Design of a Low Power Variable Length Decoder for MPEG-2 System

by

SeongHwan Cho

Bachelor of Science in Electrical Engineering,
Korea Advanced Institute of Science and Technology (KAIST) 1995

Submitted to the Department of Electrical Engineering and Computer Science in partial fulfillment of the requirements for the degree of

Master of Science
in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 1997

Signature of Author _____
Department of Electrical Engineering and Computer Science
June, 1997

Certified by _____
Anantha P. Chandrakasan
Assistant Professor of Electrical Engineering
Thesis Supervisor

Accepted by _____
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Design of a Low Power Variable Length Decoder for MPEG-2 System

by

SeongHwan Cho

Submitted to the
Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science in Electrical Engineering and Computer Science

Abstract

Variable length coding is a widely used technique in digital video compression systems. The main idea of variable length coding is to minimize the average codeword length by exploiting the statistics of the data. Shorter codewords are assigned to frequently occurring data while longer codewords are assigned to infrequently occurring data. Therefore, minimum average codeword length and bit rate reduction can be achieved. Previous works related to variable length decoders are primarily aimed at high throughput applications, but the increased demand for portable multimedia systems have made power a very important factor.

This work presents a data driven variable length decoding algorithm, which exploits the signal statistics of variable length codes to reduce power. It uses various table partitioning methods based on codeword frequency. Power minimization is focused on high level architecture and algorithms as well as circuit optimizations. A chip is implemented, which is capable of decoding all 15 different types of MPEG-2 variable length codes. An order magnitude of power reduction is possible compared to a conventional parallel decoding scheme with a single lookup table.

Thesis Supervisor: Anantha P. Chandrakasan

Title: Analog Devices Career Development Assistant Professor
of Electrical Engineering and Computer Science

Key Words: variable length decoder, low power, table partitioning, MPEG-2, Huffman code, variable length code

Acknowledgments

I would like to express my sincere gratitude to a number of people who have contributed to the completion of this work.

- Anantha Chandrakasan for his insight, enthusiasm and constructive supervision. His patience and encouragement guided me to the completion of this project. Working with him was a privilege.
- All the lab mates of ‘ananthagroup’ for helping me out not only with this project but also with the life at MIT. I’d especially like to thank Thucydides (Duke) Xanthopoulos for all the tutorials and advice he gave me. I am greatly indebted to him for all his patient answers and lessons. None of this work could have been possible without him.
- Members of Korean Graduate Students Association and Hansori who gave me *life* in this foreign world. I had some fun in Boston, thanks to them.
- My high school and college colleagues of ‘nine’ board in bbs@csqueen.kaist.ac.kr, who gave me love and support from Korea. I’d especially like to thank Kee-Eung Kim, who stood by me along the way since my early years.
- My deepest thanks and respect goes to my family: Halmoni, Appa, Umma and Hyun-Ju nuna. They have consistently given me love and support more than I deserve. I love them always.

부모님께...

Table of Contents

List of Figures

List of Tables

Chapter 1
Introduction

Video data coding/decoding is one of the key components of multimedia application systems. Variable length coding is a lossless coding scheme widely used in digital video compression standards. This coding method is often applied together with other lossy coding techniques to further compress the data without degrading the image quality. The main idea of variable length coding is to minimize the average codeword length by exploiting the statistics of the data. Shorter codewords are assigned to frequently occurring data while longer codewords are assigned to infrequently occurring data. Therefore, minimum average codeword length and bit rate reduction can be achieved.

With the emergence of portable multimedia devices, power has become an important factor and it should be explicitly considered in the design methodology. This thesis involves developing techniques to reduce power of a variable length decoder in an MPEG-2 decoder system. Power reduction is mainly achieved by exploiting the signal statistics as well as low level circuit optimizations.

1.1 Variable Length Decoder

The task of a variable length decoding system is to decode the incoming variable length codes and produce the corresponding output for the next unit. There are some unique

properties of this variable length code which makes it attractive to video/image compression systems, one of which is compression without loss of information. Hence, this coding method is used together with other lossy coding techniques to further compress the data and bit rate. In an MPEG-2 system, variable length coding is applied together with the discrete cosine transform (DCT), run length encoding and motion compensation.

1.2 MPEG-2

MPEG-2 is a compressed digital video syntax standard. Due to the huge amount of the video data, the information is compressed several times before it is sent to the receiver. Figure 1.1 shows the basic diagram of the system. First the video is compressed by motion estimation, DCT and quantization. The redundancy of the compressed data is further reduced by run length coding and variable length coding. The decoder does the opposite of the encoder. The variable length decoder is placed at the front most part of the decoder and its output is further decompressed as it is passed along the decoder system.

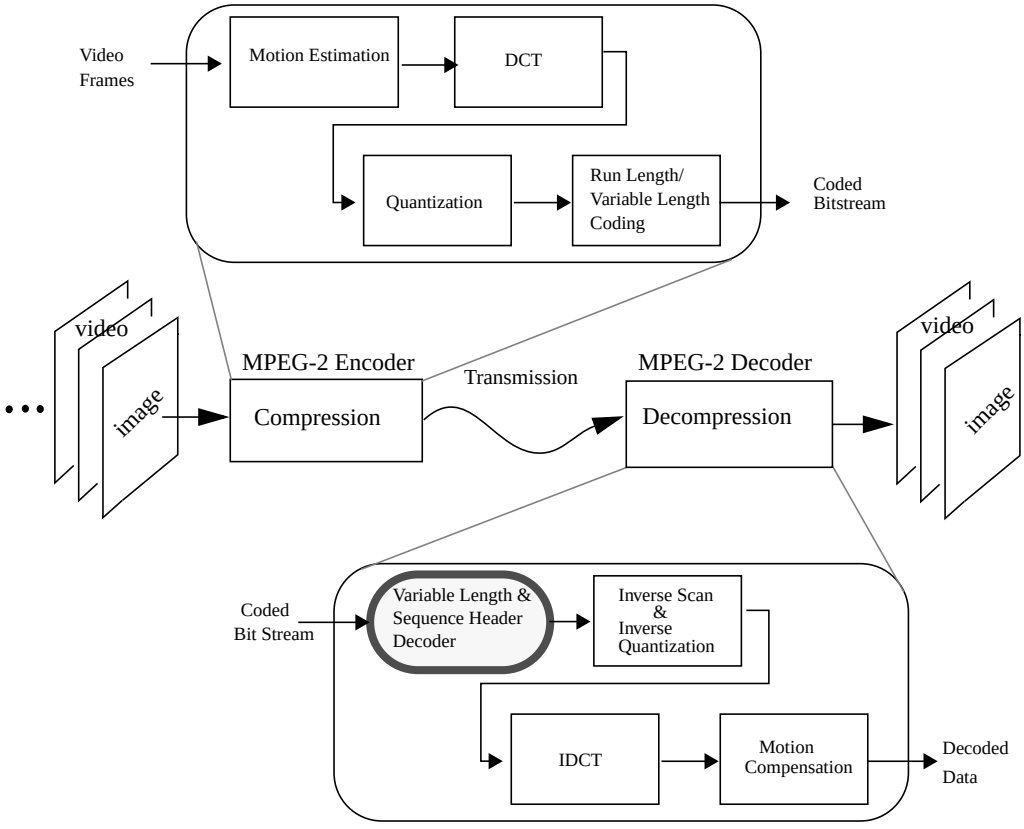


FIGURE 1.1: Simplified View of MPEG-2 Encoder/Decoder System [Ker93]

1.3 Low Power Design

To lower the power in a system, all aspects of that system must be considered, from low level technology to high level architecture. Often, there are constraints on the implementation process and technology. For instance when implementing a chip with a given process, the designer cannot change the device level parameters, such as the threshold voltage V_T.

As for a typical CMOS integrated circuits, the switching power in the variable length decoder can represented as follows.

$$P_{switching} = \alpha \cdot C \cdot V_{dd}^{2} \cdot f \tag{1.1}$$

where C is the physical capacitance, V_{dd} is the supply voltage, f is the clock frequency and α is the activity factor which is the probability that C will be charged at any given cycle. Although there are other components of power as short circuit and leakage power, the dynamic power is the most dominant. All the terms in this equation should be reduced to lower power. Obviously V_{dd} is the most influential term: it has a quadratic influence. Circuit and architectural level optimizations such as parallel processing and minimizing critical delay path, can be applied to reduce V_{dd}. The α factor or the frequency can be reduced with algorithmic transformations and C can be reduced using appropriate circuit style selection and transistor size optimization. There is of course certain trade-off by changing the parameters. We seek the optimum condition by making various changes in these parameters and transformations. This work is primarily focused on architecture and algorithm level optimizations to reduce power as well as circuit optimization.

1.4 Contribution of This Work

There have been various approaches proposed to increase the throughput of the variable length decoder. Although the performance has increased, the power dissipation has also steadily increased. This work proposes a novel method which we call “non-uniform fine grain table partitioning” which splits the table into several variable size blocks to reduce power. Several ways to decomposing the table are introduced, which exploits the incoming

variable length code statistics. An order magnitude power reduction can be achieved compared to the conventional schemes with a single lookup table.

Chapter 2
Background

This chapter gives background information on variable length codes, MPEG-2 and variable length decoders. First, the generation of the variable length codes and its properties will be discussed. Next, the MPEG-2 is explained, including where the variable length code is used. Finally, previous works of variable length decoder are shown, with their throughput and power.

2.1 Variable Length Code

Variable Length Code(a.k.a. Huffman Code) was first established here at MIT, by D.A. Huffman [Huf52]. It has some unique properties, which makes it attractive to certain applications as low bit rate communication and compressed video transmission systems. It has been shown that this coding method results the minimum average length code.

2.1.1 Huffman code and its properties

Suppose we want to transmit a certain message composed of N symbols. The occurring probability for each symbol is P_1, P_2, P_3,... P_N. Before the symbols are sent to the receiver, they need to be encoded in a binary representation. Denoting the encoded symbols as S_1, S_2, ... S_N, the following equation holds, where L_i is the length of the encoded symbol S_i and L_{avg} is the average symbol length.

$$\sum_{i=1}^{N} P_i = 1 \qquad L_{avg} = \sum_{i=1}^{N} P_i \cdot L_i \tag{2.2}$$

The goal is to minimize the L_{avg}. Huffman defined a minimum-redundancy code (i.e. Huffman code), which for a message consisting of a finite number of members, N, and for a given number of coding digits, D, yields the lowest possible average message length. The following constraints are imposed on the optimum redundancy codes.

(a) No two messages will consist of identical arrangements of coding digits.

(b) The message code will be constructed in such a way that no additional indication is necessary to specify where a message code begins and ends once the starting point of a sequence of messages is known.

The above restrictions necessitate that no message be coded in such a way that its code appears as a prefix to any other message code of greater length. For example, suppose there are four symbols we want to encode and transmit: a,b,c,d. They are encode as follows: a=1, b=01, c=001, d=011. A sequence of the coded messages 1001010101 can be broken up into the individual codes 1-001-01-010-1 (acbda). All the receiver needs to know is how each symbol is coded and where the message starts. However, if the symbols are coded in such a way that one of the code includes other code as its prefix, then it is not possible to decode the table. For instance, if a=1, b=01, c=001, d=010 is the symbol representation, it is not immediately certain whether the message 0101 means 01-01(bb) or 010-1(da). The whole decoding procedure is destroyed and the reconstruction of the compressed data is impossible. The above conditions enable the reconstruction of the coded messages in a unique way.

2.1.2 Construction of the Huffman Code and the Huffman Tree.

With the restrictions given as above, the Huffman code can be generated by the procedure shown in Figure 2.1. First of all, the symbols are ordered by their probability. Next, two symbols with the smallest probabilities are combined into one. As they are combined, each is assigned a binary digit 0 and 1. The reduced number of symbols are then reordered with the probability and the same procedure is applied again. The is done until the proba-

bility sums up to a symbol whose probability is one. Figure 2.1 shows the result of this coding method. The encoding procedure is quite straightforward once this tree has been drawn. The Huffman code is generated by concatenating the binary digits down along the tree starting from the root. Figure 2.2 shows the final result of the coding and its binary tree representation. The binary tree representation is important for further analysis and also for the decoding procedure.

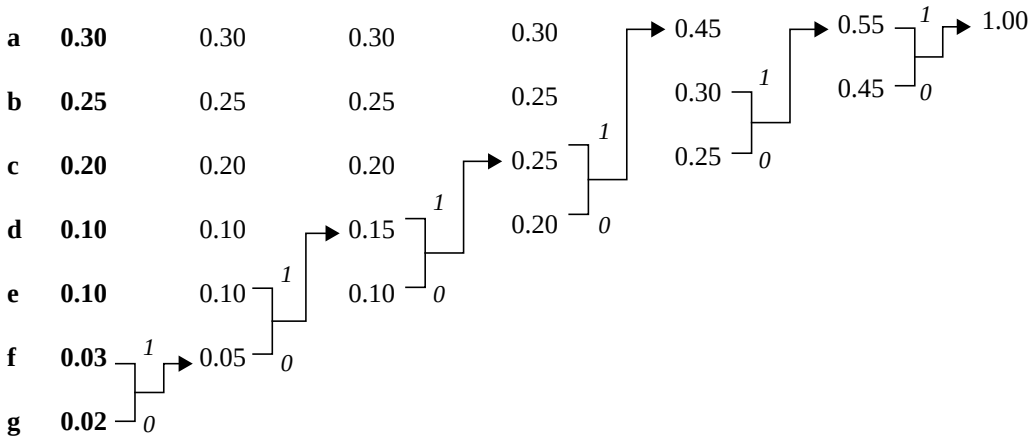


FIGURE 2.1: Construction of Huffman Codes

Symbol	P_i	L_i	Code
a	0.30	2	11
b	0.25	2	10
c	0.20	2	00
d	0.10	3	010
e	0.10	4	0111
f	0.03	5	01101
g	0.02	5	01100

TABLE 2.1: Example of a Huffman Code

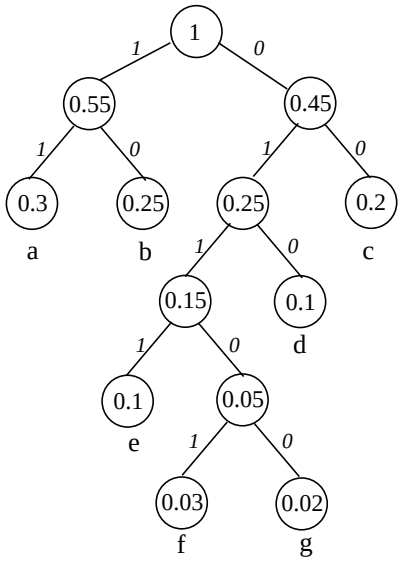


FIGURE 2.2: Constructed Huffman code and its binary tree

2.2 MPEG

ISO/IEC 13818, commonly known as MPEG-2 (Moving Picture Experts Group) is a compressed digital video syntax standard [MPEG-2]. The first generation of MPEG was MPEG-1[1] which was optimized for compact disc videos or applications at about 1.5 Mbits/sec. Due to the low bit rate that the compact disc is capable of, the image quality of MPEG-1 was similar to that of a VHS tape and it was not suitable for high bit rate systems. People in the broadcast television area recognized the potential of MPEG technology to increase the channel efficiency of the satellite transponders and cable networks, but the broadcast industry was not limited to the compact disc low bandwidth and was unwilling to settle for VHS resolution. Consequently the MPEG committee developed a second standard called MPEG-2 specifically designed for broadcasting applications. The MPEG-2 standard is designed to represent CCIR 601 resolutions (720x480x30 Hz: 720 samples/line by 480 lines per frame by 30 frames/sec) video at data rates of 4.0-8.0 Mbits per second. In addition, MPEG-2 provides support for interlaced fields, 16:9 aspect ratio video, multiple video channels in a single system stream, and extensibility to HDTV. It is also important to note that MPEG1 is a subset of MPEG-2 so any MPEG-2 decoder will be able to decode MPEG1 syntax video.

