in rows can be easily identified by the fact that no modification is needed in the data path as in [90]. Therefore the overheads will be minimum since the same data path is used as the one in the conventional implementation.

This chapter presents the implementation of the scheme, starting with the scheme exploration at multiplier unit to the MAC unit. Some results are presented which reflect a power saving of up-to 24%.

7.2 Order-Based Processing

The cosine coefficients [E] for any DCT implementation are constant, therefore ordering can be done prior to loading them into a ROM unit. When the ordering is undertaken, the original row location of the coefficient is tagged along with the corresponding cosine coefficient word. The proposed cosine coefficient representation is shown in Figure 7.1. This consists of two portions:

1. the original location of the cosine coefficients with a row in the cosine matrix.
2. the cosine coefficient value

This allows the saved location to be used as the address of the corresponding pixel element to be computed. The three MSB's of Figure 7.1 represent the pixel row location in the pixel matrix, whereas the remaining bits represent the value of the cosine coefficient.

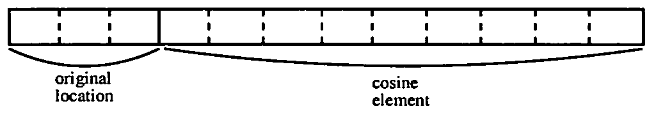


Figure 7.1: *cosine coefficient with saved location*

The flowchart for the implementation is depicted in Figure 7.2. The flowchart assumes that ordering of the cosine coefficients has already been done prior to commencement of the steps in the chart. Note that the flowchart is for processing one block of the image (8x8). For the processing of an entire image, the complete flowchart should be followed in order to process each 8X8 block. The steps for the scheme are listed below.

1. Decode the cosine word *n*E into its constituent parts. ie, pixel location (*n*) and cosine word(E)
2. Get the value pixel(D) from location specified by *n* in (1)
3. Multiply ; E * D and add the result to the accumulator
4. Repeat steps (1) to (3) for the remaining row entries
5. Obtain the DCT coefficient (C) and clear the accumulator
6. repeat 1) to 5) until the last row of cosine matrix is processed
7. Restart processing of the next image block

A simplified DCT processor for the proposed scheme is depicted in Figure 7.3. This differs with the conventional implementation by the fact that a slice of the data coming from *cosine coeff memory* is used as an address for the *pixel memory.* This slice is the *original location* portion of the cosine word as depicted in Figure 7.1.

The depth of the *pixel memory* is 64, therefore a 6 bit address is required to access a memory of this depth. This problem is alleviated by using a 3 bit counter for the rest of the 6 bit address. The address from the counter is appended as the most significant bits of the pixel memory address to make a 6 bit word. The counter is incremented after every 8 memory accesses. The overhead for this extra circuitry is very minimum since ordinarily a 6 bit counter would have been required to generate the address for the *pixel memory.* Another issue that might arise from this design will be the larger data word size of the *cosine coeff*

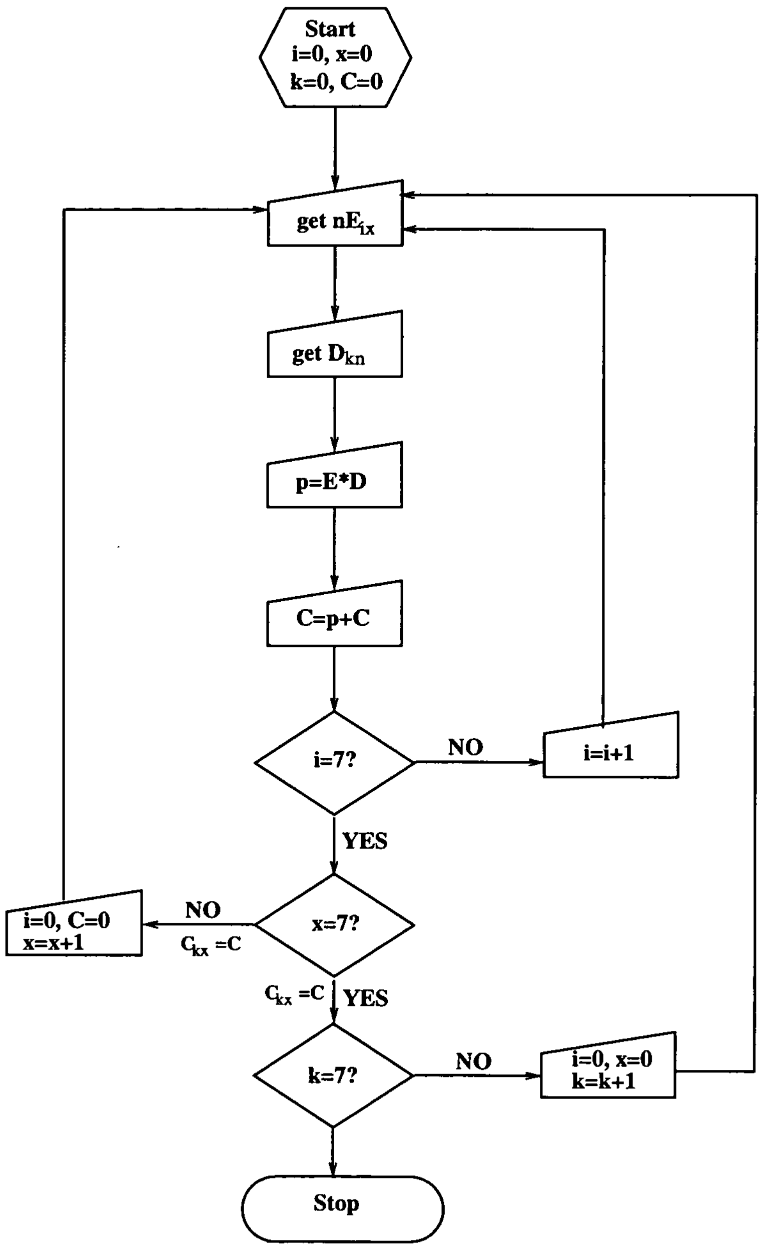


Figure 7.2: *Flowchart of cosine ordering scheme*

memory compared to the conventional implementation. This is covered by the fact that on a conventional implementation a 6 bit address bus will be required to connect between the *control* and the *pixel memory* (the dashed connection in Figure 7.3, labelled *p_addr*). So, as far as bus toggling is concerned, for the proposed implementation there are 3 bits that can toggle compared to the 6 bits that would be generated otherwise.

The multiplier implementations that were used for the scheme exploration are the same as the ones used in the previous chapter. These are listed here for better clarity; the **csa, nbw** and **wall.**

The emphasis of the implementation is on the data path of the proposed architecture. The data path for the design is considered to be the multiply and accumulate (MAC) unit shown in Figure 7.4. The column based MAC unit consists of the multiplier, an adder, and a register. For the multiplier unit, several multiplier implementations were used, the csa, nbw, and wallace architectures. This are listed in Table 7.1. The adders used are also listed in Table 7.1.

Adders	Multipliers
Brent-Kung (**bk**)	Carry-save array (**csa**)
Carry look-ahead (**cla**)	Non-Booth coded Wallace tree (**nbw**)
	Booth-coded Wallace tree (**wall**)

Table 7.1: *some DesignWare components*

The same MAC unit architecture (see Figure 7.4)was used for both the conventional implementation and the cosine ordering scheme. The modules in the MAC unit are mult, add and reg. The module *mult*, was implemented using *Baugh-Wooley* multiplier. For the *add* module, *Brent-Kung* adder implementation was used. After testing several adders, the *Brent-Kung* adder proved to be more power efficient and suitable for high speed applications . An ordinary register with reset, clock and enable signal was used for the module *reg*. The enable signal is asserted after every 8 multiplications to clear the register.

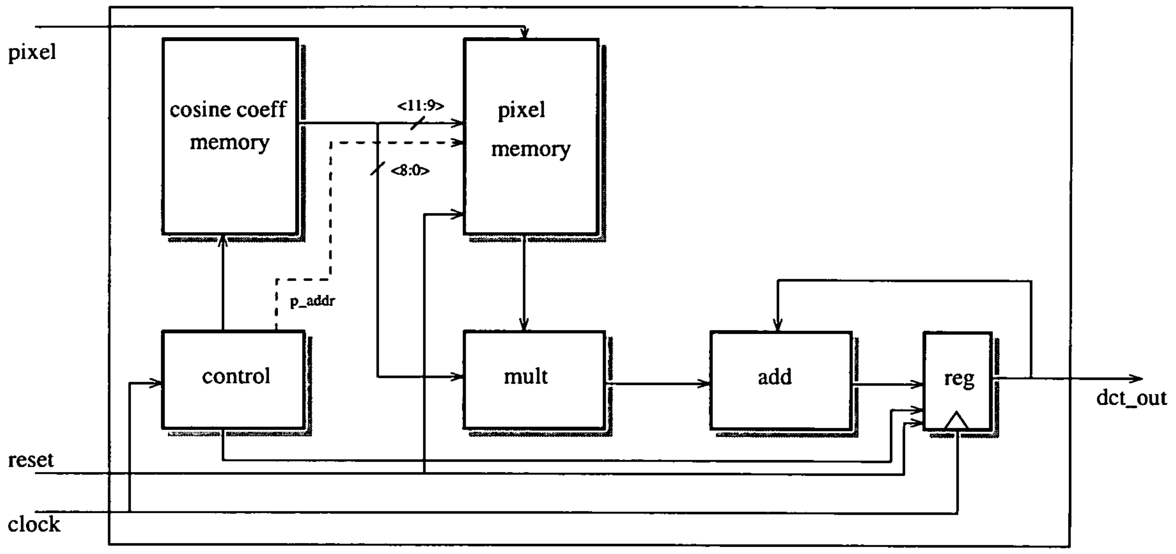


Figure 7.3: *A simplified DCT processor architecture*

To illustrate the above algorithm, consider an example where the cosine matrix is as shown in Figure 7.5(a). After sorting, according to minimum hamming distance, the cosine matrix will be as in Figure 7.5(b).

As an example consider the multiplication of nE (Figure 7.5(b)) with the following matrix.

$$\mathbf{D} = \begin{bmatrix} 35 \\ 32 \\ 29 \\ 26 \end{bmatrix}$$

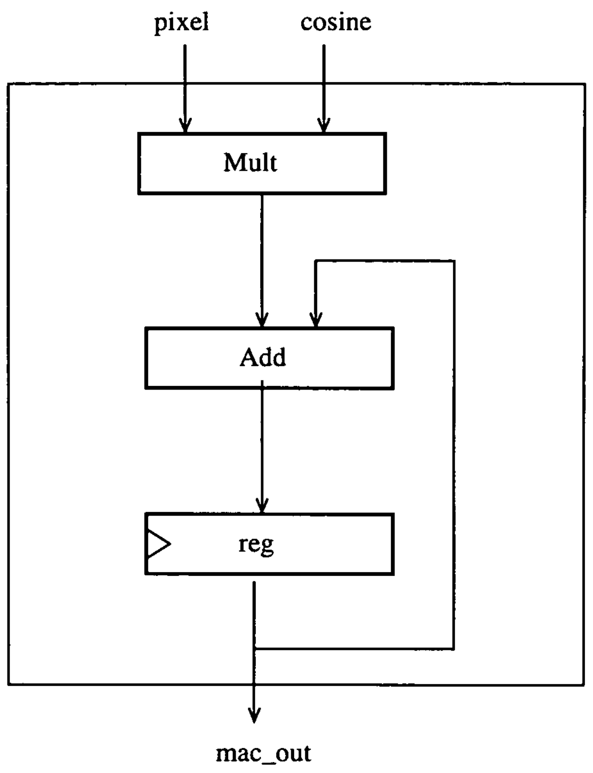


Figure 7.4: *A simplified MAC unit architecture*

$$\begin{bmatrix} 64 & 64 & 64 & 64 \\ 84 & 35 & -35 & -84 \\ 64 & -64 & -64 & 64 \\ 35 & -84 & 84 & -35 \end{bmatrix} \qquad \begin{bmatrix} {}_0 64 & {}_1 64 & {}_2 64 & {}_3 64 \\ {}_0 84 & {}_2 {-35} & {}_3 {-84} & {}_1 35 \\ {}_0 64 & {}_3 64 & {}_1 {-64} & {}_2 {-64} \\ {}_0 35 & {}_2 84 & {}_3 {-35} & {}_1 {-84} \end{bmatrix}$$

Original Cosine Matrix **(a)**

Sorted Cosine Matrix (nE) **(b)**

Figure 7.5: *Example cosine matrix before and after ordering*

Evaluation of the first row in Figure 7.5(b), the multiplication procedure between this row and [D] will follow the same steps as conventional matrix multiplication since the order has not been altered. The second row however has been ordered according to minimum hamming distance. So the multiplication for the row will resume with 84*35. This is because the first entry (84) in the second row has n=0 which implies that the first entry in [D] should be processed. The next entry to be processed is -35 which has n=2, hence the multiplication -35*29 will be carried out. The third computation is -84*26. Finally 35*32 is processed. The multiplication procedure carries on to the next row of the cosine matrix and follows the same technique.