2.2.1 MPEG-2 bit stream

MPEG-2 bit stream is a single bit stream consisting of various sub-streams. In its general form, it can be decomposed into two layers: system layer and compression layer. The system layer contains timing and other information needed to demultiplex the video and audio streams. The compression layer has the compressed video and audio streams. In its physical form, the bit stream can be categorized into two streams: the fixed length code stream and the variable length code stream. Hence in addition to the variable length decoder, there needs to be a fixed length decoder (or a microcontroller), which is capable of decoding the fixed length codes. The microcontroller does not know when and where to start decoding the fixed length codes because it cannot not tell where the variable length

1. ISO/IEC 11172

code ends in the bitstream. Therefore there is a handshake logic between the microcontroller and the variable length decoder. Details are discussed in chapter 5, where we deal with the chip specifications.

2.2.2 MPEG-2 Video Hierarchy

The hierarchical structure of MPEG-2 video data is defined as follows.

Video Sequence > Group of pictures > Picture > Slice > Macroblock > Block

The below figure shows the schematic view of the hierarchy. The picture is decomposed all the way down to an 8 by 8 pixel block. The variable length codes represents the macroblock and block level characteristics of the video data.

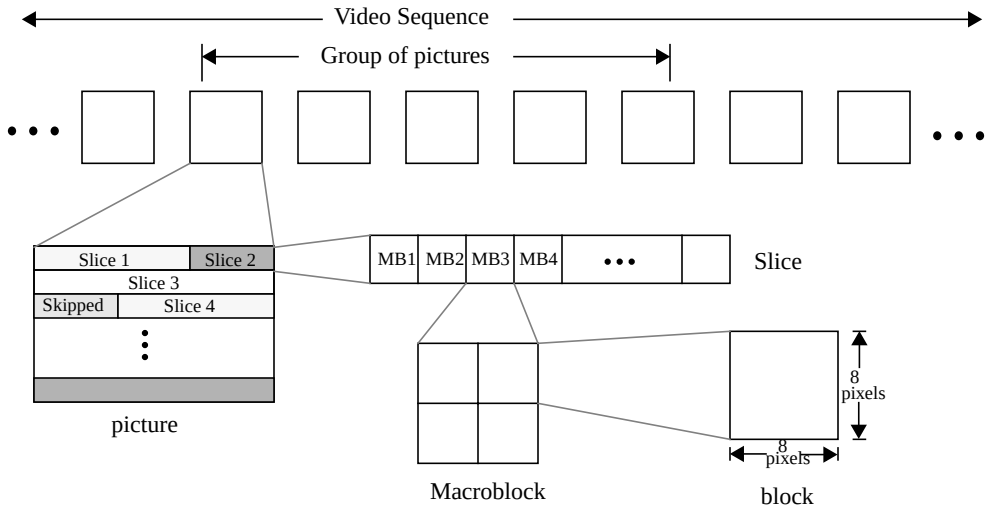


FIGURE 2.3: MPEG-2 Video Hierarchy

2.2.3 MPEG-2 Variable Length Codes

There are 15 different variable length code (VLC) tables in the MPEG-2 standard. Each VLC conveys information of the video sequence and image data. MBA is macroblock address increment which tells the difference between the previous and current macroblock address. Macroblock address defines the absolute position of the current macroblock. Macroblock type indicates the method of coding and contents of the macroblock. There are seven different macroblock types: I,P,B picture each with or without spatial scalability and one with SNR scalability. The coded block pattern (CBP) indicates how each block in

the macroblock is ordered. Motion vectors (MV) are used for motion compensation. It provides an offset from the coordinate position in the current picture to the coordinates in the reference frame. Finally there are the DCT coefficients. They are grouped according to their block and frame types, whether it's a luminance or a chrominance block and whether the frame is an intra or non-intra type. Table 2.2 shows the variable length code (VLC) information of each table. It is retrieved from a 60 frame MPEG-2 sequence. It shows the number of VLC entries in each table. Minimum, maximum and average length of the VLC in the table are also shown. As can be seen, the DCT coefficients are dominant, consuming more than 80% of the variable length code stream.

VLC TYPE	ISO 13818	Table	# of VLCs	MIN Length	MAX Length	Avg Length	%
MBA	B-1	MBA	34	1	11	1	2.01%
MB Type	B-2	I	2	1	2	1.23	.66%
	B-	P	7	1	6	4.45	.66%
	B-4	B	11	1	6	4.36	.66%
	B-5	I w/ ss	5	1	4	0	0%
	B-6	P w/ ss	16	1	7	0	0%
	B-7	B w/ ss	20	1	9	0	0%
	B-8	SNR	3	1	3	0	0%
CBP	B-9	CBP	64	3	9	5.09	.88%
M V	B-10	MV	33	1	11	2.49	4.10%
	B-11	DMV	3	1	2	0	0%
DCT DC	B-12	DC lum	12	2	9	3.36	3.71%
	B-13	DC chm	12	2	9	7.46	1.90%
DCT AC	B-14	R&L (0)	114	2	17	5	59.2%
	B-15	R&L (1)	113	2	17	5.55	26.34%

TABLE 2.2: MPEG-2 Variable Length Codes (average length = 4.91)

2.3 Variable Length Decoder: Previous Works

2.3.1 Overview of Variable Length Decoder

The basic function of a variable length decoder (VLD) can be described in two words:

“Table Lookup”. Since the VLCs are encoded based on a one to one mapping without any arithmetic operations, same applies to the decoder, i.e. no computation is need to decompress the data. The VLD receives the input data and simply generates output by looking at the corresponding codeword in its table. Although the basic functionality lies in the table lookup, the key operation of the VLD is how to detect the VLC in the bit stream. The input to the VLD is a bit stream without explicit word boundaries. The VLD has to decode a codeword, determine its length and shift the input data stream by the number of bits corresponding to the decoded code length, before decoding the next codeword. These are recursive operations that cannot be pipelined nor parallelized. The input data is variable length and the VLD doesn’t know the where the next code starts until the previous VLC is decoded. Hence, at the very beginning of the variable length decoding process, the start of the variable length code stream must be told to the VLD. In addition, each time a VLC is decoded, a feedback from the decoded codeword must be fed to determine the start of the next VLC.

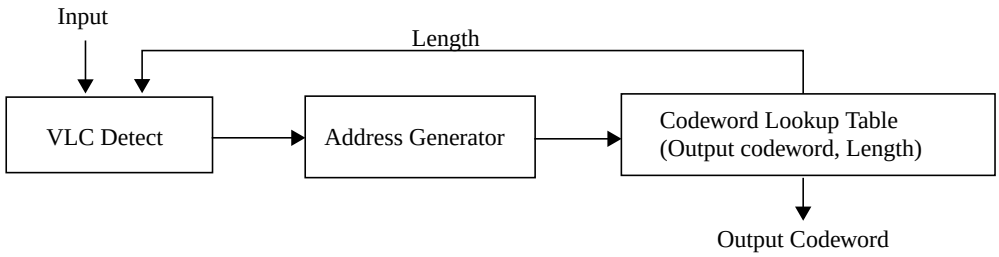


FIGURE 2.4: High Level View of a Variable Length Decoder

Figure 2.4 shows a high level view of a variable length decoder. As shown in this figure, the variable length decoder can be decomposed into three components: the VLC detector, the address generator and the lookup table. The variable length code detector receives the input bit stream and gives the aligned VLC to the address generator. The VLC detector aligns the VLC at a fixed position for the address generator, by receiving a feedback (the length) from the codeword table. The address generator gives the address to the lookup table and the lookup table produces the corresponding output codeword and the VLC length. The length is necessary to decode the next codeword, to tell the VLC detector where the next codeword starts. In many cases, it is difficult to split the system into three parts, because the address generator and the VLC detector can be combined into one unit.

The address generation can be thought of as another table lookup procedure, which is redundant to the next codeword lookup step. Hence, there are often cases where the VLC detector and the address generator are implemented as one. In such cases the combined unit will be referred to as the VLC detector.

In the next section, we'll study various implementations of the VLD. Some of the important designs will be analyzed with their pros and cons, including the power and throughput.

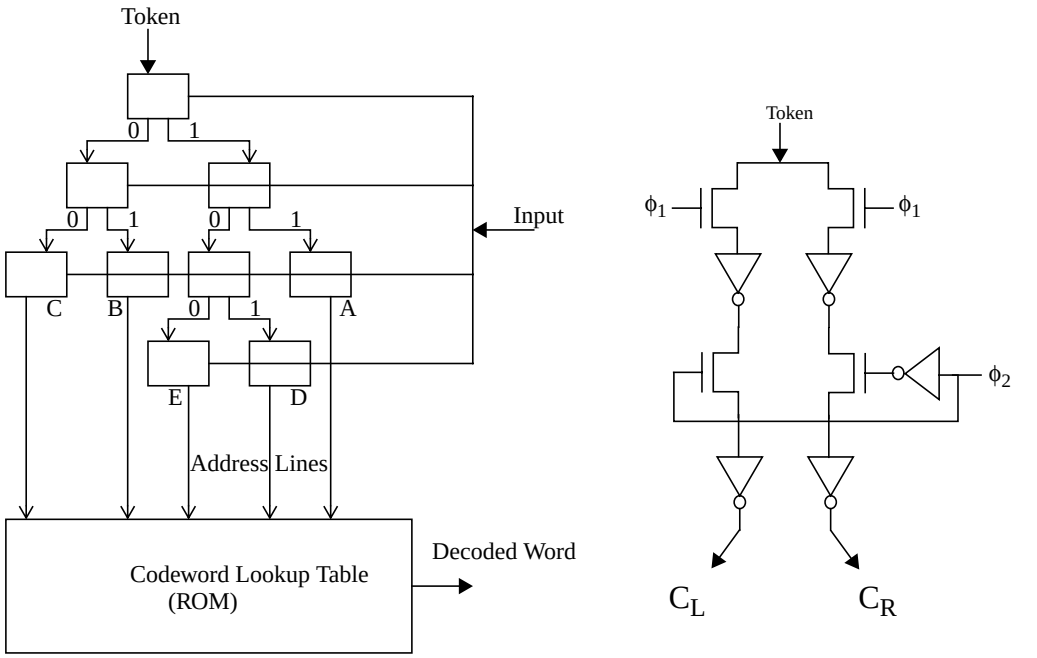
2.3.2 Binary Search Variable Length Decoder (Serial VLD)

Mukherjee proposed a bit serial approach for the VLD [MRB91]. The basic idea is to traverse through the Huffman tree in the reverse order in which the VLCs were generated. Figure 2.5 shows the fundamental architecture and the circuit implementation of the proposed scheme. The leaves of the Huffman tree are connected to the input of the lookup table as the address. The function of a node in the Huffman tree module is to pass the incoming token to one of its two child nodes depending on the value of the input code bit. If the code bit is a "1", the token is passed to the left child C_L and if the code bit is a "0" the token is passed to the right child C_R. The operation of the reverse Huffman tree architecture is as follows. At the beginning, a start token is applied at the root of a tree. This token traverses down the tree, controlled by the input VLC bits. If the token emerges out of a leaf node, it initiates a read operation from the table. The decoded codeword is loaded into the output buffer register, and the token is fed back to the root node to continue the decoding of the next symbol. Thus the token that emerges out of the leaf node initiates a memory read as well as the decompression of the next symbol. The critical delay of the circuit depends on the time needed for the read operation from the ROM. The ROM access time will determine the speed at which code bits can be pumped into the device. The critical delay of the circuit can be improved by replacing the ROM by a balanced binary tree module built to represent the uncompressed fixed length codes.

The average throughput can be represented as the following equation, where f_{clock} is the rate at which the token is going through, and L_{av} is the average length of the VLC.

$$f_{throughput} = f_{clock}/L_{av} \tag{2.1}$$

The disadvantage of this scheme is that the output is generated at a variable rate and that it takes several cycles to decode a codeword, where the number of decoding cycles is equal to the VLC length. Therefore to achieve a high throughput, the system clock must be running at a very high frequency. In low power design, this is undesirable since the power increases linearly with the clock frequency. In addition, it is difficult to operate the system at high clock speed under a low voltage.



(a) Reverse Tree Architecture (b) Circuit Implementation of the node

FIGURE 2.5: Bit Serial VLD

2.3.3 Parallel Variable Length Decoder

Lei and Sun proposed a parallel architecture of the VLD, which decodes the VLCs in one cycle, regardless of its length [LS91]. As opposed to the bit serial method, where the incoming VLC is processed one bit per cycle, the parallel method uses multiple bits per cycle by using a barrel shifter. The basic architecture is shown in Figure 2.6. The VLC detector uses a barrel shifter and an accumulator and the codeword lookup table is implemented using a PLA.

The basic operation is as follows. We assume that the maximum length of the VLC does not exceed 8. In the beginning the input data are stored in 8 bit registers D_0 and D_1.

The barrel shifter reads 16 bit input from these registers and produces 8 bit output. The AND plane of the PLA essentially performs a parallel pattern matching on the data stream. When a codeword is matched, the corresponding wordline in the PLA AND plane is activated. This enables the corresponding word transistors in the OR planes to output the decoded codewords and the VLC length. The VLC length is fed to the accumulator, which stores the amount of bits that have been processed. The accumulator tells the barrel shifter how many bits to shift so that the start of next VLC is aligned at the most significant bit of the barrel shifter output. When the accumulated code length exceeds 8, the carry out bit becomes 1. It indicates that all the bits in D_1 have been used and that D_0 may not contain the next whole codeword. When the carry out signal becomes one, a new set of data is fed into the register D_0 from the buffer and D_1 receives it data from D_0. If the accumulated code length does not exceed 8, the carry out signal stays 0. Since the maximum code length is 8 and at least 8 bits of data in D_0 and D_1 are not used yet, there are always enough bits for the next decoding cycle. Using this parallel scheme, the VLC decoding is achieved in one clock cycle regardless of the code length. Figure 2.6 shows an example of the decoding process. The highlighted bits represent the barrel shifter output. The output bits are shifted by the accumulator which stores the amount of bits that have been used.

The lookup table is implemented in PLA. It could also be implemented using a ROM or a CAM (Content Addressable Memory). However, it would require a 2^{Lmax} size ROM to build a VLC table which has maximum codeword length L_{max}. This would be very wasteful and hence we prefer PLA or CAM whose sizes are determined by the number of code book entries. The PLA is preferable to a CAM since CAM requires much larger and complex circuit. CAM is may be used for programmable VLDs, where the contents of the codeword table can be changed.

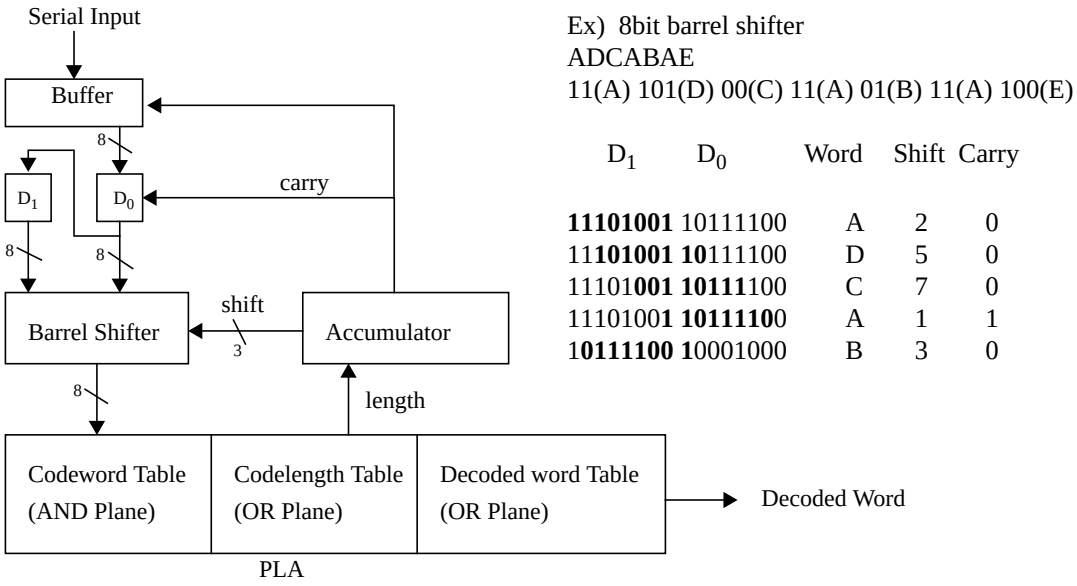


FIGURE 2.6: Parallel Variable Length Decoder [LS91]

The throughput of the parallel VLD is set by the clock frequency of the system,

$$f_{throughput} = f_{clock} \tag{2.2}$$

It has a constant throughput. The clock speed is determined by the critical path which is the feedback from the PLA to the accumulator and then the barrel shifter.

A large portion of the power is dissipated by the PLA. The PLA becomes very large for big VLC tables and it is charged and discharged every cycle.

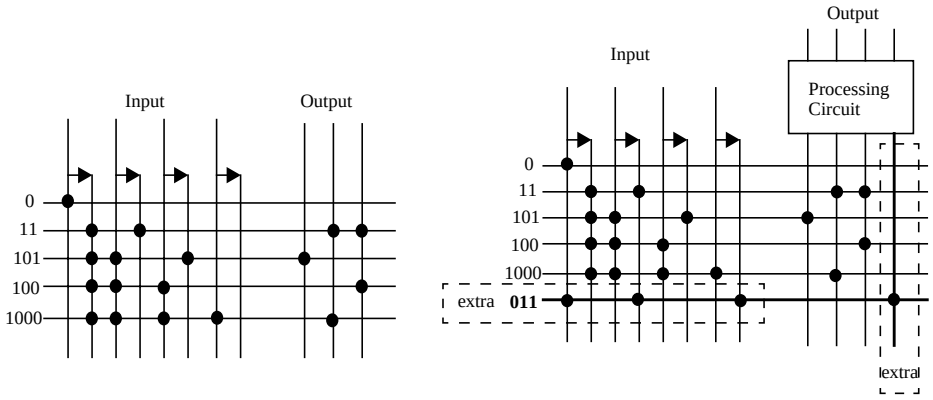
2.4 Other methods

2.4.1 High Speed Parallel VLD

The performance of the previous parallel VLD is further improved by exploiting the VLC characteristics [Jan92]. This scheme is under the assumption that the probability of short VLCs coming in successively is high. Whereas the previous VLD decoded one codeword per cycle, this method decodes more than one codeword per cycle. The main idea of this approach is to decode successive short codewords in one cycle. In the actual circuits, this can be implemented by adding extra rows and columns which detects these short successive codewords to the PLA. The below figure shows an example of this high speed parallel

VLD. The augmented PLA has extra “011” which decodes successive occurrence of “0” and “11”. A post processing circuit is necessary at the output because more than one output is produced at a time, and they need to be ordered appropriately.

Although the throughput has increased, this scheme needs complex circuitry and larger PLA for proper operation. It also loses the advantage of constant output rate. From a power perspective, it is not very attractive when it is compared to the original scheme Lei and Sun proposed. This is because of the added capacitance in the PLA and additional processing circuit at the output.



(a) Original PLA (b) Augmented PLA

FIGURE 2.7: High Speed VLD

2.4.2 Adaptive VLD

This adaptive method has a very fast and efficient method of decoding the VLCs [OTD94] . It is an extended method of the binary search technique. Instead of searching one bit at a time as in the binary tree search method, this adaptive VLD traces couple of bits per cycle. Figure 2.8 shows a adaptive tree representation of the VLC. As the figure shows, the adaptive method traces multiple bits per cycle, as opposed to the binary search technique where only one bit is processed per cycle. The adaptive method has fewer nodes, which enables faster VLC decoding. Yet, this tree search technique is not attractive at low supply voltage and hence low power. This is because the system frequency must be higher than the output codeword throughput.

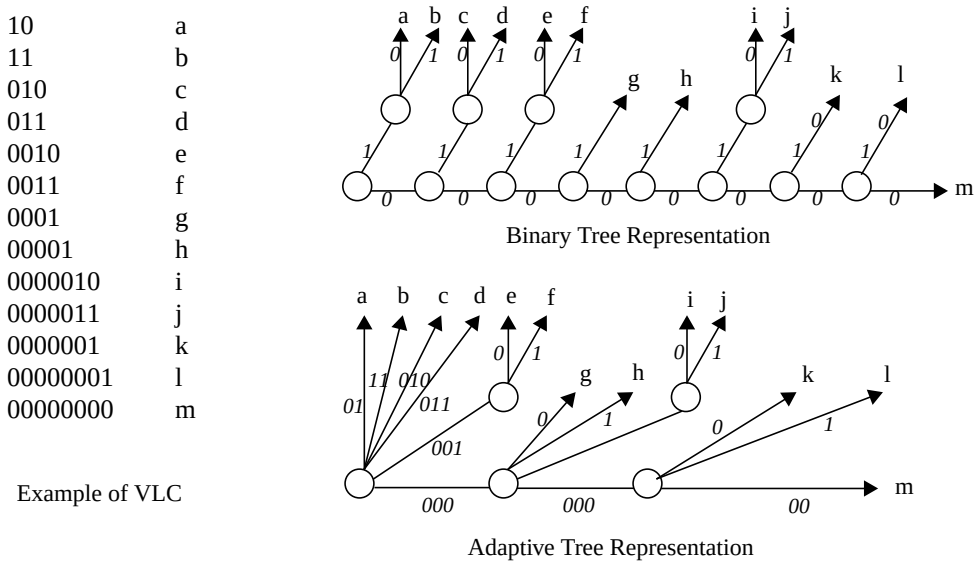


FIGURE 2.8: Adaptive Tree VLD

2.4.3 Pattern matching VLD

Choi and Lee proposed a high speed pattern matching VLD [CL94]. It does what is so called a “pre-decoding”. As the VLC table gets larger, there are many VLCs which have same bits that are common, which we’ll refer to as prefix. The key idea is to pre-decode the VLCs according to their prefix first and then decode the rest. Figure 2.9 shows an example of prefix based VLC pre-decoding. The bold bits indicate the prefix of the variable length code. The size of the lookup table is reduced by clustering the prefixes in one block.

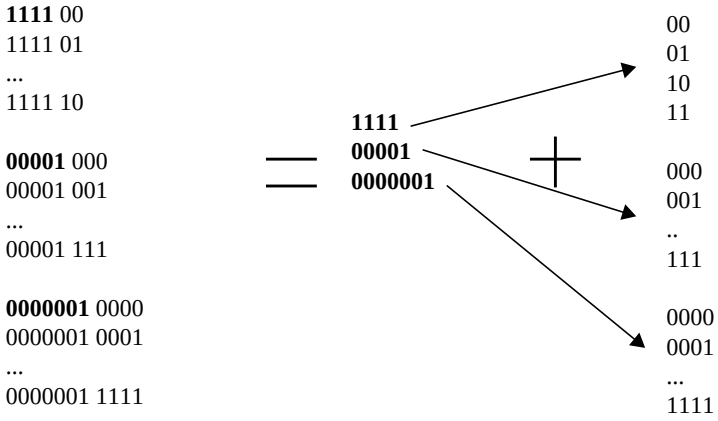


FIGURE 2.9: Prefixed based Pre-decoding VLD

2.4.4 ROM based VLD

This method uses ROMs for the lookup table [KS94]. In order not to waste the ROM area, an address generator is inserted between the VLC detector and the lookup table. The address generator produces minimum number of bits to implement the ROM such that the waste of ROM area is minimal. The ROM is decomposed into three equally sized ROMs which stores the VLCs according to their frequency of occurrence. Since we have the VLCs in separate tables, only one of the ROM is activated at a time. Therefore power and delay performance is improved. The drawback is the additional address generator, which basically does the table lookup procedure. The address generator burns additional power and its function is redundant to the output codeword lookup procedure. Figure 2.10 shows the ROM based VLD. The leading 0 detector counts the number of 0s that are in the VLC and selects appropriate ROM.

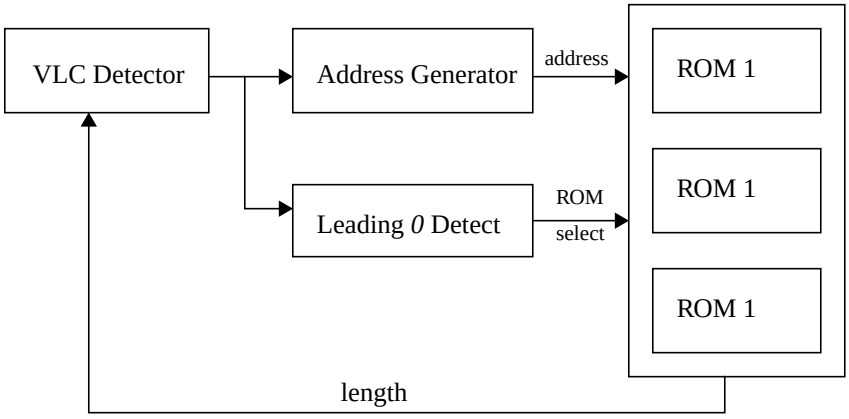


FIGURE 2.10: ROM Based VLD

2.5 Summary

Several implementations of the variable length decoder were shown, which were aimed at high throughput. These variable length decoders are largely divided into two categories according to their VLC detector method. One is a serial method where one bit is processed per cycle, and the other is the parallel method where multiple bits are processed per cycle. To achieve low power VLD, all the components of the VLD must be considered. In the next chapter, we will extensively study the VLC detector for both approaches.

Chapter 3
Variable Length Code Detector

This chapter gives an analysis of the variable length code (VLC) detector. It starts with an overview of the low power variable length decoder design, where high level optimization is discussed. The following sections show two implementations of the VLC detector, a parallel method and a bit serial method. Power and throughput is analyzed for both implementations. Finally a summary is given for these two architectures.

3.1 Overview of Low Power Variable Length Decoder Design

3.1.1 A High Level Perspective

The first step in optimizing for low power is a detailed analysis of how the power is distributed and what part of the system dissipates the dominant power. In order to do this, the system should be decomposed into its functional units. As mentioned in the previous chapter, we decomposed the variable length decoder (VLD) into three parts: VLC detector, address generator and the lookup table. To achieve our goal of designing a low power VLD, all these three units must be optimized. Since all these components are closely related, to optimize one unit, the other parts must also be taken into account. In the following sections of this chapter, we will look into the VLC detector. Its power and throughput will be analyzed for different implementations. Reducing power in the lookup table will

be explored in the next chapter, and the VLC detector will be revisited for further optimizations based on the result of the lookup table power reduction.

3.1.2 High Level Architecture

In a VLD system, the lookup table and the VLC detector are the most power and area consuming blocks. The address generation can be omitted if we use the VLC as the address input to the lookup table. After all, address generating is another table lookup procedure, which converts variable length codes to fixed length codes. We want the least number system blocks in implementing the low power VLD. Figure 3.1 shows the basic architecture of the VLD. We will use the VLC itself as the address to the lookup table and skip the address generation.

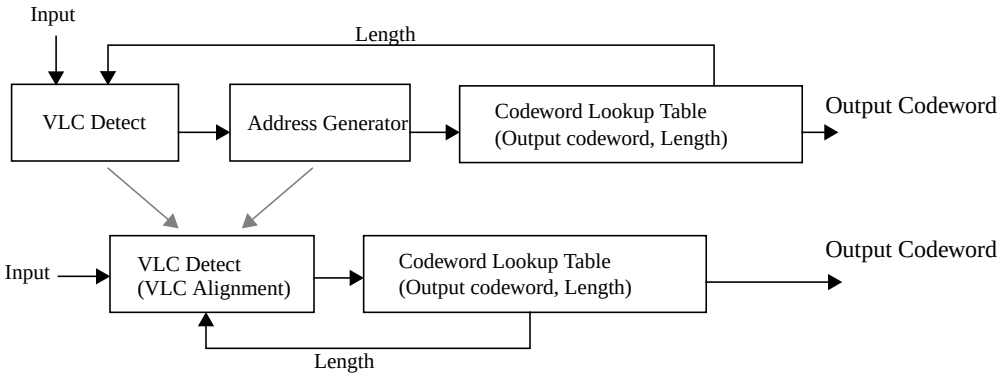


FIGURE 3.1: High Level View of the Low Power VLD

3.2 A Parallel Variable Length Detector

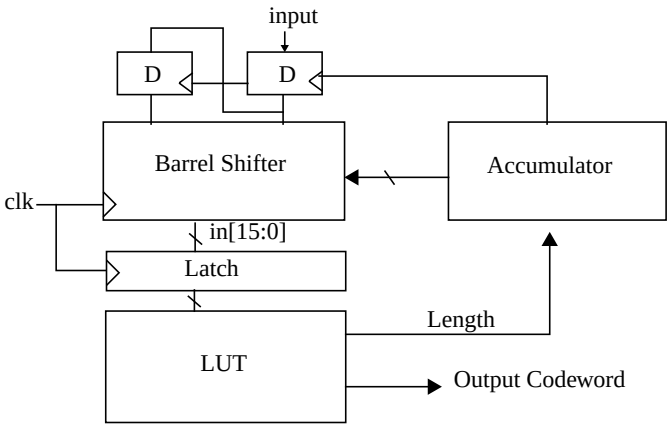


FIGURE 3.2: Parallel VLD

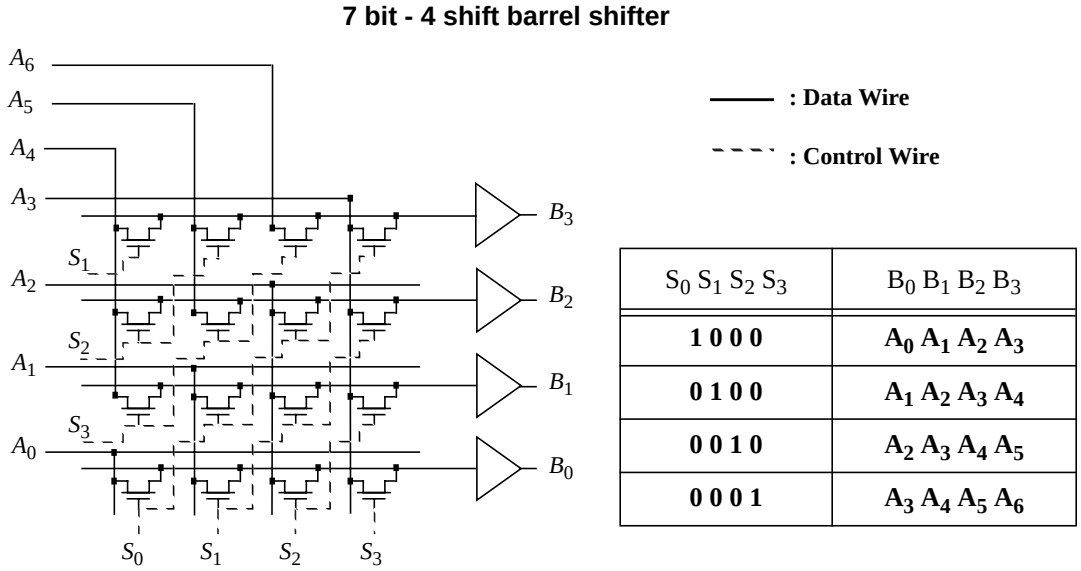
The parallel variable length detector is based on the work of Sun and Lei [SL92]. Figure 3.2 shows the hardware implementation of the VLD detector, which consists of a accumulator and a barrel shifter.

3.2.1 Barrel Shifter

The operation of the barrel shifter is to shift multiple bits per cycle. Two implementations of the barrel shifter are introduced, a regular and a logarithmic structure.

Regular Barrel Shifter

Figure 3.3 shows a typical barrel shifter. It takes 4 bits out of 7 bit input and it is capable of shifting up to 4 bits. The 4 control bits S_0 - S_3 determine how many bits to shift. Only one of the control bits must be asserted high. The functionality of the circuit is straightforward. When one of the control bits is high, it turns on the transistor which acts as a passgate. The input is shifted as it follows along the path of on-transistors and is delivered to the output buffer. In the real design, each NMOS transistor is replaced by a transmission gate. Hence both the true and its complementary signal of the control bits need to be provided.