7.3 Synthesis and Simulations Results

The multipliers used for the design exploration were implemented in Verilog and a test-bench for each multiplier was created. The input images that were used for the simulations were preprocessed to chop them into blocks of 8x8 matrices. Examples of the images tested can be found in Appendix C. The designs were synthesised with Synopsys's *Design Compiler* using the Alcatel 0.35μm CMOS library. Both top-down and bottom-up strategies were carried for the synthesis was bottom-up compile strategy.

The simulation environment is shown in figure 7.6. The same gate-level netlist for the multipliers was used for the simulation of the conventional implementation and the cosine ordering scheme. To also keep the level of fairness in comparison, the same netlist for the MAC unit was used in simulations for both the conventional implementation and the cosine ordering scheme.

Tables 7.2,7.3 and 7.4 illustrate the results obtained for the scheme exploration with different multiplier implementations. It can be seen from the tables that the scheme exhibits a maximum power saving of 24%.

Table 7.5 illustrates the results obtained with the different scenarios used to test the algorithm. As the table shows, power savings are achieved with all examples, with a maximum of 29% power saving for the MAC unit. The horizontal striped image provides the most power saving. This is the result of the inherently low toggling of the pixels within the image. For this image, the conventional MAC consumed a total dynamic power (*tdp*) of 6.57mW. This is listed under *conventional* for the horizontal stripes image. Cosine ordering according to 'minimum hamming distance' results are termed *hamming*. For the horizontal stripes, the *tdp* consumption for *hamming* is 4.75mW. Compared to *conventional*, a power saving of 22.25% is experienced. And finally *ascending* is used to represent cosine ordering according to ascending order. This produces a power saving of 29%. The cell internal power (*cip*) of the

MAC components is also listed in Table 7.5. The power consumption at the multiplier section of the MAC unit constitutes a larger part of the total power consumption. This affirms the theory that the multiplier is the most power hungry unit in the data path of a DCT processor. For the MAC unit design that was used, the multiplier unit consumed an average of 46% of the total cell internal power. See Figure 7.7

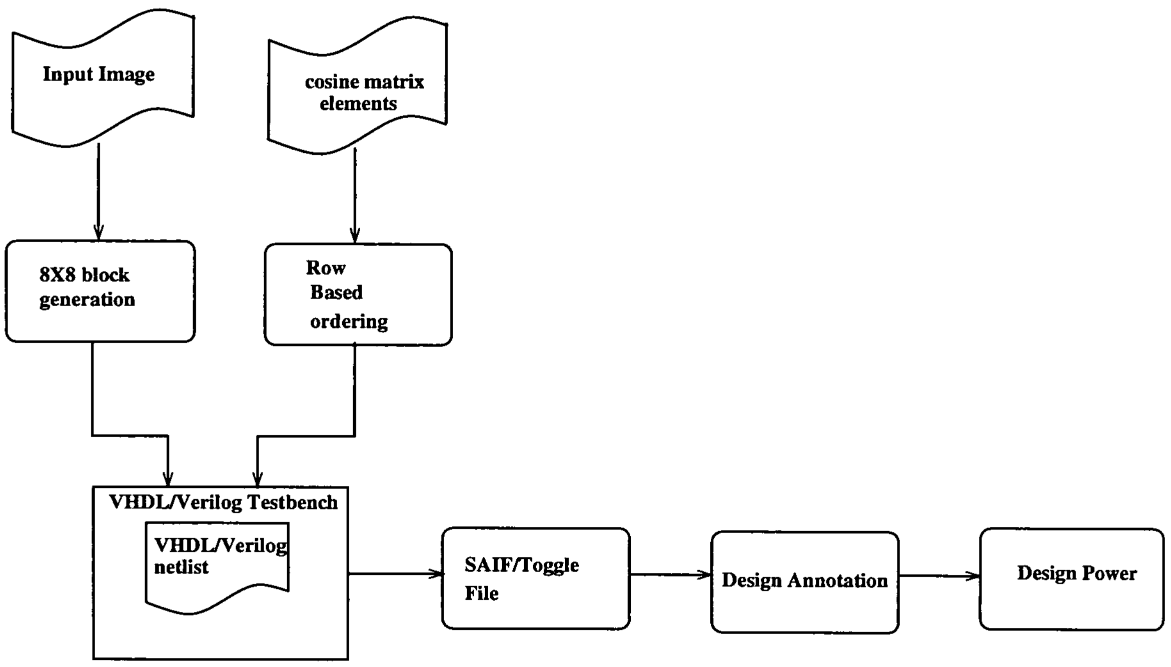


Figure 7.6: *Simulation environment*

Image	Implementation	Power(mW)			Power saving(%)
		cip	nsp	tdp	
Lena	conventional	2.33	2.74	5.07	
	cosine ordering	1.83	2.12	3.95	22.09
Wallace	conventional	2.41	2.76	5.17	
	cosine ordering	1.94	2.18	4.12	20.31
checked	conventional	2.77	3.72	6.49	
	cosine ordering	2.13	2.79	4.92	24.19

Table 7.2: *results for csa mult : ascending order*

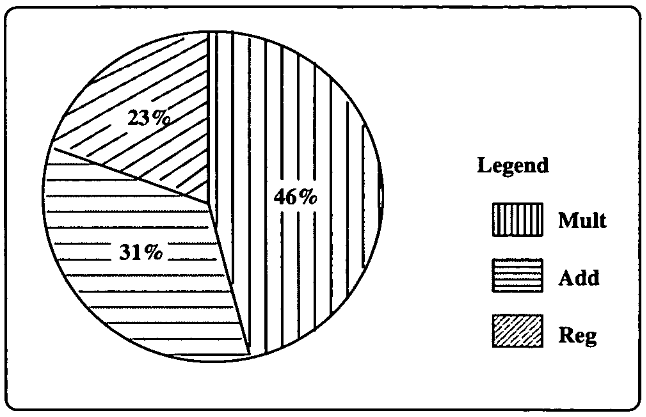


Figure 7.7: *Average percentages of cell internal power dissipation per module*

Image	Implementation	Power(mW)			Power saving(%)
		cip	nsp	tdp	
Lena	conventional	2.50	3.80	6.30	
	cosine ordering	2.25	3.39	5.64	10.48
Wallace	conventional	2.43	3.69	6.12	
	cosine ordering	2.20	3.33	5.53	9.64
checked	conventional	1.23	2.39	3.62	
	cosine ordering	1.01	1.81	2.82	22.10

Table 7.3: *results for wall mult : ascending order*

7.4 Conclusions

The chapter presented a simple but very effective low power scheme for the implementation of the DCT. The scheme takes advantage of the rows in the cosine matrix by ordering the cosine coefficients in each row according to some ordering technique such as minimum hamming distance. This minimises the toggles at the inputs of the multiplier unit in the DCT processor. Reducing the number of toggles at the inputs of the multiplier reduces power consumption in the unit. This scheme is able to save up-to 24% power dissipation compared to the conventional implementation of the DCT. Because the same implementation architecture as the conventional DCT is used, the scheme can be used in conjunction with other tech-

Image	Implementation	Power(mW)			Power saving(%)
		cip	nsp	tdp	
Lena	conventional	1.65	2.58	4.23	
	cosine ordering	1.42	2.18	3.60	14.89
Wallace	conventional	1.77	2.75	4.52	
	cosine ordering	1.59	2.40	3.99	11.73
checked	conventional	2.44	4.00	6.44	
	cosine ordering	2.14	3.32	5.46	15.22

Table 7.4: *results for nbw mult : ascending order*

Image	Implementation	Power(mW)			mult	add	reg	Power
		cip	nsp	tdp	(cip)	(cip)	(cip)	saving(%)
Lena	conventional	5.25	4.59	9.84	2.33	1.75	1.17	-
	hamming	4.41	3.75	8.16	1.84	1.49	1.08	17.07
	ascending	4.35	3.73	8.08	1.83	1.45	1.07	17.89
Wallace	conventional	5.34	4.60	9.94	2.41	1.75	1.18	-
	hamming	4.46	3.76	8.22	1.91	1.46	1.08	17.30
	ascending	4.48	3.79	8.27	1.94	1.45	1.09	16.80
checked	conventional	5.48	5.19	10.67	2.77	1.66	1.05	-
	hamming	4.57	4.12	8.76	2.21	1.38	0.99	17.90
	ascending	4.41	4.06	8.47	2.13	1.31	0.97	20.62
vertical stripes	conventional	5.35	5.08	10.43	2.73	1.59	1.03	-
	hamming	4.60	4.21	8.81	2.27	1.35	0.98	15.53
	ascending	4.40	4.05	8.45	2.16	1.28	0.96	19.01
horizontal stripes	conventional	3.54	3.03	6.57	1.56	1.06	0.92	-
	hamming	2.62	2.13	4.75	0.98	0.79	0.85	22.25
	cosine ordering	2.55	2.09	4.63	0.963	0.75	0.83	29.53

Table 7.5: *results for mac with csa mult and bk adder*

niques in literature which reduce computational complexity of the DCT. It should be noted that some overheads are possible due to the addressing of the ordered coefficients. This overhead is minimised by using a 3 bit counter instead of the 6 bit address that is reqiured for the conventional implementation.

Chapter 8
Cosine Coding Scheme

8.1 Introduction

It is a well known principle amongst the digital hardware design circles that a shift operation is more energy efficient than a multiplication operation, hence the motivation for developing a scheme which removes the burden of the multiplication operation and replacing it with a more power conscious shift operation.

In this scheme the need for a multiplication unit is eliminated by formating the data representation of the cosine elements such that the pixel values are processed using a shifter unit and an addition unit.

8.2 Data Representation

A close examination of the cosine elements reveals that in their fractional representation, they are all numbers between -0.4904 and 0.4904, see Equations 8.1 and 8.2. The proposed data representation scheme is as depicted in Figure 8.1. The MSB is used to indicate the sign of the cosine coefficient. The MSB is high for a negative coefficient and low for a positive one. The remaining bits are used to define the fractional part of the cosine coefficient. Depending on the accuracy desired, the number of bits for the fractional part can be increased as appropriate.

Therefore, wordlength affects quality of the results in this case. A reduced wordlength will have negative impact on the image quality whereas an increase in wordlength will improve the quality.

$$\mathbf{E} = \begin{bmatrix} d & d & d & d & d & d & d & d \\ a & c & e & g & -g & -e & -c & -a \\ b & f & -f & -b & -b & -f & f & b \\ c & -g & -a & -e & e & a & g & -c \\ d & -d & -d & d & d & -d & -d & d \\ e & -a & g & c & -c & -g & a & -e \\ f & -b & b & -f & -f & b & -b & f \\ g & -e & c & -a & a & -c & e & -g \end{bmatrix} \tag{8.1}$$

where

$$a = 0.4904, b = 0.4619, c = 0.4157, d = 0.3536, e = 0.2778, f = 0.1913, g = 0.0975 \tag{8.2}$$

For example if the cosine element is 0.2778, the MSB will represent the sign bit, '0', and the remaining bits will represent the fraction, 0.2778. For a 6 bit fractional part precision, 0.2778 will be represented as 010010. See table 8.1

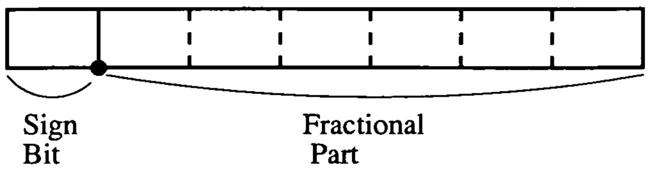


Figure 8.1: *Data word partitioning*

decimal partitioning	0	0.2778
binary representation	0	010010

Table 8.1: *data word coding example*

8.3 Shift-Only Data Manipulation

The previous section described the data representation adopted to make the data manipulation for this scheme possible. An example of the data manipulation procedure will enhance the description above.

If there are two numbers -0.2778 and 236, as the cosine element and the pixel element respectively, the multiplication procedure will be as follows: From the example given in the previous section, -0.2778 would be represented as 1010010, so the computation will be between 1010010 and 011101100. Decoding the cosine word, 1010010, should result in '1' for the sign part and 010010. This implies that there are 2 shifts to the right and 6 additions in total. The two shifts to be executed are, a) shift right by 2 places and b) shift right 5 places.

8.4 Implementation

To reduce the computational complexity of the architecture, the addition is deferred until the final addition stage, this ensures that there is only one final adder stage unit.