S_0 S_1 S_2 S_3	B_0 B_1 B_2 B_3
1 0 0 0	**A_0 A_1 A_2 A_3**
0 1 0 0	**A_1 A_2 A_3 A_4**
0 0 1 0	**A_2 A_3 A_4 A_5**
0 0 0 1	**A_3 A_4 A_5 A_6**

FIGURE 3.3: Regular Structure Barrel Shifter [Rab95]

Logarithmic Barrel Shifter

The logarithmic barrel shifter is based on the idea that all numbers can be represented in binary form, i.e. sum of the powers of two. The previous implementation of *2n-1* bit input, *n*-bit shift barrel shifter can be designed using a network of $log_2 n$ stages, where each stage shifts a distance of *0* or 2^i bits. Each stage is implemented by a *2* input multiplexer which is controlled by the corresponding S_i bit. As opposed to the previous barrel shifter where each shift has its own control bit, (i.e. *n*-bit shift operation has *n*-bit control input) this method receives the input control in binary representation. Hence only $log_2 n$ bits are necessary for the control input. A 7 bit 4 shift logarithmic barrel shifter is shown Figure 3.4. Again, each NMOS transistor is replaced by a transmission gate in the actual design.

Using binary representation for the M-bit shift

$$M = \sum_{i=0}^{r-1} s_i \cdot 2^i$$

ex) $9 = 1*2^3 + 0*2^2 + 0*2^1 + 1*2^0$

S_0 S_1	B_0 B_1 B_2 B_3
0 0	$\mathbf{A_0\ A_1\ A_2\ A_3}$
0 1	$\mathbf{A_1\ A_2\ A_3\ A_4}$
1 0	$\mathbf{A_2\ A_3\ A_4\ A_5}$
1 1	$\mathbf{A_3\ A_4\ A_5\ A_6}$

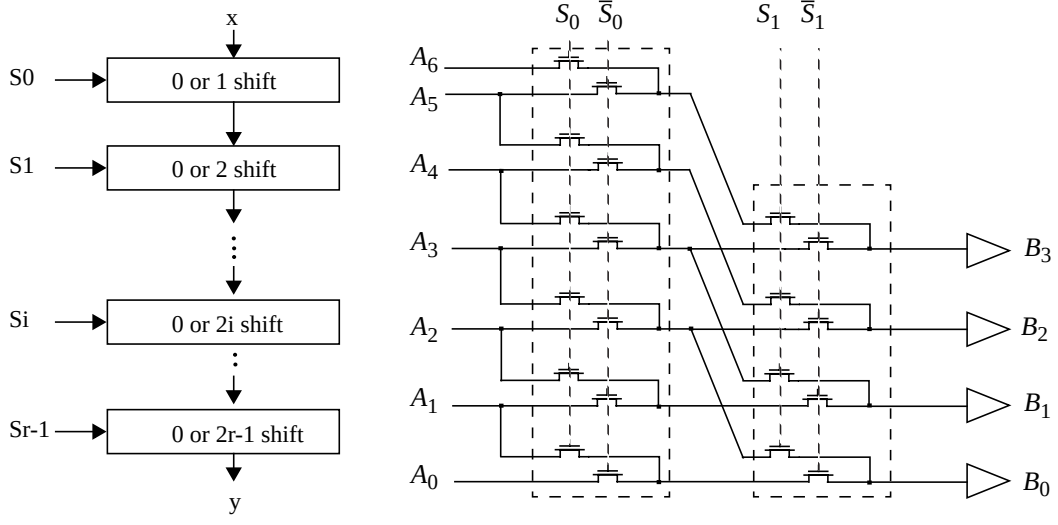


FIGURE 3.4: Logarithmic Barrel Shifter [Erc85]

Power, Delay

The delay of an n bit shift logarithmic barrel shifter is $log_2 n$ transistors whereas a regular structure has only one transistor delay independent of the barrel shifter size. The following graph shows a summary of the two types of barrel shifter. The regular structure has a better performance over delay and power when the input size is less than 12. Another important thing about the logarithmic barrel shifter is that, the delay is a series connection of transmission gates. For a low voltage supply operation, this is very dangerous and hence buffer must be inserted between the shift stages of the barrel shifter, which gives additional time delay.

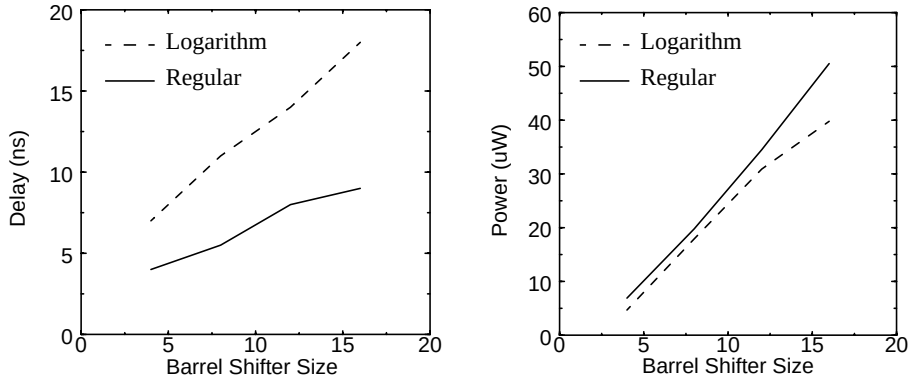


FIGURE 3.5: Power and Delay of Barrel Shifter (@ 1.3 V)

3.2.2 Accumulator

Another necessary part of this barrel shifter based VLC detector is the accumulator. The accumulator stores the length of the previously decoded VLCs to decode the next VLC.

Adder Accumulator

Figure 3.6 shows a common way to implement an accumulator, using an adder and a register. The operation is straightforward. The output of the adder is fed back into the input, and the inputs are accumulated. An interesting to note in this design is the position of the level and the edge sensitive latch. The edge sensitive latch is usually placed at the output of the adder, where the level sensitive latch is positioned in this design. The reason for placing the level sensitive latch there is to allow more computation time for the length gen-

eration and the addition. If we place a edge triggered latch at the end of the adder, then the adder must receive the length and finish the addition before the negative edge of the clock.

Hybrid Adder

The circuit implementation of the sum and the carry is shown in Figure 3.7. The carry is designed from a mirror adder and the sum is implemented using DCVSL[1]. The reason for such an implementation is because of the low supply voltage. Since we want to lower the power supply down as much as possible, the drive is very weak, especially for PMOS devices. In this design we are aiming at a supply voltage of less than 1.4V with a PMOS $|V_t|$ of around 0.9V. Hence more than two PMOS transistors in series are unacceptable. The DCVSL sum has only two PMOS transistors and they are cross coupled. When one of the output goes high the positive feedback immediately turns on which enables a very fast operation of the sum. It is very important that we reduce the delay time of the accumulator, since the critical path lies within this accumulator.

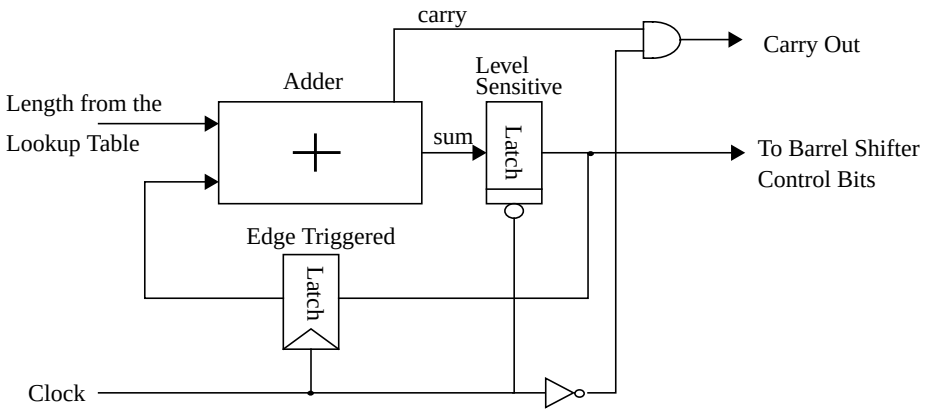


FIGURE 3.6: Adder Accumulator

1. Differential Cascode Voltage Switching Logic

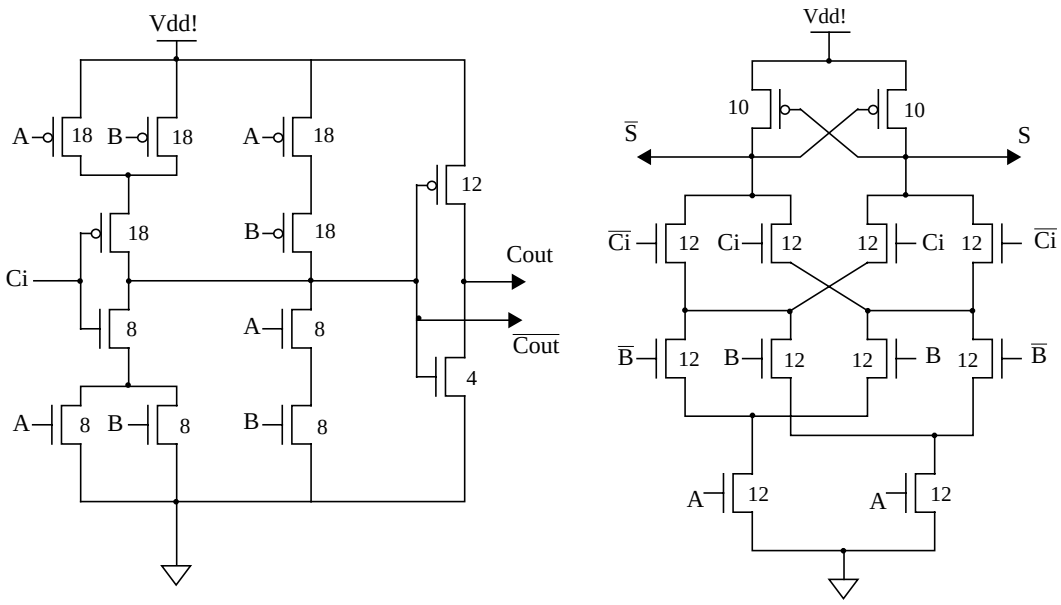
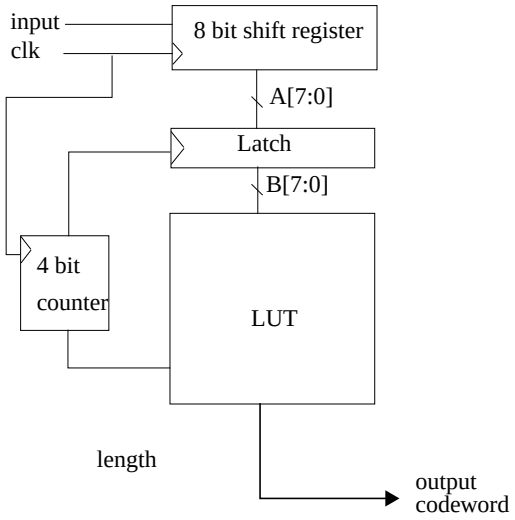


FIGURE 3.7: Circuit Schematic of Hybrid Adder (Courtesy of Thucydides Xanthopolous)

3.3 Serial Variable Length Code Detector

As Opposed to the parallel approach where multiple bits are shifted per cycle, the bit serial method shifts one bit per cycle. Hence the whole VLD system has to wait for several cycles to decode the next VLC until the new VLC is fully loaded into the shift register. Figure 3.8 shows the proposed VLD based on the bit serial scheme. The VLC detector is implemented using a shift register and a counter. The operation is as follows. Suppose a VLC of length *l* was decoded. The length *l* is immediately loaded to the counter, which counts *l* reference clock signals. During this time, the lookup table rests and the shift register loads the new VLC. After counting the *l* clock signals, the counter triggers the latch which will give the new VLC to the lookup table. This is when the previous VLC has been put out of the shift register and a new VLC is aligned at the most significant bit of the shift register. The new length is produced from the lookup table and is delivered to the counter and the same procedure continues. An example of this operation is shown in Figure 3.8. The bold numbers indicate the VLC that will be decoded.



A[7:0]	B[7:0]	Length	Trig
00101101	**001**01101	3	1
01**011010**	00101101	3	0
1**01101**01	00101101	3	0
01101010	**01101**010	5	1
1101**11**01	01101010	5	0
101**11**010	01101010	5	0
01**11**0100	01101010	5	0
1**11**01001	01101010	5	0
11010011	**11**010011	2	1
1**01001**10	11010011	2	0
	.		
	.		
	.		

FIGURE 3.8: Serial Variable Length Decoder

The advantage of this bit serial approach is the simplicity and compactness when implemented in circuits. While the parallel method uses a barrel shifter and an accumulator for the VLC detector, the serial method uses a counter and a shift register which consumes less power and area. The drawback is that the output is produced at a variable rate and a high clock speed is needed.

3.4 Power and Throughput

The power consumption of the VLC detector will depend on the clock frequency and its switching capacitance. Unlike the VLC lookup tables, VLC detector will always be operating. The clock frequency of the VLC detector is determined by the output codeword throughput f_{out} that we want to achieve. For the parallel approach, the barrel shifter clock frequency is the same as this output rate f_{out}. This is because the parallel method produces the output codeword independent of the input VLC length. For the serial approach, the output rate is $f_{out} = f_{SR} / L_{avg}$, where f_{SR} is the clock frequency of the shift register and L_{avg} is the average length of the variable length codes. The output codeword is produced every l clock cycles, where l is the length of the input VLC. Hence, if we want to get the same output codeword rate of the two implementations, the shift register in the serial approach

has to run L_{avg} times faster than the parallel method. Although the shift register has a higher operating frequency, simulation results show that the power consumption in the VLC detector is less for the serial approach than the parallel approach. This is because the barrel shifter and the accumulator has a higher switched capacitance than the shift register and the counter. At the same average output rate, a factor of 2 power reduction can be achieved in the VLC detector by using the serial method. However, for applications which require high throughput, the clock frequency of the serial approach has to be up to several hundred MHz, which may be too much an overhead under low supply voltage. Also, the rate at which the output codewords are produced will vary considerably in the serial approach. Additional circuits must be added to the output if a constant output rate is to be achieved.

3.5 High Level View from I/O

There are some differences between the two types of VLC detector in their I/O specifications. The bit serial method receives a single bit stream while the parallel method receives byte wide data. In reality, the actual compressed video sequence is encoded in a single bit stream and sent over the channel. In addition, the incoming compressed video sequence is not processed at a constant rate. Hence the decoder system receives a serial bit stream. This necessitates a buffer at the front end of the decoder for byte wide processing. Also, in MPEG-2, the decoded variable length code is not processed at a constant rate buffer and hence a FIFO is necessary at the output of the variable length decoder in the decoder system. Figure 3.9 shows three possible implementations of the front end of the MPEG-2 decoder system. For simplicity, the decoding units after the VLD are omitted.

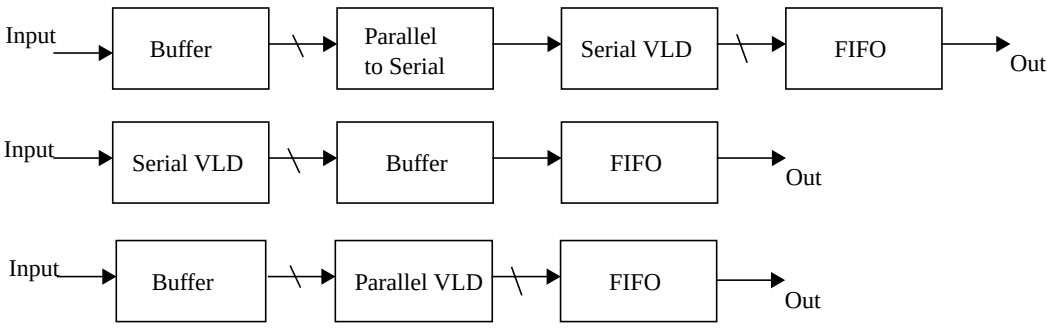


FIGURE 3.9: High Level View of the VLD with I/O Buffer

From a high level system view, the third approach is preferable. This is because it has the smallest size buffer and FIFO of all. Since the input is already compressed, it requires larger buffer to store the decoded data than the incoming bit stream. Also, a smaller FIFO can be used for the parallel VLD, since the variation of the output rate is smaller for the parallel VLD.

3.6 Summary

Two implementations of the VLC detector has been studied. The serial method has the advantage that it is compact and consumes less power. Yet a high frequency is needed to achieve a desired throughput. The parallel approach is desirable since it has a better performance in terms of low voltage operation. In the actual chip design, the parallel method with a regular barrel shifter structure is used. It will be revisited and optimized in the next chapter when we deal with table partitioning, which is based on this parallel regular barrel shifter VLC detector.

Chapter 4
Table Partitioning

This chapter presents a novel method which we call “fine grain non-uniform table partitioning”. In the first section, energy consumption in the lookup table is modeled. Based on this modelling the table is decomposed in three steps, by exploiting the VLC statistics and its characteristics. The VLC detector is revisited for further optimization based on this table partitioning.

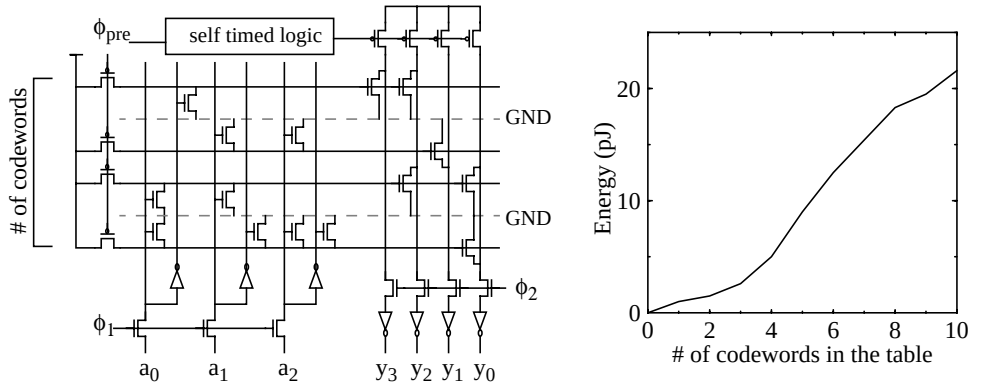
4.1 Energy Modelling

4.1.1 Overview

To lower the power dissipation in the lookup table, we first need to model the energy consumption. This is quite a difficult task, since the power depends on many various factors, such as the type of circuit, the contents of the table and the input statistics. It is not possible to deal with every possibility and therefore, it is necessary that we make some approximations and assumptions. In the following analysis, PLA and standard cell static CMOS will be used to model the energy. It has been mentioned earlier that PLA is suitable for implementing the VLC lookup tables. The PLA also has the advantage that since it has a regular structure, the power can be estimated within a reasonable level of accuracy. In the later section, static CMOS will be discussed and compared to the PLA.

4.1.2 PLA

The following figure shows an example of a regular PLA structure. As one can see in the schematic view, there are three variables mainly contributing to the power and area. The number of input bits (n_i), codewords (n_c), and the output bits (n_o). Although the area increases as $O(n_i n_c + n_o n_c)$, the power will not increase on the same order, because not all the increased capacitance will be charged and discharged every cycle. Figure 4.1 shows the energy profile of the PLA when both the number of codewords and input bits are increasing. The MPEG-2 DCT table has a maximum VLC of length of 18 bits and 227 codewords. When implemented in a single lookup table this would result in a significant amount of power dissipation.



(a) Example of a PLA *(b) Energy Consumption in PLA*

FIGURE 4.1: PLA [MC80]

4.1.3 Static CMOS

Static CMOS is another solution to implementing the lookup table. Static CMOS has the advantage that its functionality scales well with low supply voltage. The static CMOS is very robust under the conditions of $V_{supply} = 1.4V$ and $|V_{th}| = 0.7 \sim 0.9V$, In addition, there is no static power as in the case of PLA and short current power is eliminated since $|V_{tp}| + V_{tn} < V_{supply}$.

The following table shows the performance result of the PLA and static CMOS. It is obtained from a 16 codeword table simulation. The PLA consumes more power but has

a better performance in propagational delay. In this PLA simulation, additional clock generating circuits are not included. Hence, the PLA approach would in reality consume more power than the below figures.

	Delay	**Power**
PLA	3.5ns	2.14mW
Static CMOS	4.5ns	0.56mW

TABLE 4.1: Comparison of PLA and Static CMOS @2V, 50Mhz

4.2 Overview of Table Partitioning

The main idea of the table partitioning is to decompose the big single lookup table into several variable sized tables with respect to their occurring frequency. The most frequent codewords use the smallest lookup table (hence smaller wordline/bitline capacitance) while the least frequent codewords use larger tables. The basic flow chart of this table partitioning is shown in Figure 4.2.

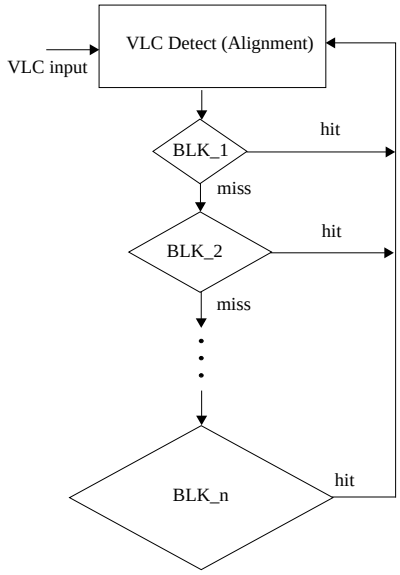


FIGURE 4.2: Low Power VLD Algorithm

The operation of this flow chart is as follows. At first, the VLC comes from the VLC detector looking for a match in the first small table. If there is a hit (i.e. a match) then the corresponding output codeword is produced and the length is provided to the VLC detector for the next detection (alignment) of the VLC. If the incoming VLC is a miss, the VLC goes on to the next block looking for a match. This process continues until the VLC

finds a match and it is fully decoded.

Now the question is how to decompose the table. Many factors are involved in the power optimization and it's not an easy task to achieve the optimum result. We must make sure that the throughput is not degraded by having multiple lookup tables. Another important issue is the power consumption in the VLC detector. In the proposed VLD scheme, the VLC detector is always operating unlike the decomposed lookup tables, of which only one of them is turned on at a given cycle.

In the next following sections, we'll look into optimum table partitioning, based on the energy profile studied in the previous section. The table partitioning will be done in three steps. First, the lookup table will be partitioned based on the VLC detector. It will be decomposed such that the VLC detector consumes the least amount of energy at a given throughput. Second, we will partition the table based on the prefix of the VLCs Table size and area will be reduced using this method. Finally, the table will be further decomposed by allocating different number of VLCs.

4.3 VLC Detector based Table Partitioning

This section introduces a first step to table partitioning. The lookup table will be partitioned based on the VLC detector's power dissipation.

To gain the maximum throughput, a single lookup table is necessary. In this case the barrel shifter size has to be at least the size of the maximum length VLC. On the other hand, the power in the VLD detector increases nearly up to a square order as the barrel shifter size. A small barrel shifter is desirable for low power but the opposite is preferred for high performance. An optimum solution can be achieved from the following analysis.

The following notations are used.

L_{max} : Maximum length of the input VLC.

Y_b : number of output bits from the barrel shifter (barrel shifter size).

$E(Y_b)$: Energy consumption of the VLD detector when barrel shifter size = Y_b

f_0 : operating frequency of the VLD detector.

f_{out} : throughput of the VLD (codewords/sec)

$P_{cr}(x,y)$: Sum of the occurring probabilities of VLCs which have length greater than x and less than y.

There are two cases of concern, $L_{max} =< Y_b$, $L_{max} >= Y_b$.

1) $L_{max} =< Y_b$

$$f_{out} = f_0 \tag{4.3}$$

$$Energy = E(Y_b) \tag{4.4}$$

This is when the size of the barrel shifter is larger than the maximum length of the VLC. It basically has the same structure as what Lei and Sun proposed [SL91]. The VLC can be decoded every cycle at a constant rate and the energy consumption in the VLC detector will be independent of the codeword probability.

2) $L_{max} >= Y_b$

$$f_{out} = (P_{cr}(0, Y_b) + 2 \cdot P_{cr}(Y_b, 2Y_b) + \ldots + n \cdot P_{cr}((n-1)Y_b, nY_b))^{-1} \cdot f_0 \tag{4.5}$$

$$Energy = [P_{cr}(0, Y_b) + 2 \cdot P_{cr}(Y_b, 2Y_b) + \ldots + n \cdot P_{cr}((n-1)Y_b, nY_b)] \cdot E(Y_b) \tag{4.6}$$

In this case, the decoding cannot be done in one cycle for VLCs which have length greater than Y_b. The VLC must be decomposed and decoded in several cycles in order to fit the barrel shifter. For instance, if the VLC is 000001000011 (length=12) and Y_b=8, then obviously the whole VLC cannot fit into the barrel shifter. The VLC has to be decomposed into two codewords of length l_1 and l_2 where l_1, l_2 represents the length of the decomposed VLC and $l_1 + l_2$ =12 (l_1, l_2 =< *8*). Equations (4.5) and (4.6) represents the throughput and energy consumption. The sum of probabilities indicates the average cycle it takes to decode a codeword. Figure 4.3 shows an example of this table partition. VLCs of length greater than 8 are decomposed into two blocks. If a VLC of length less than 8 comes in, it is decoded in blk0 and the corresponding output codeword is produced. This is done in one cycle and the operation is the same as the one with a single lookup table. If a VLC of length greater than 8 comes in, for example a 12 bit VLC, then only the first 8 bits are

loaded to the blk0. Since blk0 doesn't have all the information to decode the VLC by receiving the first 8 bits, the next four bits are passed on to blk1, where the VLC is fully decoded.

For convenience, the example in the figure shows l_1=8. A detailed analysis of how to set l_1 and l_2 will be discussed in the next section when we deal with prefix based table partitioning.

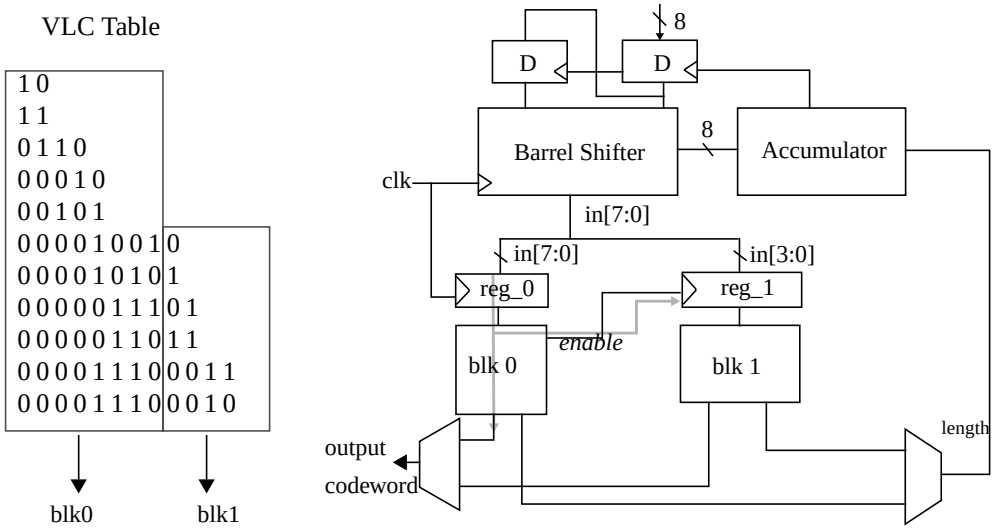
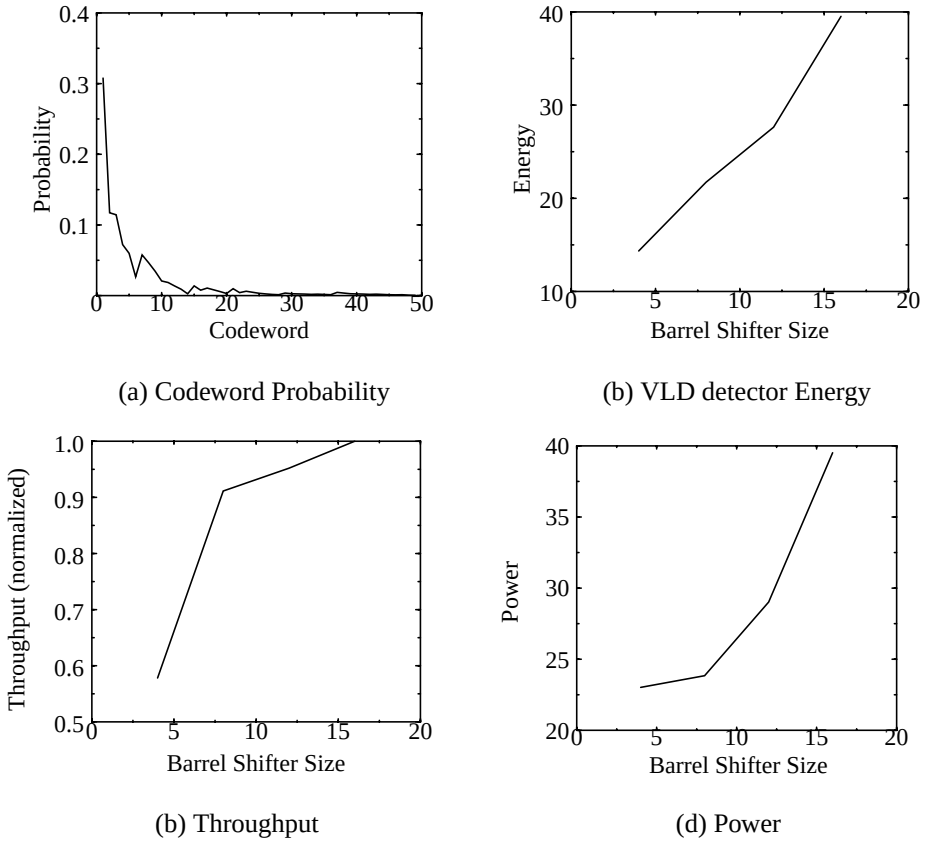


FIGURE 4.3: VLC Detector Based Table Partitioning

Optimization

The next graphs show the result of the previous equations, for the cases when Y_b=4,8,12,16. MPEG-2 DCT sequence is used for the simulation. The codeword probability is shown in the Figure 4.4 (a). The second graph shows the energy of the VLC detector as we increase the barrel shifter size (equation (4.4)). Throughput is shown in the next graph, where equation (4.3) is plotted. The last graph shows energy when the throughput is the same (i.e. Energy / throughput) and this is where the optimum solution lies. What's shown in (b) is the energy at different throughput. We must look at the energy consumption at the same throughput. The optimum energy we get from the graph is Y_b = 4 or 8. We choose Y_b=8 for two reasons. First, at the same power consumption, the "Y_b=4 system" has to run at twice the frequency of "Y_b=8 system" to achieve the same throughput. Under

a low supply voltage, we prefer a slow frequency to a higher frequency. Second, we want byte wide output from the barrel shifter. The MPEG-2 bitstream consists of not only VLCs but also fixed length codes. These fixed length codes are usually byte aligned and it is easier to process them if Y_b=8.



(a) Codeword Probability

(b) VLD detector Energy

(b) Throughput

(d) Power

FIGURE 4.4: Finding the Optimum Size of the Barrel Shifter

4.4 Prefix Based Table Partitioning

With the size of the barrel shifter set as shown in the previous section, now comes the question of spliting the VLCs which have length bigger than the barrel shifter output. For example, if Y_b=8 and the length of the VLC is 12, the VLC can be decomposed into one of the following combinations: (4,8), (5,7), (6,6), (7,5), (8,4), where (l_1,l_2) denotes the length of the first and second part of the decomposed VLC. Consider two cases: first when there is a prefix common to the long VLCs and second, when such a prefix doesn't exist.

In most cases when the VLC table is large, there are some prefixes that are com-

mon to the long VLCs. For example, the MPEG-2 DCT table has 9 different prefixes that are common to a total of 90 VLCs. In such a case, the VLC decomposition is straightforward. It is decomposed by its prefixes. The power and size of the table is also reduced.