Decode

This module decodes the coded cosine coefficient. This is simply breaking it into the components *Sign Bit* and *Fractional Part. Sign Bit* is directed to the conversion module, while *Fractional Part* is used by the shr unit to inform the unit of the required number of shifts.

convert

This module receives a signal from the decode module. If the signal is high, the input pixel to be processed has to be converted using to negative two's compliment number.

shr unit

The shr units module, contains the right shifters. For the example shown in Figure 8.2, there are four right shifters in total, that implies that the fractional part of the coded cosine word has got a precision of four bits. Within the shifting modules there are multiplexors which determine whether a shift has to be executed or not, otherwise the the value just passes through without any shifting. If the sign bit in the coded cosine word is '1', then the input pixel is converted to negative two's compliment by the conversion module before it gets shifted.

Compression

In Figure 8.3, there is a module referred to as CSA, in this module a unit adder is used instead of a fulladder. Each unit adder adds four data inputs and one carry input. It generates one sum bit and two carry outputs[91]. The carry signals are not used for the current addition, but rather for its successor[92]

add

This is the final stage addition. The carry that was saved in the previous, compression unit, is propagated within this module to produce the sums.

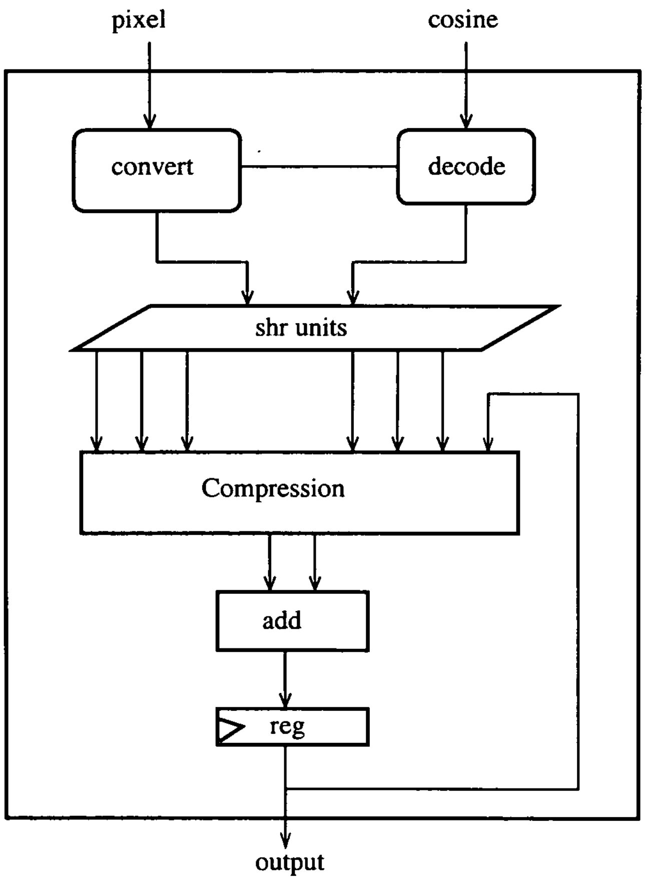


Figure 8.2: *Implementation of cosine coding scheme*

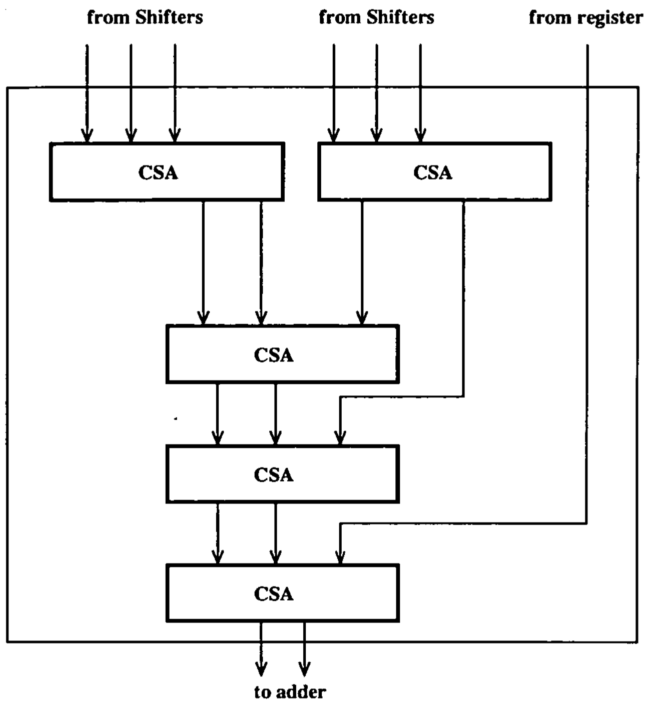


Figure 8.3: *Implementation of the compression unit*

8.5 Results and Analysis

The model was designed in VHDL and synthesised using 0.35μm technology from Alcatel. The input images that were used for the simulations were preprocessed to divide them into smaller blocks of $8x8$ matrices. Some examples of the images tested can be found in Appendix C. The designs were synthesised with Synopsys's *Design Compiler.*

The simulation environment is shown in figure 8.4. This involves pre-coding of the cosine coefficients before they are entered into a memory unit in the design. Cadence's VerilogXL was used in conjuction with *toggle count* to capture the switching activity of the design during gatelevel simulation. For the power analysis, DesignPower (See Appendix D) was used to estimate the power estimation.

Tables 8.3 illustrate the results obtained for the scheme with a number of images. The convetional implementation results are given in Table 8.2. It can be seen from table 8.3 that the scheme exhibits a maximum power saving of 41%.

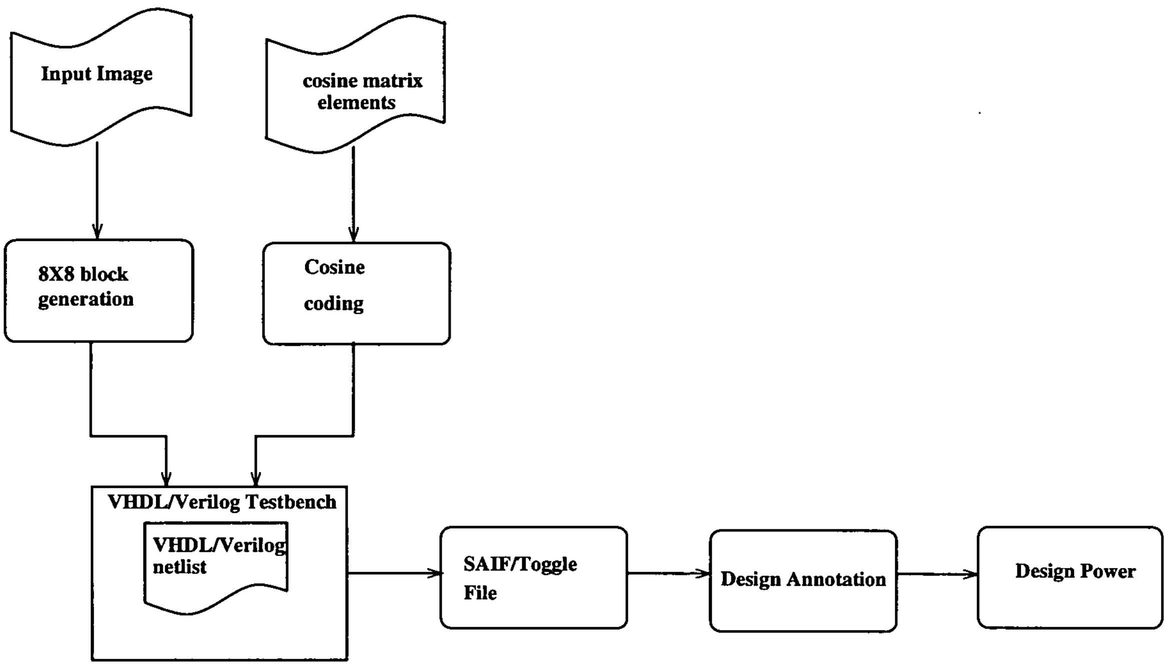


Figure 8.4: *Simulation environment*

Image	Power(mW)			mult	add	reg
	cip	nsp	tdp	(cip)	(cip)	(cip)
lena	5.25	4.59	9.84	2.33	1.75	1.17
wallace	5.34	4.60	9.94	2.41	1.75	1.18
checked	5.48	5.19	10.67	2.77	1.66	1.05
vertical stripes	5.35	5.08	10.43	2.73	1.59	1.03
horizontal stripes	3.54	3.03	6.57	1.56	1.06	0.92

Table 8.2: *results for conventional mac with csa mult and bk adder*

Image	Power(mW)			compression	add	shifters	reg	Power
	cip	nsp	tdp	(cip)	(cip)	(cip)	(cip)	saving(%)
lena	4.63	2.84	7.47	2.37	0.92	0.40	0.94	24.09
wallace	4.72	2.90	7.62	2.41	0.94	0.42	0.94	23.34
checked	5.28	3.51	8.79	2.94	0.74	0.82	0.79	17.62
vertical Stripes	4.88	3.22	8.09	2.63	0.71	0.75	0.79	22.44
horizontal Stripes	2.40	1.41	3.81	1.12	0.42	0.15	0.71	41.10

Table 8.3: *results for cosine coding scheme*

8.6 Conclusions

A new implementation scheme has been presented in this chapter which has an obvious advantage over conventional methods by using shift operators. The advantage was further enhanced by the use of compression units to defer the addition to the final stage accumulation stage. This was made possible by intoducing a new data representation scheme where the cosine coefficients are coded as sign and fractional parts. A power saving of 41% was achieved by the scheme over the conventional scheme. The accuracy of this implementation depends largely on the wordlength chosen for the fractional part of the data representation. Choosing a large fractional wordlength will result in a better image quality but reduced power savings. Hence this implementation is better suited for applications where image quality is not the main concern.

Chapter 9 Conclusions

A number of power saving schemes having been proposed and implemented in this project. These schemes focus on the multiplier section of the DCT processor since it consumes more power than the other components in the design. This chapter summarises the findings of the various techniques employed. Potential future work suggestions to further enhance this work are also presented.

9.1 Conclusions and Discussion

The literature review section, chapters 2,3 and 4, presented the basic of low power design and the basics of image compression. It also highlighted relevant important features of some well known image transforms and how they relate or compare with the DCT. It is still fact that in terms of energy compaction, the DCT performs close to the optimal KLT. Some common DCT algorithms and Architectures were presented. There are several algorithms developed for the DCT ever since its invention. This goes to show its continuing popularity amongst the image compression research and development community.

The Column-Based scheme was introduced in chapter 5. The scheme reduces power through the utilisation of shift operations, where possible, together with sequencing coefficient and pixel multiplications in an order dictated by the bit correlation of their successive values.

This results in up-to 50% power reduction with a number of standard benchmark images. The investigations revealed that that the scheme provides a generic framework for power saving in that significant power savings is achieved with a range of standard image types. These include those with sharp, continuous, and low pixel variation frequency. The scheme can easily be extended to the implementation of the 2D DCT.

Chapter 6 presented an implementation of the low power scheme for DCT processing introduced in chapter 5. The proposed architecture of the data-path was implemented in verilog. The scheme reduces power through the utilisation of sequencing coefficient and pixel multiplications in an order dictated by the bit correlation of their successive values. This results in up-to 56% power reduction within the multiplier unit for a number of standard benchmark images.

Since the column-based scheme requires saving of partial summations prior to outputting the DCT coefficients, a set of 8 registers is required compared to the conventional implementation which requires only one. This register bank has a significant overhead on the overall power saving such that the advantage revealed within the multiplier section is diminished. Theoretically, only two registers/latches are expected to be switching when the scheme is in operation. One read register and one write register, so the expected power overhead should be at least twice the consumption of the single register in the conventional implementation. This, however, can be avoided in cases where a 2D DCT implementation is required, the intermediate memory, for transposition can be used to temporarily store this partial sums and no extra circuitry will be required.

Chapter 7 presented a very effective low power scheme for the implementation of the DCT. This scheme is able to save up-to 24% power dissipation compared to the conventional implementation of the DCT. Because the same implementation architecture as the conventional DCT is used, the scheme can be used in conjunction with other techniques in literature which reduce computational complexity of the DCT. Although this scheme does not exhibit as much power saving within the multiplier unit as the Column-based scheme, it does not have any overheads.

Another implementation scheme has been presented in chapter 8 which has an obvious advantage over conventional methods by using shift operators. The advantage was further enhanced by the use of compression units to defer the addition to the final stage accumulation stage. A power saving of 41% was achieved by the scheme over the conventional scheme. This scheme can be used in conjunction with any of the two order based schemes to further enhance the power saving of the DCT processor.

9.2 Novel Outcomes of the Research

Three new schemes for reducing the switched capacitance, within the multiplier section, of the DCT processor were successfully developed. The first two schemes, column-based and order-based, reduce switched capacitance through manipulation and exploitation of correlation in pixel and cosine coefficients during the computation of the DCT coefficients. The techniques are generic and can be extended to include other applications where matrix multiplication is required. The third scheme proposes coding cosine coefficients such that only shift operations are used to process the DCT computation.