If such a prefix does not exist, it is best to have the minimum possible length in the first block. For the case of *l=12*, and Y_b*=8*, it should be decomposed as (4,8). This is because we want the minimum power consumption in the first block (the most frequent block). The power increases as the number of input bits and hence we want minimum number of input VLC bits in the first block. The next figure shows an example of the prefix based table partitioning. The highlighted bits indicate the VLC prefixes. When a VLC without a prefix comes in, the decoding process is same as the single lookup table method. It get decoded in blk0 and the corresponding output is produced. When a VLC with a prefix comes in, the blk0 will enable one of the blocks depending on the prefix it has decoded.

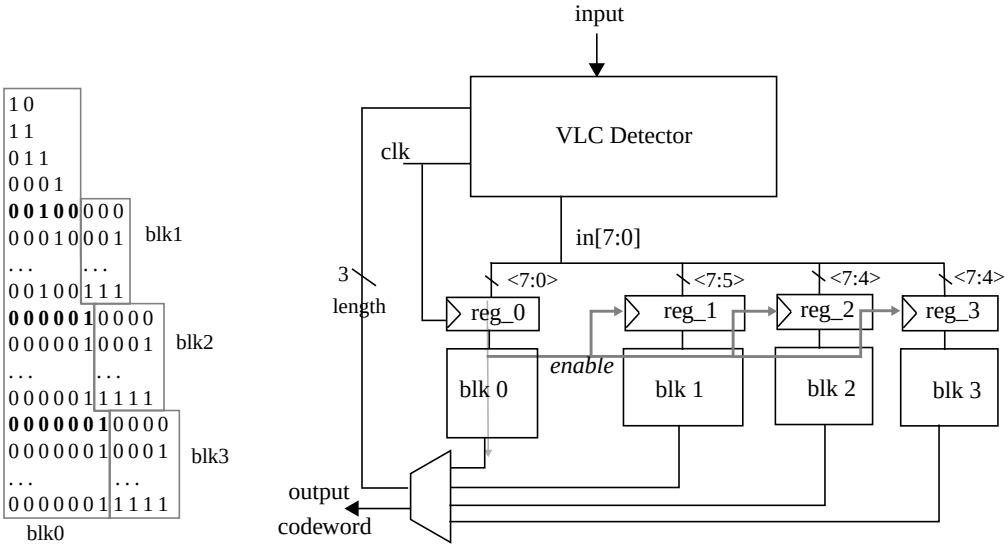


FIGURE 4.5: Prefix Based Table Partitioning

4.5 Fine Grain Table Partitioning

This section introduces table partitioning with respect to the number of VLCs in the decomposed table. In other words, as opposed to the previous two methods where we were partitioning the table by decomposing the VLC, here we deal with how many codewords should be allocated to each block [CXC97]. In the beginning, we'll start from a raw VLC table, assuming that none of the previous table partitioning has yet been applied.

The average energy consumption per codeword in the lookup table can be modelled by the following equation, where P_{cwi} is the probability that codeword i will occur and E_i is the energy required to decode the codeword i.

$$Energy = \sum P_{cwi} \cdot E_i \tag{4.7}$$

In conventional approaches, the energy required to decode a VLC did not vary much over the codeword probability (i.e., $E_i \approx E_{constant}$ for all i). Therefore, the average energy in equation (4.7) is dominated by codewords which have high probability of occurrence. Low power can be achieved if the dominant term in equation is made small, which is E_i with high P_{cwi}. With the proposed algorithm shown in Figure 4.6, the energy consumption is modelled as the following,

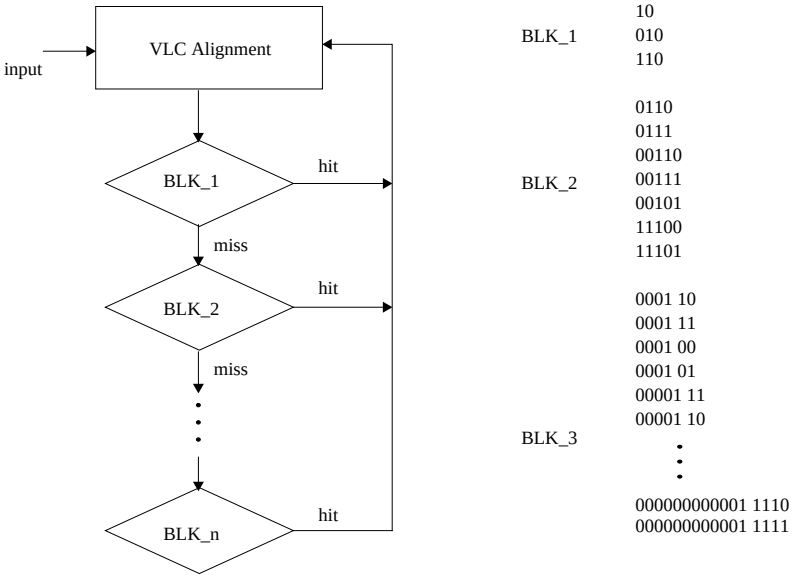


FIGURE 4.6: Fine Grain Table Partitioning

$$Energy = Pr_1 \cdot E_{H1} + Pr_2 \cdot (E_{H2} + E_{M1}) + \ldots + Pr_n \cdot \left(E_{Hn} + \sum_{i=1}^{n-1} E_{Mi} \right) + E_{overhead}(n) \tag{4.8}$$

where Pr_i is the probability that block i is a hit (i.e. a match was found in block i), E_{Hi} is the energy dissipated by block i when there is a hit and E_{Mi} is the energy when there is a miss. $E_{overhead}(n)$ represents the energy overhead of the table partitioning when the table

is decomposed into *n* groups. The overhead includes from clock generation, output multiplexers and input latches.

Now the question comes down to how to partition the table, so that the above equation has the minimum energy. There are many ways to split this table. In fact, for *n* number of VLCs, we can decompose the table up to *n* groups, and for each decomposed block, there exist many ways to how we allocate the VLCs. Obviously, we cannot consider for all these cases. To make things simpler, we'll make somewhat obvious assumptions for low power table partitioning.

Given the codewords $A_1,A_2,\ldots A_N$, their occurring probability $P_1,P_2,\ldots P_N$, and the corresponding length $L_1,L_2,\ldots,L_N$, the following conditions hold.

1) $$P_1 \geq P_2 \geq P_3 \geq \ldots \geq P_n \qquad \sum P_i = 1 \tag{4.9}$$

2) $$L_1 \leq L_2 \leq L_3 \leq \ldots \leq L_n \tag{4.10}$$

Assumption: All the codewords in the decomposed table will have its preceding and following codewords in the same table except for the first codeword and the last codeword. The first and last codeword in the table will only have its following and preceding words in the table respectively. That is, a decomposed table with *j* codewords will be of the form (A_i, A_{i+1}, A_{i+2}, ..., A_{i+j}). (i.e. There won't be a table that has codewords such as (A1, A2, A8, A3).)

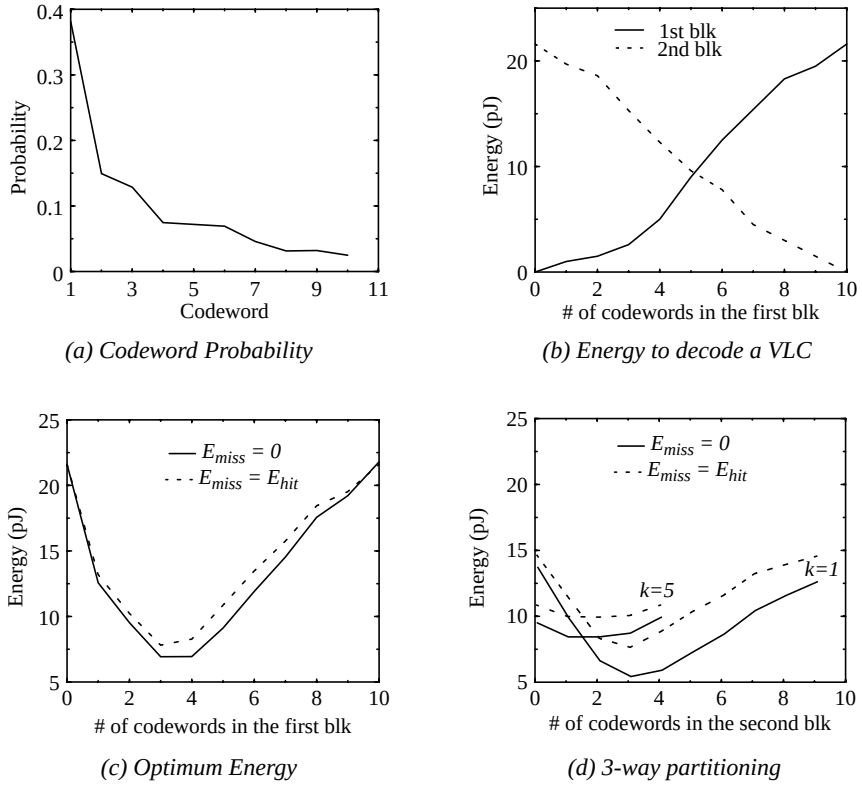
This is rather obvious since we want the least power in the first block. Thus we don't want any long length VLCs in the first block. Since the codewords are ordered according to their probability and length, we want to cluster them in the same order as described.

Now, even with the above conditions, there are still many ways to decompose the table. For convenience, let's first take an example of splitting the table into two blocks and continue on with the further analysis.Let's take a simple example of a ten codeword VLC table shown in Figure 4.6. In the case of this binary partitioning, the question is how many codewords should be assigned to the first block to achieve minimum energy. With the pro-

posed algorithm, equation (4.8) reduces down to the following equation.

$$Energy = Pr_1 E_{H1} + (1 - Pr_1)(E_{H2} + E_{M1}) \quad (4.11)$$

For this binary partitioning analysis we'll ignore the overhead energy. The probability of occurrence for each codeword is shown in Figure 4.7(a). Assuming that the first *n* codewords are in the first block, the energy required to decode a codeword in the first and the second blocks is shown in Figure 4.7(b).



(a) Codeword Probability *(b) Energy to decode a VLC*

(c) Optimum Energy *(d) 3-way partitioning*

FIGURE 4.7: Result of a Binary Partitioning

In general, determining a specific value for E_{hit} and E_{miss} is not an easy task. These energies will vary depending on how the block is implemented. For blocks implemented in PLA or ROM, $E_{miss} \approx E_{hit}$. In general, it is reasonable to assume that E_{miss} will be same as or less than E_{hit}, since the switched capacitance will typically be larger in case of a hit than a miss. Hence, considering this variation of E_{miss}, let's assume $0 < E_{miss} \leq E_{hit}$ and solve

equation (4.11) for two extreme cases: $E_{miss}=0$ and $E_{miss} = E_{hit}$. Each will represent a lower and an upper bound for equation (4.11). Using the data from Figure 4.7(b), we can numerically solve equation (4.11) and find the optimum number of codewords *n,* that should be allocated to the first block for minimum energy. The results are shown in Figure 4.7(c). The graph shows total energy vs. *n,* the number of codewords in the first block. The energy is calculated for two cases: $E_{miss}=0$ and $E_{miss} = E_{hit}$. The actual energy plot will lie somewhere between the two curves. As can be seen in the graph, the energy does not depend much on E_{miss}. For an optimum partitioned table, this is rather an obvious result since the block size and its energy will be increasing as the codeword goes through the series of blocks and thus, E_{miss} of previous block will be negligible compared to E_{hit} or E_{miss} of the next block. In this example, the optimum number of codewords that we get is around *n = 3 or 4.* This 2-way table partitioning can be used again to further partition the LUT. An *M*-way partition is also possible. Figure 4.7(d) shows the energy plot for 3-way partitioning, where *k* represents number of codewords in the first block.

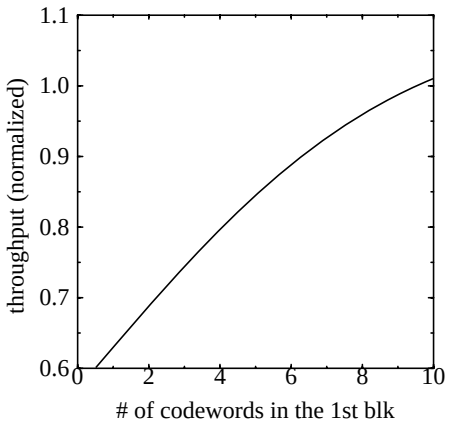
Before we go further into the M-way partition, there is an important issue which has not yet been considered: the throughput. When implemented in circuits, each conditional branch that the VLC goes through in Figure 4.6 is a critical path. We have to determine how many cycles it would take for the VLC to go through the series of decomposed blocks. Let's consider two cases, one which goes through all the critical path in one cycle, and the other which goes through one block per cycle. The throughput can be represented as follows, where equation (4.12) shows the first case and (4.13) shows the latter one.

$$f_{out} = f_a \tag{4.12}$$

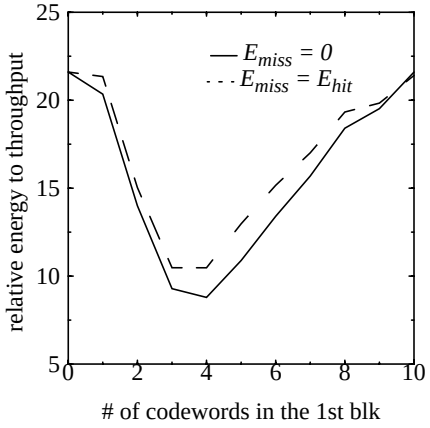
$$f_{out} = (Pr_1 + 2 \cdot Pr_2 + 3 \cdot Pr_3 + \ldots + n \cdot Pr_n)^{-1} \cdot f_b \tag{4.13}$$

Again, f_{out} is the output codeword throughput (codewords/sec) and f_a, f_b is the operating frequency of the barrel shifter for each case. It can be seen that (4.13) achieves higher throughput than (4.12). In (4.12), the throughput is determined by the worst case, which is when the least probable VLC comes in and all the decomposed blocks result a miss except for the last block. The frequency has to be slow enough so that the comparison of all these

block can be done in one cycle. Under a low supply voltage, the circuit becomes very slow and having too long a critical path is not acceptable. Hence the whole system clock has to be slowed down. Using this scheme only a small portion of the cycle will be used most of the time. On the other hand, in equation (4.13), the clock frequency is determined by the maximum delay of only one block. Hence, equation (4.13) achieves higher throughput than (4.12). Therefore, we choose the second scheme. Figure 4.8 shows the result of the minimum energy graph when the throughput is considered.



(a) throughput



(a) relative energy

FIGURE 4.8: Throughput and Relative Energy of Binary Partitioning

The optimum solution we get from Figure 4.8 is *n=3~4.* This doesn't show much difference to the previous analysis when the throughput is not considered. This is because the throughput at n=3 and n=4 doesn't vary much in the binary partition.

Now let's consider the case of *M*-way partitioning. We'll find the minimum energy with regard to the throughput. The results of *M*-way partitioning are shown in Figure 4.9. It is obtained from a simulation which takes all combinations of *M*-way partition. The minimum energy for each partitioning is plotted. The energy in the block is modelled by the average energy in a combinational logic, with the number of input bits and codewords taken into account. The equation for the energy is given as the following,

$$Energy(x,y) = \sum_{i=x}^{x+y-1} E(i) \qquad E(i) = K_{out}(i) \cdot g(K_{in}(i)) \cdot E_{gate} \tag{4.14}$$

where *Energy(x,y)* is the energy consumption of a block which has *y* codewords, starting from the *x*-th codeword in the initial single table. *E(i)* is the energy consumption of the codeword *i* when it is implemented in combinational logic. *E(i)* is further represented by the right equation in the above, where $K_{out}(i)$ is the number of output bits, $g(K_{in}(i))$ is the number of gates when $K_{in}(i)$ is the number of input bits, and E_{gate} is the average energy of the gate when random inputs are coming in. Hence *E(i)* is the energy of a codeword which has $K_{out}(i)\ g(k_{in}(i))$ number of gates. Here we are assuming that there aren't any sharing of common product terms in implementing the logic.