The *column-based* scheme reduces switched capacitance through manipulation and exploitation of correlation in pixel and cosine coefficients during the computation of the DCT coefficients. The manipulation is of the form of ordering the cosine coefficients per column, according to some ordering technique such as, ascending order or minimum hamming distance and processing the matrix multiplication using a column-column approach instead of the usual row-column approach. Several low power design techniques were used in implementing this design. Some of the techniques used include clock gating and the use of latch-based storage. A power saving of up-to 50% within the multiplier section was achieved with this implementation.

The *order-based* scheme takes advantage of the rows in the cosine matrix. Ordering the cosine coefficients in each row according to some ordering technique such as, ascending order or minimum hamming distance, minimises the toggles at the inputs of the multiplier unit in the DCT processor. The above techniques are generic and can be extended to include other applications where matrix multiplication is required. This scheme is able to save up-to 24% power dissipation compared to the conventional implementation of the DCT.

The third scheme proposes coding cosine coefficients such that only shift operations are used to process the DCT computation. In this scheme the need for a standard multiplication unit is eliminated by formating the data representation of the cosine elements such that the pixel values are processed using a shifter unit and an addition unit. With this scheme, a power saving of up-to 41% can be achieved.

9.3 Future Work

There are a few areas in which the work in this thesis can be further investigated. Some are listed below.

1. For the Column-Based scheme, an implementation of the 2D DCT is needed. This will enable the investigation of the proposed solution to the overhead incurred by the scheme. The proposed solution is to reuse the matrix transposition memory as an intermediate storage, therefore eliminating extra circuitry (latch bank).

2. For an increased power efficient DCT engine, a combination of the Order-Based and Cosine-coding schemes should be considered. This should result in the advantages of both schemes being experienced by the combined unit. Some overhead is expected due to the increase of the *cosine word* wordlength.

3. A comparison of this implementation technique to other techniques such as ROM-Based multipliers and digit-serial implementations should be considered.

4. The power estimation methodology used was pretty accurate, it was based on the gate-level netlist after synthesis. A more accurate power reading can be achieved after circuit layout and back annotating the resulting parasitics. This will offer a reading that is highly close to the actual value. This suggests that a layout of circuit should be carried out.

5. An implementation of the IDCT using the proposed schemes can be considered as an enhancement.

6. A complete DCT engine Intellectual Property (IP) should be considered. This involves all the components in the proposed architectures of the schemes. It would be an advantage if the IP is reconfigurable and/or programmable. The parameters could include input wordlengths (both pixel word and cosine coefficient word), output wordlength, data representations (e.g two's complement or sign-magnitude), internal data accuracy(wordlength, truncation, rounding).

7. The IP developed can be incorporated in a working environment for either JPEG or MPEG standards and its performance observed relative to other implementations.

8. The multiplication schemes can be used in other applications where matrix multiplication is required

Appendix A
Image Compression Standards

A.0.1 JPEG

JPEG(Joint Photographic Experts Group) is a standardized image compression mechanism. The 'Group' reflects the original committee that wrote the standard.

JPEG is designed for compressing either full-colour or gray-scale images of natural, real-world scenes. JPEG is designed to exploit known limitations of the human eye. For example, the fact that small colour details aren't perceived as well as small details of light-and-dark. Thus, JPEG is intended for compressing images that will be looked at with a human eye, not for machine analysis[93–95].

A useful property of JPEG is that the degree of lossiness can be varied by adjusting compression parameters. This means that during image capture, trade off file size against output image quality can be made. Image files can be made extremely small if high image quality is of no importance. This is useful for applications like indexing image archives and view finders. On the other hand, if output image quality is an issue, high compression ratios can be sacrifised for satisfactory image quality [93, 95].

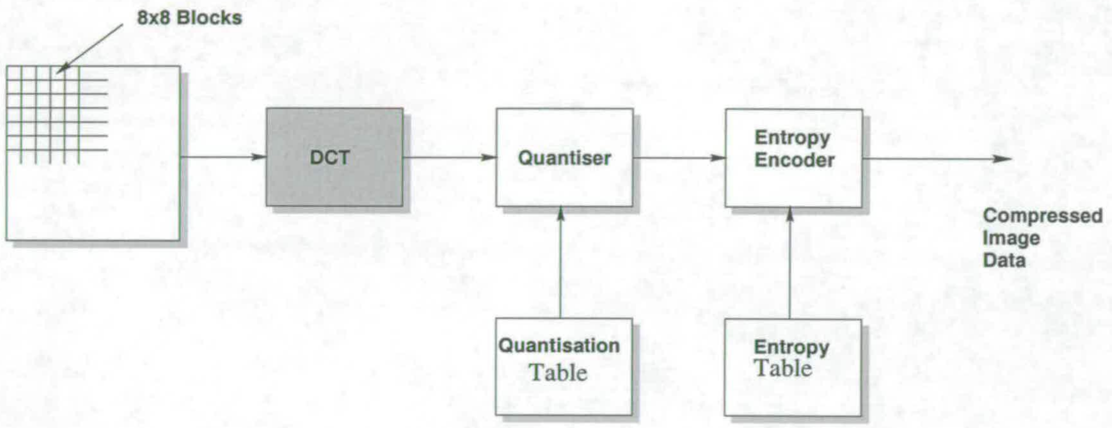


Figure A.1: *Block diagram of JPEG Compression*

Why use JPEG

There are two fundamental reasons to use JPEG

1. to make the image files smaller
2. to store 24-bit-per-pixel colour data instead of 8-bit-per-pixel data

Making image files smaller is an advantage for transmitting files across networks and for archiving libraries of images.

If the viewing software does not support JPEG directly, JPEG has to be converted to some other format for viewing or manipulating images. Even with a JPEG-capable viewer, it takes longer to decode and view a JPEG image than to view an image of a simpler format such as GIF. Hence, using JPEG is essentially a tradeoff between time and space.

The second fundamental advantage of JPEG is that it can store full colour information: 24 bits/pixel (16 million colours). The other image format, GIF, widely used on Usenet, can only store 8 bits/pixel (256 or fewer colours) [95].

Baseline JPEG (Lossy)

Steps involved in the baseline compression algorithm[93]:

1. Firstly an image is transformed into a suitable colour space. For greyscale this step in unnecessary, but for colour images RGB is transformed into a luminance and chromin-

ance color space (YCbCr, YUV, etc). The luminance component represents gray scale and the other two axes are colour information. The reason for changing colour spaces is that more information can be lost in the chrominance components at acceptable costs than in the luminance component. The human eye is not as sensitive to high-frequency colour information as it is to high-frequency luminance.

2. Downsample each component by averaging together groups of pixels. This is an optional step. The luminance component is left at original resolution, while the colour components are usually reduced 2:1 horizontally and either 2:1 or 1:1 vertically. This step immediately reduces the data volume by one-half or one-third, while having virtually no impact on perceived quality.

3. Group the pixel values for each component into blocks of $8x8$. Transform each 8x8 block through a discrete cosine transform (DCT). The motivation for doing this is that high-frequency information can be thrown away without affecting low-frequency information.

4. For each block, divide each of the 64 DCT coefficients by a separate *quantization coefficient*, and round the results to integers. This is the fundamental *Lossy* step. A quantisation coefficient of 1 loses no information, while larger ones lose successively more. The higher frequencies are usually reduced much more than the lower.

5. Encode the resulting coefficients using Huffman. This step is lossless, therefore it does not affect image quality.

6. Attach appropriate headers, etc, and output the result. In an *interchange* JPEG file, all

of the compression parameters are included in the headers so that the decompressor can be able to reverse the process.

The decompression algorithm reverses the above process, and normally adds some smoothing steps to reduce pixel-to-pixel discontinuities.

A.0.2 MPEG

The MPEG standards are generic and universal in the sense that they specify a compressed bitstream syntax. There are three main parts of the MPEG specifications. Namely systems, video, and audio components. The video part defines the syntax and semantics of the compressed video bitstreams. The audio defines the same thing for audio bitstream, while the system addresses the multiplexing of video and audio streams into a single stream with all necessary timing information[19, 94].

Standard	Bit rate	Function	Screen size
MPEG-1	1.14 to 3.0 mbps	delivery of video for a CD-ROM	352x240 pels at 30 for NTSC
MPEG-2	6 to 8 mbps	broadcast quality compressed video	720x480 pels at 60 for NTSC and 720x576 pels at 50 for HDTV
MPEG-4	64 kbits/s	low bit rate video phones, interactive databases, interactive newspapers etc	Under development

Table A.1: *Typical MPEG standards specifications*

At the top level of the hierachy (Figure A.2), the video bitstream consists of video sequences. MPEG-1 allows only progressive sequences, while MPEG-2 allows both progressive and

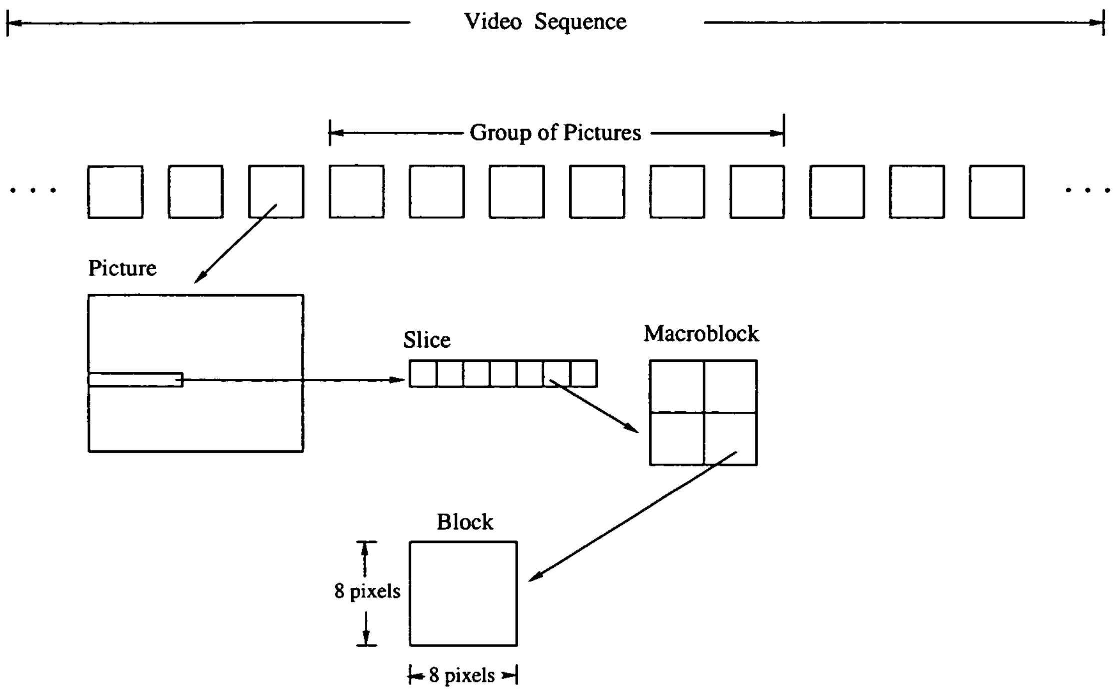


Figure A.2: *Hierachy of video signals*

interlaced sequences. Each video sequence contains a variable number of Group of Picures (GOP). A GOP contains a variable number of pictures (P). A picture can either be a frame picture or a field picture. In a frame picture, the fields (YUV) are coded together to form a frame, while a field picture is a coded version of the individual field. In MPEG video frames are broken down into $8x8$ pixels regions called blocks. Four of these blocks can be put together to create a 16x16 macroblock. The macroblocks are then grouped together into runs of macroblocks called slices. The slice structure allows the receiver to resynchronise at the beginning of a slice in case of data corruption because each slice begins with a unique header [94, 96].

Inside each GOP, two types of coding are permited, Intra frame and inter frame coding.

The intra coding of frame proceeds without any references to other frames exploiting only its spatial redundancy. The intra coded frames (I-frames) provides the access point to the coded sequence. Inter frame coding uses motion compensation prediction from previous or subsequent frames in order to exploit both spatial and temporal redundacny. In MPEG, two kinds of inter coded frames are distinguished, P-frames and B-frames. P-frames are motion compensated from the a past I-frame whereas B-frame require both past and future reference frames for mption compensation. Since B-frames uses both future and past frames for prediction, the highest degree of cpmpression is obtained for B-frames but they cannot be used as a reference for prediction(Figure A.3)[96].

The MPEG derives its maximum compression from P and B-frames. This is done through a technique named motion compensation based prediction. This exploits temporal redundancy. Since framce are closely related, it is assumed that a current picture can be modelled as a transition of the picture at a previous time. It is possible then to acurately predict the data of one frame based on the data of a previous frame. The encoder searches the previous frame in half pixel increments for other macroblock locations that are close match to the information that is contained in the current macroblock. The displacement in the vertical and horizontal directions of the match macroblock from the cosited macroblock are called motion vectors. If a matching block is found in the search region the motion vectors are encoded. If no matching is found in the neighbouring region, the macroblock is intra-coded and the DCT coefficients are encoded [96].

The intra-frame coding involves the same process described in the JPEG standard.

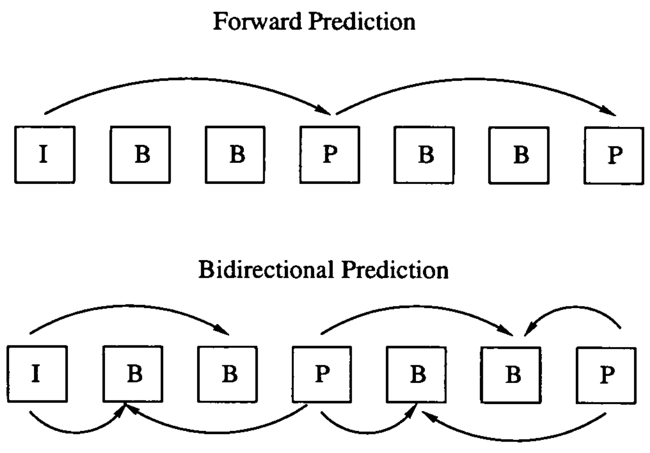


Figure A.3: *Pand B frame predictions*

Appendix B
Cosine Coefficients

$$\mathbf{E} = \begin{bmatrix} d & d & d & d & d & d & d & d \\ a & c & e & g & -g & -e & -c & -a \\ b & f & -f & -b & -b & -f & f & b \\ c & -g & -a & -e & e & a & g & -c \\ d & -d & -d & d & d & -d & -d & d \\ e & -a & g & c & -c & -g & a & -e \\ f & -b & b & -f & -f & b & -b & f \\ g & -e & c & -a & a & -c & e & -g \end{bmatrix} \tag{B.1}$$

where

$$a = cos\frac{\pi}{16}, b = cos\frac{\pi}{8}, c = cos\frac{3\pi}{16}, d = cos\frac{\pi}{4}, e = cos\frac{3\pi}{16}, f = cos\frac{\pi}{8}, g = cos\frac{\pi}{16} \tag{B.2}$$

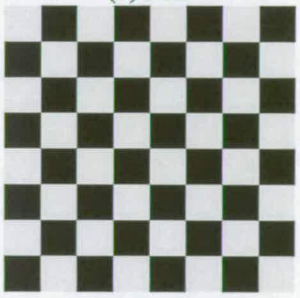
Appendix C
Test Images

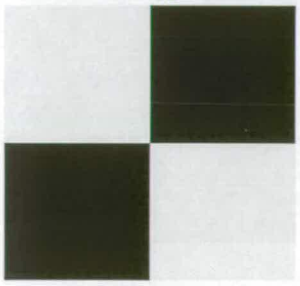
Some of the test images that were used for the simulations.

(a) Lena

(b) Man

(c) Peppers


(d) Checked


(e) Blocks

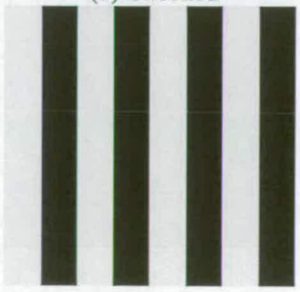

(f) Vertical

Figure C.1: *Some of the tested images*

Appendix D
Synopsys DesignPower Analysis

After the RTL model has been captured, the next level down the deisgn hierachy is obtaining the gatelevel netlist through synthesis. When the netlist is finalized, DesignPower can use the RTL simulation switching activity information to accurately determine a power estimation of the design. DesignPower's accuracy can be further enhanced through the use of switching activity generated from a gate-level simulation. The gate-level simulation supports power dissipation due to glitching and can use the state-dependent and path-dependent library power models [3, 97].

D.1 Power Analysis Techniques

The power of a circuit can be determined by either using an RTL simulation approach or a netlist simulation-based approach. RTL simulation-based power analysis can be used before synthesis and is based on switching activity captured from RTL simulation. Gate-level simulation-based power analysis combines both state-dependent and path-dependent library power models with state-dependent and path-dependent switching activity. This yields an improved accuracy. However, the complexity of this analysis depends on the number of input stimuli applied to the circuit inputs, as well as the size of the circuit being analyzed. Hence, for large circuits with a large number of input stimuli, gate-level simulation-based power estimation may require substantial computing resources [97].

D.2 Simulation-Based Power Analysis

D.2.1 RTL Simultation Based Power Analysis

For the RTL simulation based power analysis approach, the static probability and transition density of each synthesis invariant element is captured from RTL simulation. The captured information is then propagated to all intermediate nodes in the circuit using an internal propagation engine. The internal propagation engine does not generate state-dependent and path-dependent switching activity, so the library power models are not used to their full extent [97].

The accuracy can also affected by the delay model chosen for the analysis. Analysis techniques have been proposed using zero delay, unit delay, and full timing-delay models. With zero-delay model, glitching is ignored while unit-delay and full timing-delay include the impact of glitching on the average switching activity. The zero-delay model assumption simplifies the computation of power estimate, therefore, it used in the switching activity propagation engines [97].

D.2.2 Gate-Level Simulation-Based Power Analysis

Gate-level (or netlist) simulation-based power analysis is based upon switching activity data captured by observing the transitions made by every net in the circuit during a gate-level simulation run. Some of the commercially available netlist simulators can also count the transitions based on pre-defined states of various nodes in the design. For example, they can record the switching activity on the clock pin of a RAM when $RW = 1$ separated from those when $RW = 0$. The switching information derived from the simulation is back-annotated and combined with information from the library to compute the design's total power dissipation. When using good quality library power models with good simulation models for interconnect, glitching, and gate delays, the gate-level simulation-based technique can be very accurate. The penalty for high accuracy, however, is lengthy gate-level simulation [97].

Appendix E
Program Listings

E.1 Pre-processing Programs

```
#!/usr/local/bin/perl
# This program arranges the vectors in array vector starting with the one
# with minimum number of ones first, and then does a sort on minimum hamming
# distance.


print stderr "Enter nput filename --> input :-  ";
$input = <stdin>;
chop $input;

print stderr "Enter output filename --> hamming :-  ";
$output_ham = <stdin>;
chop $output_ham;

open(FIN,"< $input");
open(FOUT, ">$output_ham");
open(FOUT_debug,">debug.out");


@vector = <FIN>;

for ($i = 0 ; $i < 8 ; $i++)
  {

     for ($k = 0 ; $k < 8 ; $k++)
       {
        $m = ($k+($i*8));
        @vector_to_sort[$k]= @vector[$m] ;
        }
     @vectors = sort {($a=~s/1/1/g)<=>($b=~s/1/1/g)} @vector_to_sort;
     while($#vectors+1)
       {
        push @sorted, shift(@vectors);
       @vectors = sort by_hamming @vectors;
       }
       print FOUT @sorted;
}


sub by_hamming()
{
 (join "",sort split //,@sorted[-1]^$a) cmp (join "",sort split //,@sorted[-1]^$b);
}
```

```
#!/usr/local/bin/perl
# This program arranges the vectors in array vector in ascending order
print stderr "Enter filename to be sorted :";
$input = <stdin>;
chop $input;

print stderr "Enter output filename --> ascending :-  ";
$output_asc = <stdin>;
chop $output_asc;


open(FIN,"<$input");
open(FOUT_asc, ">$output_asc");
#open(FOUT_min, ">$output_min");

@vector = <FIN>;

#@vector_sort = sort { $a <=> $b } @vector; #this sorts the entire array @vector
#print @vector;
#print FOUT_asc "@vector_sort";

for ($i = 0 ; $i < 8 ; $i++)
   {
  for ($k = 0 ; $k < 8 ; $k++)
   {
     $m = ($k+($i*8));

       @vector_to_sort[$k]= @vector[$m]

   }
   @vector_sorted = sort { $a <=> $b } @vector_to_sort;
   print FOUT_asc @vector_sorted;
}
```

E.2 Some Design Files Examples

Order Based Implementation

```
// Filename        : cb_mac_sim.v
// Description     : column based mac unit for simulation
// Author          : Shedden Masupe

'define pixel_width 9
'define counter_width 3
'timescale 1ns / 1ns
//
// Lower hierachy modules
//


module mult_csa(a_in, b_in, mult_out);

parameter width = 'pixel_width;

output [(width*2)-1 : 0] mult_out;
input [width-1 : 0]      a_in;
input [width-1 : 0]      b_in;

assign  mult_out = a_in * b_in;

endmodule //mutiplier

module adder_bk(a_in, b_in, adder_out);

parameter width = 'pixel_width;

output [(width*2)-1 : 0] adder_out;
input  [(width*2)-1 : 0] a_in;
input  [(width*2)-1 : 0] b_in;

assign  adder_out = a_in + b_in;

endmodule //adder

module mux(a_in, sel, mux_out);
   parameter width = 'pixel_width;

   input [(width*2)-1 : 0] a_in;
   input     sel;
   output [(width*2)-1 : 0] mux_out;

   reg [(width*2)-1 : 0]    mux_out;

   always @(a_in or sel)
     if (sel)
       mux_out = 0;
     else
       mux_out = a_in;

endmodule // mux

module out_reg(a_in, en, clk, reset, reg_out);

parameter width = 'pixel_width;

input [(width*2)-1 : 0] a_in;
input clk, reset, en;
```

```
output [(width*2)-1 : 0] reg_out;

reg [(width*2)-1 : 0] reg_out;

  always @(posedge clk or negedge reset)
   if(!reset)
     reg_out = 0;
   else
     if(en)
       reg_out = a_in;
     else
       reg_out = 0;
endmodule //register


//
//  Top module
//

module cos_order(a_in, b_in, en, clk, reset, mac_out);

parameter width = 'pixel_width;
parameter addr_width = 'counter_width;

output [(width*2)-1 : 0] mac_out;

input [width-1 : 0] a_in;
input [width-1 : 0] b_in;
input clk, reset,en;

// signals
wire [(width*2)-1 : 0] sum;
wire [(width*2)-1 : 0] mux_out;
wire [(width*2)-1 : 0] product;
wire c_in, c_out;
wire clk, reset;

// multiplier used here is behavioural, basically (*)

mult_csa mult_1(.a_in(a_in), .b_in(b_in), .mult_out(product));

// the adder is behavioural as well (+)

adder_bk add_1(.a_in(mux_out), .b_in(product),  .adder_out(sum));

//multiplexor
 mux mux_1(.a_in(sum), .sel(en), .mux_out(mux_out));
//output register instantiation

out_reg reg_1(.a_in(sum),.en(en), .clk(clk), .reset(reset), .reg_out(mac_out));

endmodule
```

```
// Filename        : cb_mac_syn.v
// Description     : column based mac unit for synthesis
// Author          : Shedden Masupe

'define pixel_width 9
'define counter_width 3

/*
 * Lower hierachy modules
 */


module mult_csa(a_in, b_in, mult_out);

parameter a_width = 'pixel_width;
parameter b_width = 'pixel_width;

output [(a_width + b_width)-1 : 0] mult_out;
input  [a_width-1 : 0] a_in;
input  [b_width-1 : 0] b_in;

wire tc_control = 1;

 //synopsys dc_script_begin
   //  set_implementation csa csa1
   //synopsys dc_script_end
DW02_mult #(a_width, b_width) csa1(.A(a_in), .B(b_in), .TC(tc_control), .PRODUCT(mult_out));

endmodule //mult_csa

module adder_bk(a_in, b_in, adder_out );

   parameter  width = 'pixel_width*2;

   input [width-1:0] a_in;
   input [width-1:0] b_in;

   output [width-1:0] adder_out;

   wire      c_out;
   wire      c_in = 0;
   //synopsys dc_script_begin
   //  set_implementation bk bk1
   //synopsys dc_script_end
DW01_add #(width) bk1(.A(a_in), .B(b_in), .CI(c_in), .SUM(adder_out), .CO(c_out));
endmodule //adder_bk

module mux(a_in, sel, mux_out);
   parameter width = 'pixel_width;

   input [(width*2)-1 : 0] a_in;
   input     sel;
   output [(width*2)-1 : 0] mux_out;

   reg [(width*2)-1 : 0]    mux_out;

   always @(a_in or sel)
     if (sel)
       mux_out = 0;
     else
       mux_out = a_in;

endmodule // mux
```

```
module out_reg(a_in, en, clk, reset, reg_out);

parameter width = `pixel_width;

input [(width*2)-1 : 0] a_in;
input clk, reset, en;
output [(width*2)-1 : 0] reg_out;

reg [(width*2)-1 : 0] reg_out;

  always @(posedge clk or negedge reset)
   if(!reset)
     reg_out = 0;
   else
     if(en)
       reg_out = a_in;
     else
       reg_out = 0;
endmodule //register


/*
 * Top module
 */

module cos_order(a_in, b_in, en, clk, reset, mac_out);

parameter width = `pixel_width;
parameter addr_width = `counter_width;

output [(width*2)-1 : 0] mac_out;

input [width-1 : 0] a_in;
input [width-1 : 0] b_in;
input clk, reset, en;

// signals
wire [(width*2)-1 : 0] sum;
wire [(width*2)-1 : 0] mux_out;
wire [(width*2)-1 : 0] product;

wire clk, reset;

// multiplier used here is behavioural, basically (*)

mult_csa mult_1(.a_in(a_in), .b_in(b_in), .mult_out(product));

// the adder is behavioural as well (+)

adder_bk add_1(.a_in(mux_out), .b_in(product),  .adder_out(sum));

//multiplexor
 mux mux_1(.a_in(sum), .sel(en), .mux_out(mux_out));

//output register instantiation

out_reg reg_1(.a_in(sum),.en(en), .clk(clk), .reset(reset), .reg_out(mac_out));

endmodule
```

```
// Filename        : cos_order_tb.v
// Description     : test bench for column based mac unit
// Author          : Shedden Masupe


'timescale 1ns / 1ns
module cos_ord_sim_test ;


parameter pixel_width = 9;
parameter image_order = 64;
parameter matrix_order = 8;
parameter addr_width = 3;
parameter freq = 20;


reg  [pixel_width-1 : 0]  a_in, b_in;
reg clk, reset, en;
reg [pixel_width+2 : 0] cos_array[0:image_order];
reg [pixel_width-1 : 0] pel_array[0:image_order];
reg [addr_width-1:0] loc, d;
reg [pixel_width+2 : 0] temp ;
wire  [(pixel_width*2)-1 : 0] mac_out;
wire tc_control;

integer n, a, b;
integer write_file;

//instantiate dut
cos_order dut(.a_in(a_in),.b_in(b_in),.en(en),.clk(clk),.reset(reset),.mac_out(mac_out));

//initialisation
initial
 begin
  reset = 0;
  en = 0;
  a_in = 0;
  b_in = 0;
  d = 6;

 #50 reset = 1;
// #15 write_addr = 7;
//     read_addr =0;
end

//--------------------------------------------------
//read from file    .
//--------------------------------------------------
initial
  begin
    wait(reset);

    $readmemb("./input/ascending_128.dat",cos_array);
    $readmemb("./input/v_stripes.dat",pel_array);

//    $readmemb("./input/ones.dat",cos_array);
//    $readmemb("./input/twos.dat",pel_array);

    write_file = $fopen("./output/cos_ord_out.dat");
//    $set_toggle_region (dut);
//    $toggle_start;
    $fwrite(write_file, "                  time en  a_in   b_in  mult_out  adder_out  mac_out\n");

  //read in vectors
   for(n = 0; n<matrix_order; n= n+1)//counter for pel cols
```

```
     begin
       for(b=0; b<matrix_order; b=b+1)//counter for cosine cols and pel rows
         begin

           for(a=0; a< matrix_order; a=a+1) //do the cosine rows
             begin
               @(posedge clk);

       temp =  cos_array[a+(b* matrix_order)];
       loc = temp[11:9];
               b_in =  temp[8:0];
               a_in = pel_array[loc +(n* matrix_order)];
               $fwrite(write_file, $time,, "  %d %d    %d %d    %d    %d\n",  en, a_in, b_in, dut.mult_1
               //$monitor($time,,"%d  %d  %d   %d",d, a_in, b_in, mac_out);
             end // counter a
         end // counter b
     end //counter n
//   $toggle_stop;
//   $toggle_report("./output/cos_ord_norm.saif", 1.0e-9, "cos_ord_sim_test.dut");
   $finish;
  end
//counter
always @ (negedge clk)
begin
 wait(reset);
 d = d + 1;
   if (d==0)
     en = 1;
   else
     en = 0;
end

// generate clock
initial
     clk = 0;
     always #(freq/2) clk = !clk;

endmodule // cos_ord_sim_test
```