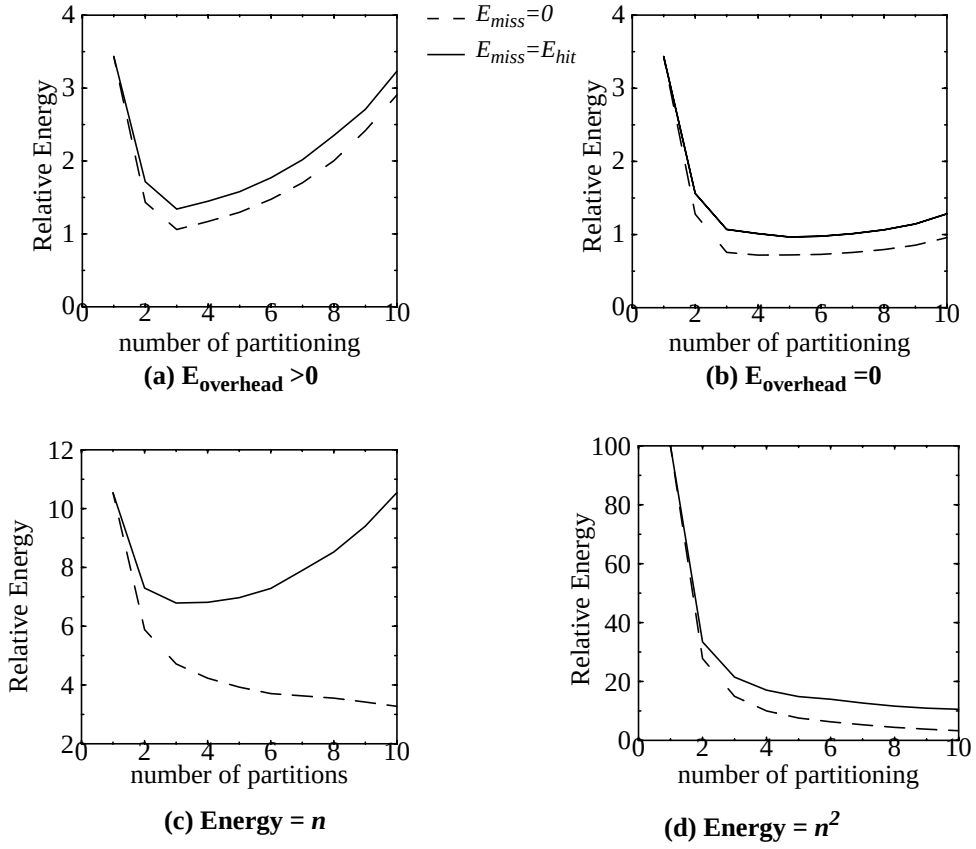
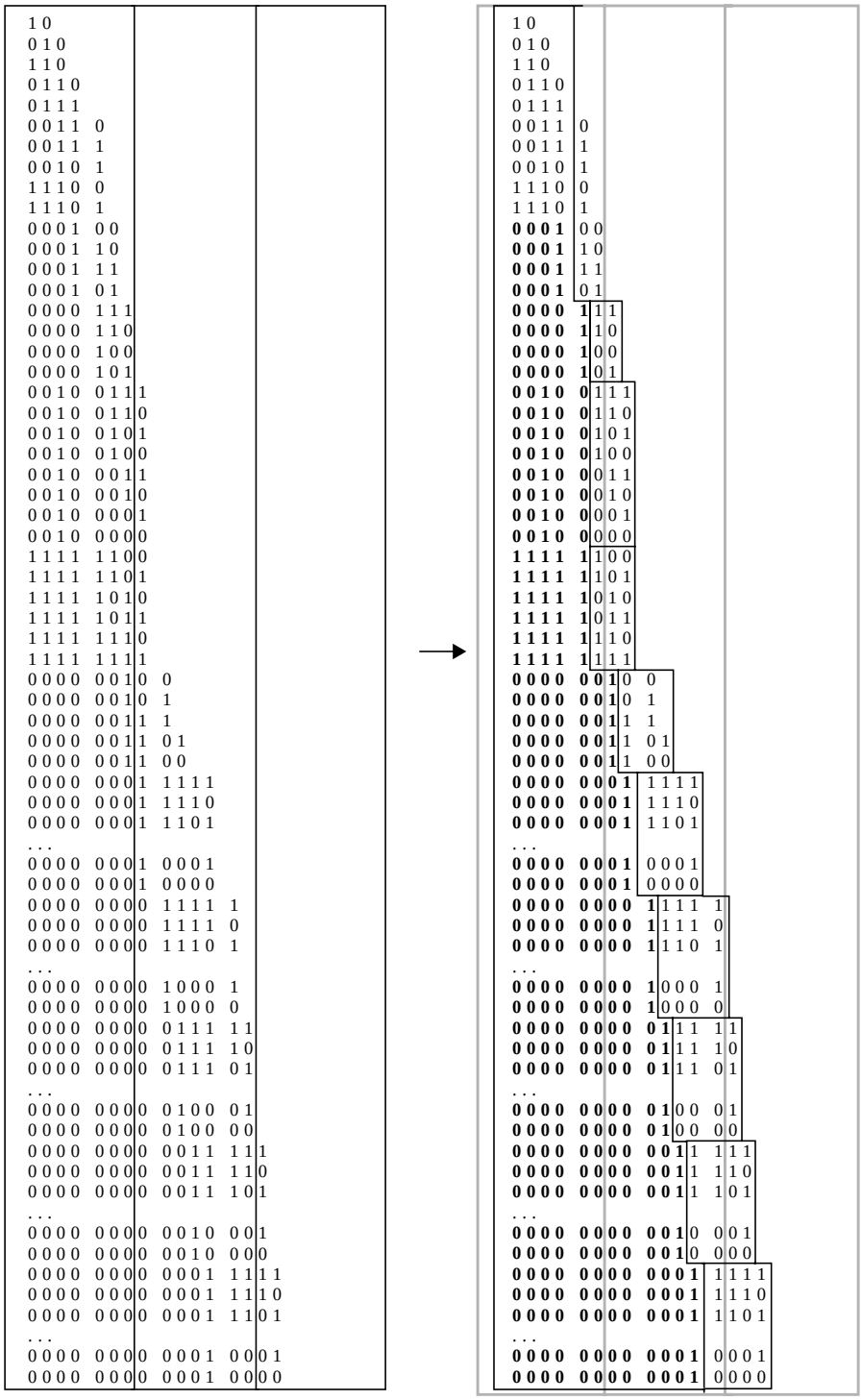


FIGURE 4.9: Result of M-way partitioning

Figure 4.9(a) shows the energy when $E_{overhead}$ is bigger larger than 0. Simulation Results show that about 20% of the first block energy is added as the partitioning increases. Figure 4.9(b) shows the result of the same equation except that now $E_{overhead}$ =0. Both (a) and (b) use the energy model in equation (4.14). As can be seen, the optimum number of partitioning is 3 for (a) and 3~8 for (b). Figure 4.9(b) has a broad range of optimum points and hence we choose 3 as the optimum number of partitions as to account for both cases. The next figures (c) and (d) show results when the energy increases with n and n^2, respectively. Although this is a poor assumption for the energy, the simulation result is included for reliability purposes. We will look at how the optimum number of partition changes as the energy profile varies. This is to account for the error in the energy models we have. Same reason is applies to why we have two cases of $E_{overhead}$ for (a) and (b). In Figures 4.9(c) and (d), it is assumed that $E_{overhead}=0$. Again the optimum point lies around 3. In Figure 4.9(d) the energy does not vary much when the number of partitioning exceeds 3. In these graphs, we should look at $E_{miss} = E_{hit}$ case, since we're dealing with random combinational logic.

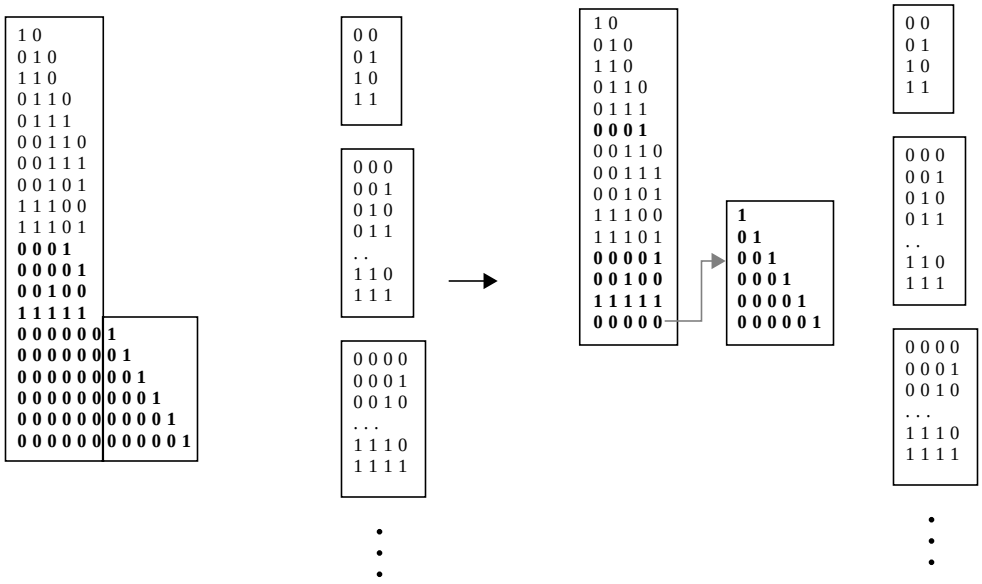
4.6 Review of Table Partitioning

Figure 4.10 shows an example of VLC table partitioning based on the three steps discussed earlier. The initial raw table consists of 113 VLC codewords. The maximum VLC length is 16. First, the table is partitioned into three blocks, (Figure 4.10 (a)) by the maximum length of the barrel shifter output to the lookup table, which is 7. (Although the barrel shifter output is 8 bits, the most significant bit is used for the sign bit). Next the VLCs are decomposed according to their prefix. The bold characters indicate the prefix of the VLC. The order which these two steps are performed can be switched, since the VLC changes its form after the prefix based decomposition. Figure 4.10 (c) shows the result of the decomposed table after these two partitioning technique has been applied. Finally, the decomposed table is further partitioned by allocating optimum number of VLCs per each block. In this example, binary partitioning has been applied. Figure 4.11 shows the schematic of this diagram.

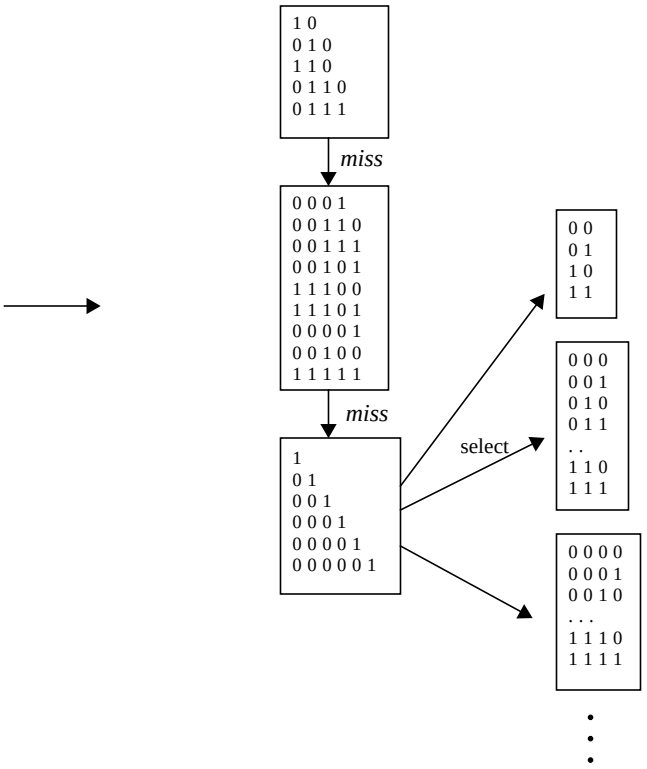


(a) VLC Detector Based Partitioning (b) Prefixed Based Partitioning

FIGURE 4.10: Example of Table Partitioning



(c) Result of Prefix and VLC Detector Table Partitioning



(d) Fine Grain Non Uniform Table Partitioning

FIGURE 4.10: Example of Table Partitioning

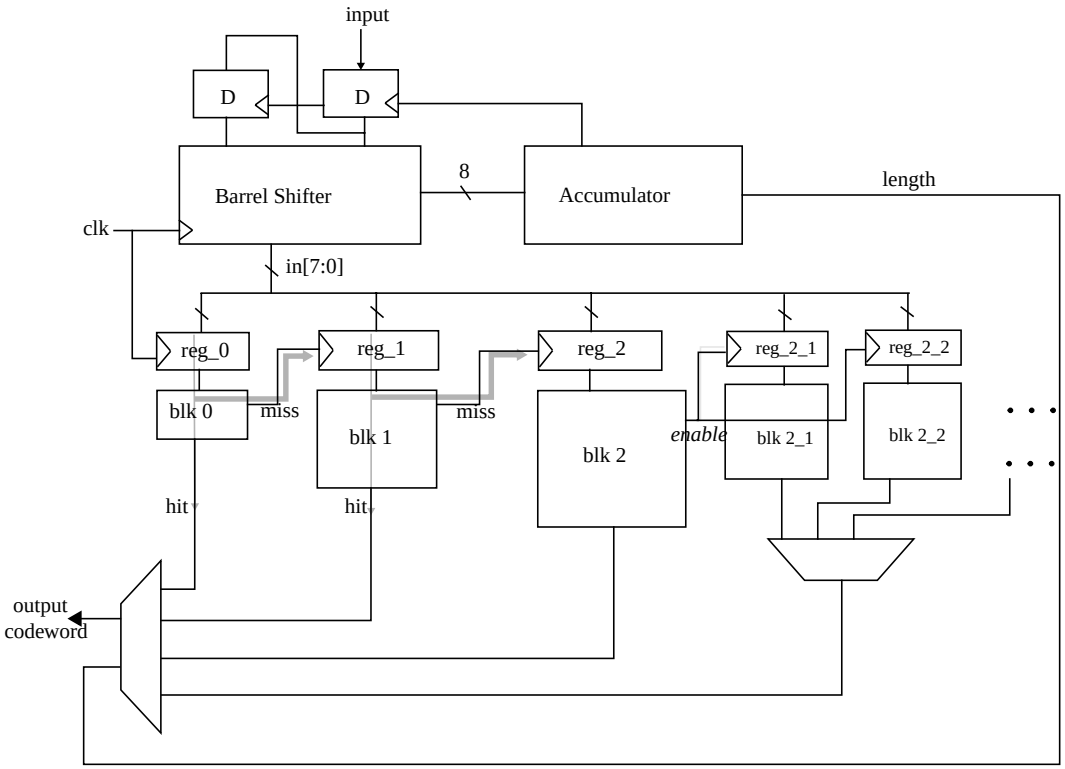


FIGURE 4.11: Schematic of a Low Power VLD with Fine Grain Table Partitioning

4.7 Other Table Reduction Methods

4.7.1 Sign Extended VLC Tables

In the MPEG-2 VLC tables, there are tables which have sign extension. The sign extended VLCs have the following format: $a_n a_{n-1}..a_1 a_0 s$, where each a_i represents a bit in the VLC, and s denotes the last bit of the VLC, which is the sign of the output codeword. Figure 4.12 shows an example of a VLC code with the sign extension. In the MPEG-2 VLC table, the DCT and motion vector VLCs have this sign extension.

An easy way to decode this VLC table would be to implement the lookup table for both cases of s=1 and 0. But, this would double the area and power of the lookup table. Another way is to think of the whole VLC (without the last bit s) as a prefix and go through the prefix based table partitioning scheme. In this case, the area and power increases by only a small amount. On the other hand, the throughput decreases, since we always need an extra cycle for the sign extension bit. An efficient way to decode the VLC

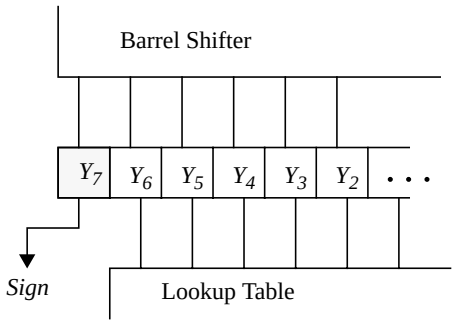
with sign extension is proposed, without losing area, power or throughput. The idea is to decode the sign bit with the next VLC at the same time. Since the sign is the least significant bit of the VLC, it will always be placed one bit prior to the start of the next VLC. Hence, instead of reading the VLC input from the most significant bit of the barrel shifter output, the lookup table will receive the bits from the second most significant bit. Figure 4.12 shows the example of VLC with sign extension and how it's efficiently decoded.