Column Based Implementation

```
// Filename        : cb_mac_sim.v
// Description     : column based mac unit for simulation
// Author          : Shedden Masupe


'define pixel_width 8
'define counter_width 3

//
// Lower hierachy modules
//


module mult(a_in, b_in, mult_out);

parameter width = 'pixel_width;

output [(width*2)-1 : 0] mult_out;
input [width-1 : 0]      a_in;
input [width-1 : 0]      b_in;

assign  mult_out = a_in * b_in;

endmodule //mutiplier

module adder(a_in, b_in, adder_out);

parameter width = 'pixel_width;

output [(width*2)-1 : 0] adder_out;
input  [(width*2)-1 : 0] a_in;
input  [(width*2)-1 : 0] b_in;

assign  adder_out = a_in + b_in;

endmodule //adder

module latch(a_in, enable, reset, latch_out);

parameter width = 'pixel_width;

input [(width*2)-1 : 0] a_in;
input enable, reset;
output [(width*2)-1 : 0] latch_out;

reg [(width*2)-1 : 0] latch_out;

reg q;
  always @(a_in or enable or reset)
   if(!reset)
     latch_out = 0;
   else
     if (enable)
       latch_out = a_in;
endmodule //latch


module latch_bank(a_in, clk, en, reset, write_addr, read_addr, latch_out);

parameter width = 'pixel_width;
parameter addr_width = 3;

input [(width*2)-1 : 0] a_in;
input [addr_width-1:0] write_addr, read_addr;
input reset, en, clk;

output [(width*2)-1 : 0] latch_out;

reg [(width*2)-1 : 0] latch_out;

wire enable_0,
     enable_1,
     enable_2,
     enable_3,
     enable_4,
     enable_5,
     enable_6,
     enable_7;

reg  in,
     in_0,
     in_1,
     in_2,
     in_3,
     in_4,
     in_5,
     in_6,
     in_7; //signals for the latch enable gating signals

wire clk_n;


wire [(width*2)-1 : 0] out_0,
                       out_1,
                       out_2,
                       out_3,
                       out_4,
                       out_5,
                       out_6,
                       out_7;

//demultiplexing the input to the latch_bank
always @(write_addr or in)
   begin
     if(write_addr == 3'b000)
       in_0 = en;
     else in_0 = 0;

     if(write_addr == 3'b001)
       in_1 = en;
     else in_1 = 0;

     if(write_addr == 3'b010)
       in_2 = en;
     else in_2 = 0;

     if(write_addr == 3'b011)
       in_3 = en;
     else in_3 = 0;

     if(write_addr == 3'b100)
       in_4 = en;
     else in_4 = 0;

     if(write_addr == 3'b101)
       in_5 = en;
     else in_5 = 0;

     if(write_addr == 3'b110)
       in_6 = en;
     else in_6 = 0;

     if(write_addr == 3'b111)
       in_7 = en;
     else in_7 = 0;
   end //always

//Gating the clk signal with enable
not inv_clk(clk_n, clk);
and act_en0(enable_0, clk_n, in_0);
and act_en1(enable_1, clk_n, in_1);
and act_en2(enable_2, clk_n, in_2);
and act_en3(enable_3, clk_n, in_3);
and act_en4(enable_4, clk_n, in_4);
and act_en5(enable_5, clk_n, in_5);
and act_en6(enable_6, clk_n, in_6);
and act_en7(enable_7, clk_n, in_7);

//end of gated clock

//instantiation of the latches required
latch L0  (.a_in(a_in) , .enable(enable_0), .reset(reset),.latch_out(out_0));
latch L1  (.a_in(a_in) , .enable(enable_1), .reset(reset),.latch_out(out_1));
latch L2  (.a_in(a_in) , .enable(enable_2), .reset(reset),.latch_out(out_2));
latch L3  (.a_in(a_in) , .enable(enable_3), .reset(reset),.latch_out(out_3));
latch L4  (.a_in(a_in) , .enable(enable_4), .reset(reset),.latch_out(out_4));
latch L5  (.a_in(a_in) , .enable(enable_5), .reset(reset),.latch_out(out_5));
latch L6  (.a_in(a_in) , .enable(enable_6), .reset(reset),.latch_out(out_6));
latch L7  (.a_in(a_in) , .enable(enable_7), .reset(reset),.latch_out(out_7));

//multiplexing the output of the latch_bank
always @(read_addr or out_0 or out_1 or out_2 or out_3 or out_4 or  out_5 or out_6 or out_7)
  begin
    case(read_addr)
      3'b000 : latch_out = out_0;
      3'b001 : latch_out = out_1;
      3'b010 : latch_out = out_2;
      3'b011 : latch_out = out_3;
      3'b100 : latch_out = out_4;
      3'b101 : latch_out = out_5;
      3'b110 : latch_out = out_6;
      3'b111 : latch_out = out_7;
      default : latch_out = 0;
    endcase
  end //always
endmodule //latch_bank


//
//  Top module
//

module mac(a_in, b_in, write_addr, read_addr, en, clk, reset, mac_out);

parameter width = `pixel_width;
parameter addr_width = `counter_width;

output [(width*2)-1 : 0] mac_out;

input [addr_width-1 : 0] write_addr;
input [addr_width-1 : 0] read_addr;
input [width-1 : 0] a_in;
input [width-1 : 0] b_in;
input clk, reset, en;

// signals
```

```
wire [(width*2)-1 : 0] sum;
wire [(width*2)-1 : 0] product;
wire c_in, c_out;
wire clk, reset;

// multiplier used here is behavioural, basically (*)

mult mult_1(.a_in(a_in), .b_in(b_in), .mult_out(product));

// the adder is behavioural as well (+)

adder add_1(.a_in(mac_out), .b_in(product),  .adder_out(sum));

//the latch bank has been coded at RTL level

latch_bank latches(.a_in(sum), .clk(clk), .en(en),.reset(reset), .write_addr(write_addr), .read_addr(rea

endmodule
```

```
// Filename        : cb_mac_syn.v
// Description     : column based mac unit for synthesis
// Author          : Shedden Masupe

'define pixel_width 8
'define counter_width 3

/*
 * Lower hierachy modules
 */


module mult_csa(a_in, b_in, mult_out);

parameter a_width = 'pixel_width;
parameter b_width = 'pixel_width;

output [(a_width + b_width)-1 : 0] mult_out;
input  [a_width-1 : 0] a_in;
input  [b_width-1 : 0] b_in;
wire tc_control = 1;

 //synopsys dc_script_begin
   //  set_implementation csa csa1
   //synopsys dc_script_end
DW02_mult #(a_width, b_width) csa1(.A(a_in), .B(b_in), .TC(tc_control), .PRODUCT(mult_out));


endmodule //mult_csa

module adder_bk(a_in, b_in, adder_out);

   parameter  width =  'pixel_width*2;

   input [width-1:0]  a_in, b_in;

   output [width-1:0] adder_out;
   wire      c_out;
   wire      c_in = 0;
   //synopsys dc_script_begin
   //  set_implementation bk bk1
   //synopsys dc_script_end
DW01_add #(width) bk1(.A(a_in), .B(b_in), .CI(c_in), .SUM(adder_out), .CO(c_out));
endmodule //adder_bk

module latch(a_in, enable, reset, latch_out);

parameter width = 'pixel_width;

input [(width*2)-1 : 0] a_in;
input enable, reset;
output [(width*2)-1 : 0] latch_out;

reg [(width*2)-1 : 0] latch_out;
  always @(a_in or enable or reset)
   if(!reset)
     latch_out = 0;
   else
     if (enable)
       latch_out = a_in;
endmodule //latch


module latch_bank(a_in, clk, en, reset, write_addr, read_addr, latch_out);
```

```
parameter width = `pixel_width;
parameter addr_width = 3;

input [(width*2)-1 : 0] a_in;
input [addr_width-1:0] write_addr, read_addr;
input reset, en, clk;

output [(width*2)-1 : 0] latch_out;

reg [(width*2)-1 : 0] latch_out;

wire enable_0,
     enable_1,
     enable_2,
     enable_3,
     enable_4,
     enable_5,
     enable_6,
     enable_7;

reg  in,
     in_0,
     in_1,
     in_2,
     in_3,
     in_4,
     in_5,
     in_6,
     in_7; //signals for the latch enable gating signals

wire clk_n;


wire [(width*2)-1 : 0] out_0,
                       out_1,
                       out_2,
                       out_3,
                       out_4,
                       out_5,
                       out_6,
                       out_7;

//demultiplexing the input to the latch_bank
always @(write_addr or en)
   begin
     if(write_addr == 3'b000)
       in_0 = en;
     else in_0 = 0;

     if(write_addr == 3'b001)
       in_1 = en;
     else in_1 = 0;

     if(write_addr == 3'b010)
       in_2 = en;
     else in_2 = 0;

     if(write_addr == 3'b011)
       in_3 = en;
     else in_3 = 0;

     if(write_addr == 3'b100)
       in_4 = en;
     else in_4 = 0;
     if(write_addr == 3'b101)
       in_5 = en;
     else in_5 = 0;

     if(write_addr == 3'b110)
       in_6 = en;
     else in_6 = 0;

     if(write_addr == 3'b111)
       in_7 = en;
     else in_7 = 0;
   end //always

//Gating the clk signal with enable
not inv_clk(clk_n, clk);
and act_en0(enable_0, clk_n, in_0);
and act_en1(enable_1, clk_n, in_1);
and act_en2(enable_2, clk_n, in_2);
and act_en3(enable_3, clk_n, in_3);
and act_en4(enable_4, clk_n, in_4);
and act_en5(enable_5, clk_n, in_5);
and act_en6(enable_6, clk_n, in_6);
and act_en7(enable_7, clk_n, in_7);

//end of gated clock

//instantiation of the latches required
latch L0  (.a_in(a_in) , .enable(enable_0), .reset(reset),.latch_out(out_0));
latch L1  (.a_in(a_in) , .enable(enable_1), .reset(reset),.latch_out(out_1));
latch L2  (.a_in(a_in) , .enable(enable_2), .reset(reset),.latch_out(out_2));
latch L3  (.a_in(a_in) , .enable(enable_3), .reset(reset),.latch_out(out_3));
latch L4  (.a_in(a_in) , .enable(enable_4), .reset(reset),.latch_out(out_4));
latch L5  (.a_in(a_in) , .enable(enable_5), .reset(reset),.latch_out(out_5));
latch L6  (.a_in(a_in) , .enable(enable_6), .reset(reset),.latch_out(out_6));
latch L7  (.a_in(a_in) , .enable(enable_7), .reset(reset),.latch_out(out_7));

//multiplexing the output of the latch_bank
always @(read_addr or out_0 or out_1 or out_2 or out_3 or out_4 or  out_5 or out_6 or out_7)
  begin
    case(read_addr)
      3'b000 : latch_out = out_0;
      3'b001 : latch_out = out_1;
      3'b010 : latch_out = out_2;
      3'b011 : latch_out = out_3;
      3'b100 : latch_out = out_4;
      3'b101 : latch_out = out_5;
      3'b110 : latch_out = out_6;
      3'b111 : latch_out = out_7;
      default : latch_out = 0;
    endcase
  end //always
endmodule //latch_bank


/*
 * Top module
 */

module mac(a_in, b_in, write_addr, read_addr, en, clk, reset, mac_out);

parameter width = `pixel_width;
parameter addr_width = `counter_width;

output [(width*2)-1 : 0] mac_out;

input [addr_width-1 : 0] write_addr;
input [addr_width-1 : 0] read_addr;
input [width-1 : 0] a_in;
input [width-1 : 0] b_in;
input clk, reset, en;

/* signals */
wire [(width*2)-1 : 0] sum;
wire [(width*2)-1 : 0] product;

wire clk, reset;


/* multiplier used here is behavioural, basically (*) */

mult_csa mult_1(.a_in(a_in), .b_in(b_in),.mult_out(product));

/* the adder is behavioural as well (+) */

adder_bk add_1(.a_in(mac_out), .b_in(product), .adder_out(sum));

/*the latch bank has been coded at RTL level */

latch_bank latches(.a_in(sum), .clk(clk), .en(en),.reset(reset), .write_addr(write_addr), .read_addr(rea

endmodule
```

```
// Filename        : cb_mac_tb-20ns.v
// Description     : test bench for column based mac unit
// Author          : Shedden Masupe

module cb_mac_sim_test ;


parameter pixel_width = 8;
parameter image_order = 64;
parameter matrix_order = 8;
parameter addr_width = 3;
parameter freq = 20;


reg  [pixel_width-1 : 0]  a_in, b_in;
reg clk, reset, en;
reg [pixel_width-1 : 0] cos_array[0:image_order];
reg [pixel_width-1 : 0] pel_array[0:image_order];
reg [addr_width-1:0] write_addr, read_addr;

wire  [(pixel_width*2)-1 : 0] mac_out;
wire tc_control;

integer n, a, b, d;
integer write_file, latch_file;

assign tc_control = 1;

//instantiate dut
mac dut(.a_in(a_in), .b_in(b_in), .write_addr(write_addr),
        .read_addr(read_addr), .en(en), .clk(clk),
        .reset(reset), .mac_out(mac_out));

//initialisation
initial
 begin
  reset = 0;
  en = 0;
  a_in = 0;
  b_in = 0;
 // mac_out = 0;

 #50 reset = 1;
// #15 write_addr = 7;
//     read_addr =0;
end

//------------------------------------------------
//read from file
//------------------------------------------------
initial
  begin
    wait(reset);

    $readmemb("./input/col_cos_100_ham.dat",cos_array);
    $readmemb("./input/lena_part_8.dat",pel_array);
    //$readmemb("./input/ones.dat",cos_array);
    //$readmemb("./input/twos.dat",pel_array);
    write_file = $fopen("./output/mac_out-20ns.dat");
    latch_file = $fopen("./output/latch_out.dat");
    $set_toggle_region (dut);
    $toggle_start;
    $fwrite(write_file, "                time  w  r  a_in   b_in  mult_out  adder_out  mac_out\n");
    $fwrite(latch_file, "                time      L0       L1       L2      L3       L4      L5        L6
  //read in vectors
```

```
   for(n = 0; n<matrix_order; n= n+1)//counter for pel cols
     begin
reset=0;
       for(b=0; b<matrix_order; b=b+1)//counter for cosine cols and pel rows
         begin
           for(a=0; a< matrix_order; a=a+1) //do the cosine rows
             begin
               @(posedge clk);
       reset=1;
       en = 1;
               b_in = cos_array[a+(b* matrix_order)];
               a_in = pel_array[b+(n* matrix_order)]-128;
write_addr=a;
read_addr= a+1;
              // write_addr= write_addr+1;
     //  read_addr = read_addr + 1;

               $fwrite(write_file, $time,, "  %d  %d  %d    %d %d     %d   %d\n",
                       write_addr, read_addr,a_in, b_in, dut.mult_1.mult_out,
                       dut.add_1.adder_out,  mac_out);
      $fwrite(latch_file, $time,, "  %d  %d  %d %d   %d %d    %d   %d\n",
                      dut.latches.L0.latch_out, dut.latches.L1.latch_out,
                      dut.latches.L2.latch_out, dut.latches.L3.latch_out,
                      dut.latches.L4.latch_out, dut.latches.L5.latch_out,
                      dut.latches.L6.latch_out, dut.latches.L7.latch_out );
              // $monitor($time,,"%d  %d   %d", a_in, b_in, mac_out);
             end // counter a
         end // counter b
     end //counter n
   $toggle_stop;
   $toggle_report("./output/cb_mac_norm-20ns.saif", 1.0e-9, "cb_mac_sim_test.dut");
   $finish;
  end

// generate clock
initial
     clk = 0;
     always #(freq/2) clk = !clk;

endmodule // cb_mac_sim_test
```

References

[1] T. H. Meng, B. M. Gordon, E. K. Tsern, and A. C. Hung, "Portable video-on-demand in wireless communication," in *Proceedings of the IEEE*, pp. 659 – 680, IEEE, April 1995.

[2] P. Pirsch, N. Demassieux, and W. Gehrke, "VLSI architectures for video compression - a survey," in *Proceedings of the IEEE*, pp. 220 – 246, IEEE, Feb 1995.

[3] Synopsys Inc., 700 East Middlefield Road,Mountain View, CA 94043-4033, USA, *Synopsys Online Documentation, Power products reference manual*, 1998.

[4] A. T. Erdogan, *Low Power FIR Filter Implementations for VLSI Based DSP Systems.* PhD Thesis, Cardiff University, UK, August 1999.

[5] A. Bellaouar and M. I. Elmasry, *Low-Power Digital VLSI Design : Circuits and Systems.* Kluwer Academic Publishing, 1995.

[6] J. M. Rabaey and M. Pedram, *Low Power Design Methodologies.* Kluwer Academic Publishers, 1996.

[7] F. N. Najm, "A survey of power estimation techniques in VLSI circuits," in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, IEEE, December 1994.

[8] A. Chandrakasan, S. Sheng, and R. Brodersen, "Low-power cmos digital design," in *JSSC*, vol. 27, pp. 473–484, 1992.

[9] G. Keane, J. R. Spanier, and R. Woods, "Low power design of signal processing systems using characterisation of silicon IP cores," in *Proceedings of the 33rd Asimolar Conference on Signals, Systems and Computers*, pp. 767 – 771, Oct. 1999.

[10] J. Monteiro, S. Devadas, and B. Lin, "A methodology for efficient estimation of switching activity in sequential logic circuits," in *Design Automation Conference*, pp. 12–17, 1994.

[11] C. Tsui, J. Monteiro, M. Pedram, S. Devadas, A. Despain, and B. Lin, "Power estimation for sequential logic circuits," in *IEEE Trans. VLSI Systems*, vol. 3, September 1995.

[12] S. Iman and M. Pedram, *Logic Synthesis for Low Power VLSI Designs.* Kluwer Academic Publishing, 1998.

[13] R. E. Bryant, "Graph-based algorithms for boolean function manipulation," in *IEEE Transactions on Computers*, pp. 677–691, IEEE, 1986.

[14] F. N. Najm, "Estimating power dissipation in VLSI circuits," in *IEEE Circuits and Devices Magazine*, 1994.

[15] F. N. Najm, R. Burch, and P. Y. I. Hajj, "CREST- a current estimator for CMOS circuits," in *IEEE International Conference on Computer-Aided Design*, pp. 204–2–7, IEEE, November 1988.

[16] F. N. Najm, "Transition density : A new measure of activity in digital circuits," in *IEEE Transactions on Computer-Aided Design*, vol. 12, pp. 310–323, IEEE, February 1993.

[17] R. Burch, F. Najm, P. Yang, and T. Trick, "McPOWER: A monte carlo approach to power simulation," in *IEEE/ACM International conference on Computer-Aided Design*, pp. 90–97, IEEE, November 1992.

[18] M. Xakellis and F. Najm, "Statistical estimation of switching activity in digital circuits," in *31st IEEE/ACM Design Automation conference*, IEEE, 1994.

[19] R. S. Clarke, *Digital Compression of Still Images and Video*. Academic Press, 1995.

[20] R. C. Gonzalez and P. Wintz, *Digital Image Processing*. Addison-Wesley Publishing Company, second edition ed., 1987.

[21] Pattern-Recognition-Group, "Image Processing Fundamentals." http://www.ph.tn.tudelft.nl/Courses/FIP/frames/fip.html.

[22] K. Rao and P. Yin, *Discrete Cosine Transforms: Algorithms, Advantages and Applications*. New York Academic Press, 1990.

[23] E. Ifeachor and B. Jervis, *Digital Signal Processing : A Practical Approach*. Addison-Wesley Publishing, 1993.

[24] S. K. Mitra and J. F. Kaiser, *Handbook for Digital Signal Processing*. John Wiley and Sons, 1993.

[25] M. L. Hilton, B. D. Jawerth, and A. Sengupta, "Compressing still and moving images with wavelets," *Multimedia Systems*, vol. 2, no. 5, pp. 218–227, 1994.

[26] C. S. Burrus, R. A. Gopinath, and H. Guo, *Introduction to Wavelets and Wavelets Transforms : A Primer*. Prentice-Hall, 1998.

[27] M. W. Marcellin, M. J. Gormish, A. Bilgin, and M. P. Boliek, "An overview of JPEG-2000," in *Data Compression Conference*, pp. 523–544, 2000. citeseer.nj.nec.com/marcellin00overview.html.

[28] J. Cardinal, "Fractal compression using the discrete karhunen-loeve transform." http://citeseer.nj.nec.com/cardinal98fractal.html, Brussels Free University, Computer Science Department, Bld. du Triomphe CP212, B1050 Brussels, Belgium, 1998.

[29] N. Ahmed, T. Natarajan, and K. R. Rao, "Discrete cosine transform," in *Transactions on Computing*, vol. c, pp. 90 – 93, IEEE, Jan 1974.

[30] V. Bhaskaran and K. Konstantinedes, *Image and Video Compression Standards: Algorithms and Architectures*. Kluwer Academic Publishers, 1995.

[31] J. F. Blinn, "What's the deal with the DCT," in *IEEE Computer Graphics and Applications*, pp. 78 – 83, July. 1993.

[32] T.-S. Chang, C.-S. Kung, and C. Jen, "A simple processor core design for DCT/IDCT," in *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 10, pp. 439–447, IEEE, April 2000.

[33] I. Sundsbo, G. L. Hansen, and E. J. Aas, "Comparison of two architectures for implementation of the discrete cosine transform," in *ICASSP 96*, pp. 3272 – 3275, May 1996.

[34] G. S. Taylor and G. M. Blair, "Design for the discrete cosine transform in VLSI," in *IEEE Proceedings in Computer and Digital Tech*, vol. 145, pp. 127 – 133, Mar. 1998.

[35] N. Demassieux, G. Concordel, J.-P. Durandeau, and F. Jutand, "An optimised VLSI architecture for a multiformat discrete cosine transform," in *Proceedings of ICSP'98*, pp. 85 – 88, 1998.

[36] F. A. McGovern, R. F. Woods, and M. Yan, "Novel VLSI implementation of (8x8) point 2-D DCT," *IEE Electronics Letters*, vol. 8, pp. 624 – 626, April 1994.

[37] A. Madisetti and A. Willson, "A 100mhz 2-d 8x8 DCT/IDCT processor for HDTV applications," in *IEE Transactions on Circuits and Systems for Video Technology*, vol. 5, pp. 158–165, April 1995.

[38] A. Madisetti and A. Willson, "DCT/IDCT processor design for HDTV applications," in *Proceedings of Signals, Systems and Electronics, ISSSE'95*, vol. 5, pp. 63–66, April 1995.

[39] S. Uramoto, Y. Inoue, A. Takabatake, J. Takeda, Y. Yamashita, H. Terane, and M. Yoshimoto, "A 100mhz 2-d discrete cosine transform processor," in *IEEE Journal of Solid State Circuits*, vol. 27, pp. 492–499, April 1992.

[40] M. Matsui, H. Hara, Y. Uetani, L. Kim, T. Nagamatsu, Y. Watanabe, A. Chiba, K. Matsuda, and T. Sakurai, "A 200mhz $13mm^2$ 2-d DCT macrocell using sense-ampliflying pipeline flip-flop scheme," in *IEEE Journal of Solid State Circuits*, vol. 29, pp. 1482–1490, December 1994.

[41] Y. Jang, J. Kao, J. Yang, and P. Huang, "A $0.8mu$ 100mhz 2-d DCT core processor," in *IEEE Transactions on Consumer Electronics*, vol. 40, pp. 703–709, August 1994.

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

[43] Y. Arai, T. Agui, and M. Nakajima, "A fast DCT-SQ scheme for images," in *The Transactions of the IEICE*, vol. E71, pp. 1095–1097, November 1988.

[44] C. Chiu and K. J. Liu, "Real-time parallel and fully pipelined two-dimensional DCT lattice structures with applications to HDTV systems," *IEEE Transactions on Circuits and Systems for Video Technology*, pp. 25 – 37, March 1992.

[45] K. J. R. Liu and C. T. Chiu, "Unified parallel lattice structures for time-recursive discrete cosine/sine/hartley transforms," Technical Research Report TR 91-36, University of Maryland, College Park, EE Dept, Sysetems Research Center, University of Maryland, College Park, MD 20742, 1991.

[46] S. A. White, "Applications of distributed arithmetic to digital signal processing: A tutorial review," *IEEE ASSAP Magazine*, pp. 5 – 19, July 1989.

[47] Y.-H. Chan and W.-C. Siu, "On the realization of discrete cosine transform using the distributed arithmetic," in *IEEE Transactions on Circuits and Systems-I: Fundamental Theory and Applications*, vol. 39, pp. 705–712, IEEE, September 1992.

[48] M. Kuhlmann and K. K. Parhi, "Power comparison of flow-graph and distributed arithmetic based DCT architectures," in *Conference Record of the Thirty-Second Asimolar Conference on Signals, Systems and Computers*, vol. 2, pp. 1214–1219, 1998.

[49] G. Fettweis, J. Chiu, and B. Fraenkel, "A low-complexity bit-serial DCT/IDCT architecture," in *IEEE International Conference on Communications*, vol. 1, (Geneva), pp. 217–221, 1993.

[50] M. Sanchez, J. D. Brugera, and E. L. Zapata, "Bit serial architecture for the two dimensional DCT," in *International conference on Signal Processing, Applications and Technology, ICSPAT'95*, (Boston, USA), pp. 662–666, 1995.

[51] V. Karunakaran, *ASIC Design of Bit-Serial and Bit-Parallel Discrete Cosine Transform Processors.* Msc thesis, University of Maryland, 1994.

[52] S. Sachidanandan, *Design, Implementation and Testing of an 8x8 DCT Chip.* Master of science, University of Maryland, College Park, 1989.

[53] P. T. Balsara and D. T. Harper, "Understanding VLSI bit serial multipliers," in *IEEE Transactions on Education*, vol. 39, pp. 19–28, February 1996.

[54] D. Crook and J. Fulcher, "A comparison of bit serial and bit parallel DCT designs," in *VLSI Design*, vol. 3, pp. 59–65, 1995.

[55] P. Denyer and D. Renshaw, *VLSI Signal Processing : A bit serial Approach.* Addison-Wesley, 1985.

[56] A. P. Chandrakasan and R. W. Brodersen, *Low Power Digital CMOS Design.* Kluwer Academic Publishers, 1995.

[57] I.-M. Pao and M.-T. Sun, "Computation reduction for discrete cosine transform," in *International Symposium on Circuits and Systems*, pp. 285 – 288, IEEE, 31 May - 3 June 1998.

[58] L. G. Chen, J. Y. Jiu, H. C. Cheng, Y. P.Lee, and C. W. Ku, "Low power 2D DCT chip design for wireless multimedia terminals," in *International Symposium on Circuits and Systems*, vol. 4, pp. 41 – 44, IEEE, 31 May to 3 June 1998.

[59] A. T. Erdogan and T. Arslan, "Data block processing for low power implementation on single multiplier CMOS DSP processors," in *International Symposium on Circuits and Systems*, IEEE, 31 May - 3 June 1998.

[60] S. Cho, T. Xanthopoulos, and A. P. Chandrakasan, "Ultra low power variable length decoder for MPEG-2 exploiting codeword distribution," in *Proceedings of the 1998 IEEE Custom Integrated Circuits Conference*, pp. 177 – 180, IEEE, May 1998.

[61] T. Xanthopoulos, *Low Power Data-Dependent Transform Video and Still Image Coding*. Massachusetts Institute of Technology, PhD Thesis, 1999.

[62] T. Xanthopoulos and A. P. Chandrakasan, "A low power DCT core using adaptive bitwidth and arithmetic exploiting signal correlations and quantisation," in *IEEE Journal of Solid-State Circuits*, vol. 35, pp. 740–750, May 1999.

[63] V. G. Moshnyaga, "A MSB truncation scheme for low-power video processors," in *International Symposium on Circuits and Systems*, pp. IV–291 to IV–294, IEEE, 31 May - 3 June 1999.

[64] J. Chen and K. J. R. Liu, "Cost-effective low-power architecture of video coding systems," in *International Symposium on Circuits and Systems*, pp. I–153 – I–156, IEEE, 31 May - 3 June 1999.

[65] J. Chen and K. J. R. Liu, "Low-power architectures for compressed domain video coding co-processor," in *IEEE Transactions on Multimedia*, vol. 2, pp. 111–128, IEEE, June 2000.

[66] K. Kim, P. A. Beerel, and Y. Hong, "An asynchronous matrix-vector multiplier for discrete cosine transform," in *Proceedings of the 2000 International Symposium on Low Power Electronics and Design*, pp. 256–261, 2000.

[67] J. Li and S.-L. Lu, "Low power design of two-dimensional DCT," in *Proceedings of the IEEE*, pp. 309 – 312, IEEE, Feb 1996.

[68] E. N. Farag and M. I. Elmasry, "Low power implementation of discrete cosine transform," in *Sixth Great Lakes Symposium on VLSI*, pp. 174–177, IEEE, 1996.

[69] L. G. Chen, J. Y. Jiu, and H. C. Cheng, "Design and implementation of low power DCT chip for wireless multimedia terminals," in *IEEE Workshop on Signal Processing Systems*, pp. 85–93, IEEE, 1998.

[70] K. Masselos, P. Merakos, T. Stouraitis, and G. E. Goutis, “A movel methodology for power consumption reduction in a class of dsp algorithms,” in *International Symposium on Circuits (ISCAS'98)*, vol. 6, pp. 199–202, IEEE, June 1998.

[71] J. Little and L. Shure, *Signal Processing toolbox for use with MATLAB.* The Mathworks Inc., July 1992.

[72] Cadence, *Synergy HDL Synthesiser and Optimiser Modelling Style Guide.* Cadence Design Systems, 555 River Oaks Parkway, San Jose, CA 95134, USA, version 3.0 ed., February 1997.

[73] D. E. Thomas and P. R. Morby, *The Verilog Hardware Description Language.* Kluwer Academic Publishers, 1995.

[74] A. R. Omondi, *Computer Arithmetic Systems: Algorithms, Architectures and Implementation.* Prentice Hall, 1994.

[75] C. R. Baugh and B. A. Wooley, “A two's complement parallel array multiplication algorithm,” in *IEEE Transactions on Computers*, pp. 1045 – 1047, IEEE, 1973.

[76] D. Dahan, “17x17 bit, high-performance, fully synthesizable multiplier,” in *IFIP International Workshop on Based Synthesis Design*, Dec 1999.

[77] G. W. Bewick, *Fast Multiplication: Algorithms and Implementation.* Phd thesis, Stanford University, Feb 1994.

[78] P. Bonatto and V. G. Oklobdzija, “Evaluation of booth's algorithm for implementation in parallel multipliers,” in *Proceedings of ASILOMAR-29*, vol. 1, pp. 608 – 610, IEEE, 1996.

[79] J. Fadavi-Ardekani, “Mxn booth encoded multiplier generator using optimised wallace trees,” in *IEEE Transactions on VLSI*, pp. 120–125, IEEE, 1993.

[80] B. Parhami, *Computer Arithmetic: Algorithms and Hardware Designs.* Oxford University Press, 2000.

[81] P. C. H. Meier, *Analysis and Design of Low Power Digital Multipliers.* Phd, Carnegie Mellon University, Pittsburgh, PA, August 1999.

[82] R. Zimmermann, “Lecture notes on computer arithmetic : Principles, architectures, and VLSI design.” http://www.iis.ee.ethz.ch/zimmi/publications/, Integrated Systems Laboratory, Swiss Federal Institute of Technology(ETH), CH-8092 Zurich, Switzerland, March 1999.

[83] H. A. Al-Twaijry, *Area and Performance Optimised CMOS Multipliers.* Phd, Stanford University, August 1997.

[84] D. Duarte, J. Hezavei, and M. J. Irwin, "Power consumption and performance comparative study of logarithmic-time CMOS adders," in *IEEE Workshop on Signal Processing Systems*, pp. 467–476, 2000.

[85] R. Brent and H. Kung, "A regular layout for parallel adders," in *IEEE Transactions on Computers*, pp. 260–264, March 1982.

[86] B. Holdsworth, *Digital Logic Design*. Butterworth-Heinemann, third ed., 1993.

[87] Q. Wu, M. Pedram, and X. Wu, "Clock-gating and its application to low power design of sequential circuits," in *IEEE Transactions on Circuits and Systems-I: Fundamental Theory and Applications*, vol. 47, pp. 415–420, IEEE, March 2000.

[88] F. Theeuwen and E. Seelen, "Power reduction through clock gating by symbolic manipulation," in *IFIP International Workshop on Logic and Architecture Synthesis*, 1996.

[89] C. H. Roth, *Fundamentals of Logic Design*. PWS Publishing Company, fourth ed., 1995.

[90] S. Masupe and T. Arslan, "Low power DCT implementation approach for CMOS based DSP processors," *IEE Electronics Letters*, vol. 34, pp. 2392 – 2394, Dec 1998.

[91] K. Chang, *Digital Systems Design with VHDL and Synthesis: And Integrated Approach*. IEEE, Computer Society, 1999.

[92] P. Pirsch, *Architectures for Digital Signal Processing*. John Wiley and Sons Ltd, 1998.

[93] G. K. Wallace, "The JPEG still picture compression standard," in *IEEE Transactions on Consumer Electronics*, vol. 38, pp. xviii – xxxiv, Feb. 1992.

[94] K. Jack, *Video demystified: A handbookfor the digital engineer*. Eagle Rock VA 24085: LLH Technology Publishing, third ed., 2001.

[95] T. Lane, "JPEG FAQ." http://www.faqs.org/faqs/jpeg-faq/, March 1999.

[96] F. Gadegast, "MPEG FAQ." http://www.faqs.org/faqs/mpeg-faq/, March 1999.

[97] "Synopsys designpower and power compiler : Technology backgrounder." http://www.synopsys.com/products/power/power_bkg.html, 700 East Middlefield Road,Mountain View, CA 94043-4033, USA, January 1998.