VLC	Output
:	
00011	-3
0011	-2
011	-1
1	0
010	1
0010	2
00010	3
:	

→

VLC	Output	Sign
1	0	s
01s	1	s
001s	2	s
0001s	3	s
:	:	:

(a) VLC with Sign Extension



(b) Decoding VLCs with sign extension

FIGURE 4.12: VLC with sign extension

4.7.2 Arithmetic Operation

Exploiting the input and output codeword relations gives additional reduction in power and area in the lookup table. In the MPEG-2 DCT coefficients, there are codewords which satisfy the following simple equation, where X represents the input VLC without the prefix and Y represents output codewords.

$$Y = 31 - X \tag{4.14}$$

In this case, it can be implemented with a simple inverter as shown in the following figure.

input		output
000000011111	(15)	10000 (16)
000000011110	(14)	10001 (17)
000000011101	(13)	10010 (18)
000000011100	(12)	10011 (19)
000000011011	(11)	10100 (20)
...	..	...
...	..	...
000000010001	(1)	11110 (30)
000000010000	(0)	11111 (31)

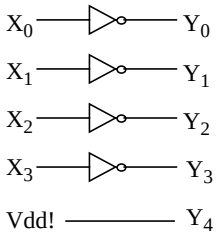


FIGURE 4.13: Example of an Arithmetic Operation to decode the VLC table

4.8 Summary

A low power table partitioning has been introduced. Three methods are applied to reduce the energy of the table. The VLD detector and prefix based partitioning was based on VLC decomposition where as the fine grain partitioning dealt with allocating optimum number of VLCs to the decomposed table. Additional table reduction methods such as VLC sign bit decoding and arithmetic operations were also introduced.

Chapter 5

Chip Implementation

This chapter explains the chip implementation of the proposed low power variable length decoder. It describes the chip input/output specifications and its interface to other MPEG-2 components. Simulation results from the layout is included.

5.1 Interface

5.1.1 Microcontroller.

The microcontroller has the information of MPEG-2 bitstream syntax and hence decodes the incoming bitstream. As mentioned earlier, the bitstream is composed of various sub-streams of different video sequence details. The microcontroller parses this bitstream according to its video bitstream syntax. However, the microcontroller cannot decode the variable length codes and therefore it needs to interact with the variable length decoder. Figure 5.1 shows the block diagram of the interface between the microcontroller and the variable length decoder (VLD). The VLD is capable of decoding all 15 different MPEG-2 VLCs. The microcontroller sends a start signal when there is a new type of VLC coming in. The VLD decodes the VLC and sends out the decoded codeword output to the corresponding MPEG-2 module. A done signal is sent to the microcontroller when the decoding is done.

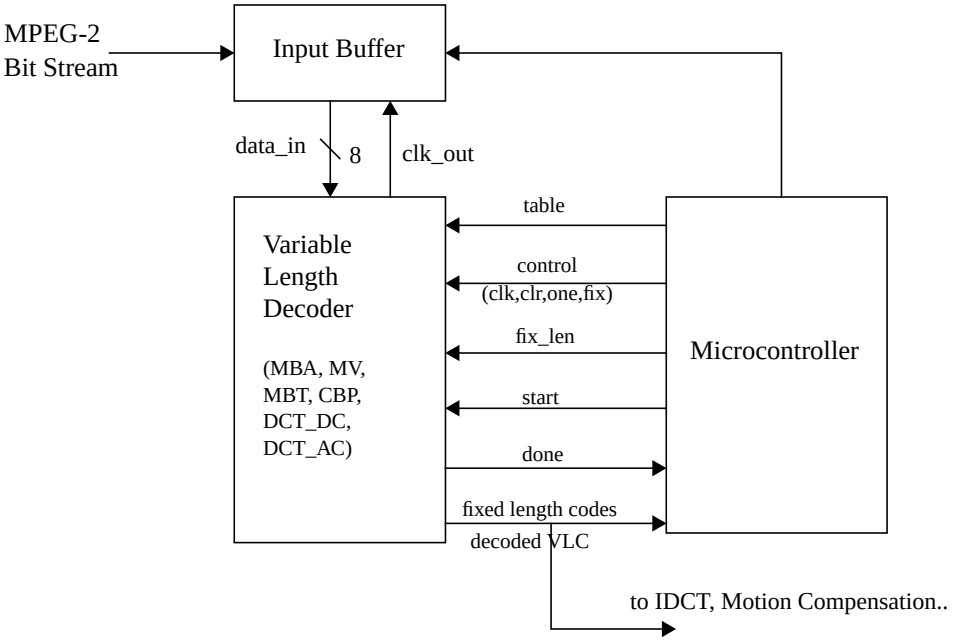


FIGURE 5.1: Overview of the VLD in MPEG-2 Decoder

An interesting thing to note in this design is that the microcontroller receives the input bit stream from the VLD, not from the input buffer. It is designed in such a way that the input bitstream is fed to the VLD. It could have been designed the other way, where the microcontroller receives the input bitstream and VLD receives its input from the microcontroller. In this case the buffer inside the VLD has to be cleared and loaded every time there is a VLC in the bitstream. This consumes additional power and clock cycles. Therefore, the VLD interface is designed as described.

5.1.2 Timing

The timing diagram is shown in Figure 5.2. The microcontroller tells the VLD what table to lookup for and sends a start signal. As soon as the VLC is decoded, the VLD sends out a done signal to the microcontroller. When there is a fixed length code, the ‘fixed’ signal is asserted high and the length is given by the microcontroller. The VLD passes the fixed length code to the microcontroller and send a done signal when it is ready to decode a new VLC. In the upper part of Figure 5.2, the bold bits indicate the variable length codes and the non-bold bits represent fixed length signal.

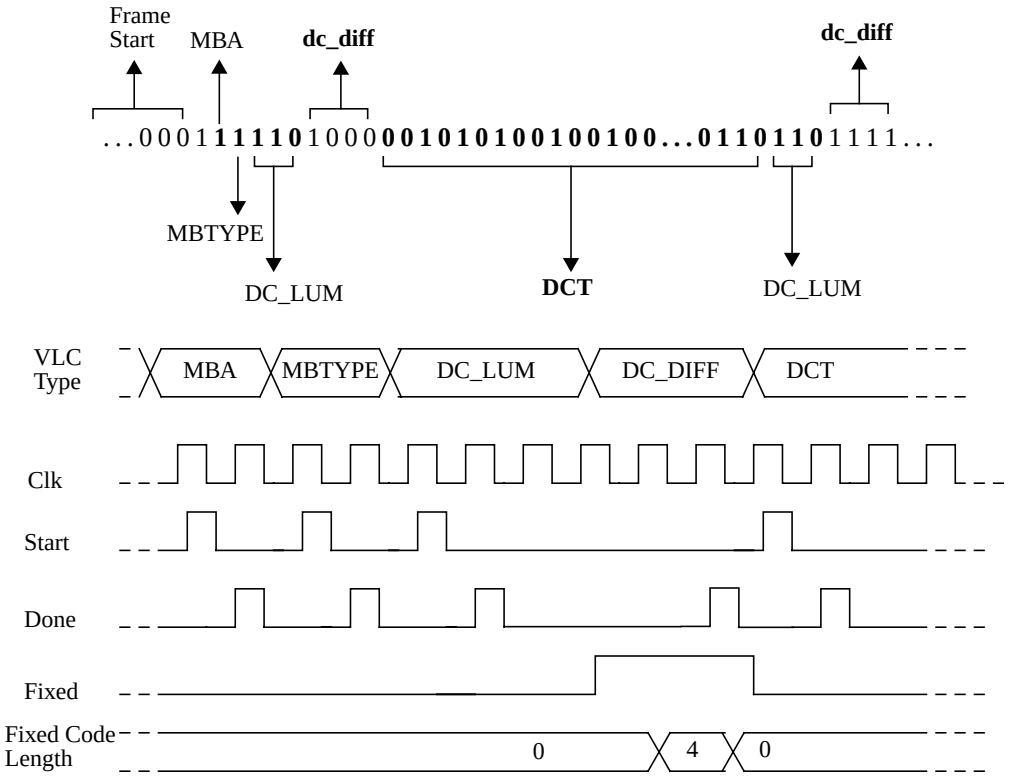


FIGURE 5.2: Timing Diagram of the VLD Chip

5.2 Signal Description

The chip has a total of 56 pins including the power and ground pins. There are 40 active signal pins, which are evenly divided by the input and output. The rest of the pins are used for power and ground.

5.2.1 Input

The following table shows the input pin specification.

input	data_in	table	fix_len	fixed	clock	clear	start	one	total
# of bits	8	4	3	1	1	1	1	1	20

TABLE 5.1: Input Pins

data_in: 8 bit input MPEG-2 bitstream from the input buffer.

table : 4 bit data from microcontroller that tells which VLC table to use. Table 5.2 shows how the table is selected.

fix_len : 3 bit length data from the microcontroller that tells the length of the fixed length code.

fixed : This signal is asserted high when there is a fixed length code in the bitstream. It comes from the microcontroller.

clock : The system clock of the variable length decoder.

start : A pulse signal from the microcontroller which tells to start decoding the VLCs.

one : When this signal is asserted high, it means that there is no change of the VLC table and the next VLC type is the same as the previous one. In this case, the VLD will keep decoding the VLC without the start pulse from the microcontroller. Figure 5.4 shows the circuit schematic of the control block regarding this 'one' signal.

table<3:0>	Table Type	ISO 13818 VLC Table
0000	Invalid	X
0001	Coded Block Pattern	B-9
0010	Motion Vector	B-10
0011	DM_Vector	B-11
0100	DCT DC Luminance	B-12
0101	DCT DC Chrominance	B-13
0110	Intra DCT Coefficients	B-14
0111	Non-Intra DCT Coefficients	B-15
1000	Macroblock Address Increment	B-1
1001	Macroblock Type - I	B-2
1010	Macroblock Type - P	B-3
1011	Macroblock Type - B	B-4
1100	Macroblock Type - I with SS	B-5
1101	Macroblock Type - P with SS	B-6
1110	Macroblock Type - B with SS	B-7
1111	Macroblock Type - SNR	B-8

TABLE 5.2: Table<3:0> Usage

5.2.2 Output

The output consists of a done signal, a buffer trigger signal and all the decoded VLC output. The MPEG-2 VLC has 15 different tables, and having independent terminal for each output results a total of 63 output pins. This is obviously not acceptable, since the number of pins on the chip is limited. Hence the output codewords are multiplexed. Table 5.4 shows how the output of each decoded VLCs are multiplexed into 18 bits.

output	out	clk_out	done	total
# of bits	18	1	1	20

TABLE 5.3: Output Pins

out : 18 bit output. The output is demultiplexed outside the VLD and sent to the corresponding output module.

clk_out : This is the carry out signal from the accumulator. It tells the input buffer that the VLD has used up all its data in the barrel shifter and needs new sets of data.

done : This signal is asserted high when the VLD has finished the decoding process and is ready to decode the next VLC.

output	**MBA**	**MBT**	**Motion**	**DC/Fixed**	**CBP**	**DCT**
17	escape		mv<4>	lum<3>		run<5>
16	mba<5>		mv<3>	lum<2>		run<4>
15	mba<4>		mv<2>	lum<1>		run<3>
14	mba<3>		mv<1>	lum<0>		run<2>
13	mba<2>		mv<0>	fixed<7>		run<1>
12	mba<1>		sign	fixed<6>		run<0>
11	mba<0>			fixed<5>		level<10>
10			dmv	fixed<4>		level<9>
9			dmv_sign	fixed<3>		level<8>
8				fixed<2>		level<7>
7		mbt<5>		fixed<1>		level<6>
6		mbt<4>		fixed<0>		level<5>
5		mbt<3>			cbp<5>	level<4>
4		mbt<2>			cbp<4>	level<3>
3		mbt<1>		chm<3>	cbp<3>	level<2>
2		mbt<0>		chm<2>	cbp<2>	level<1>

TABLE 5.4: Output<17:0> Description

output	MBA	MBT	Motion	DC/Fixed	CBP	DCT
1		mbw<1>		chm<1>	cbp<1>	level<0>
0		mbw<0>		chm<0>	cbp<0>	sign

TABLE 5.4: Output<17:0> Description

5.3 Chip

5.3.1 Circuit Schematics

Full Chip Schematic

The following figure shows the variable length decoder chip schematic. The chip can be divided into 3 large modules, the VLC detector, control logic and the lookup table. About 86% of the total area is consumed by the lookup table and the rest is consumed by the control logic and the VLC detector.

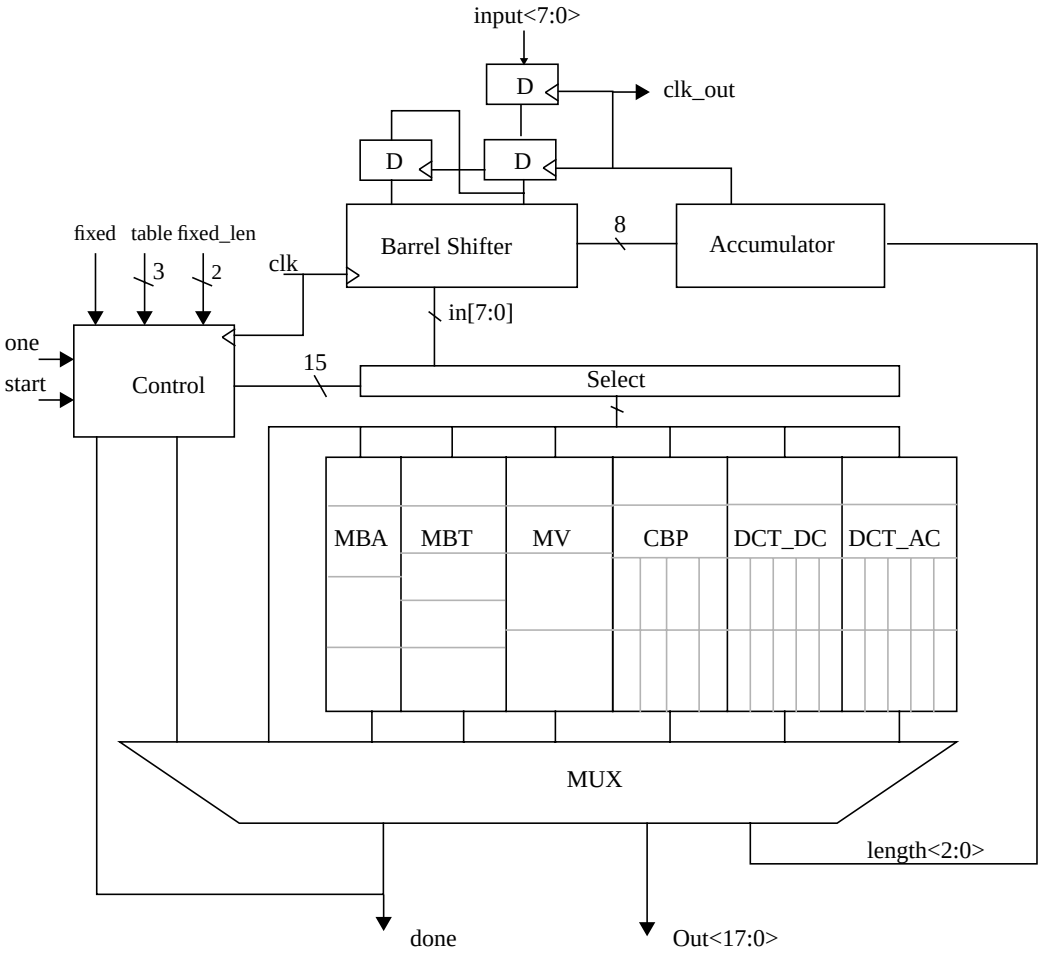


FIGURE 5.3: Chip Schematic

Control Logic Schematic

Figure 5.4 shows the simplified schematic of the control logic. When ‘one’ signal is asserted high, the VLD will generate start signal on its own. Only one of the table receives the start signal from the control logic.

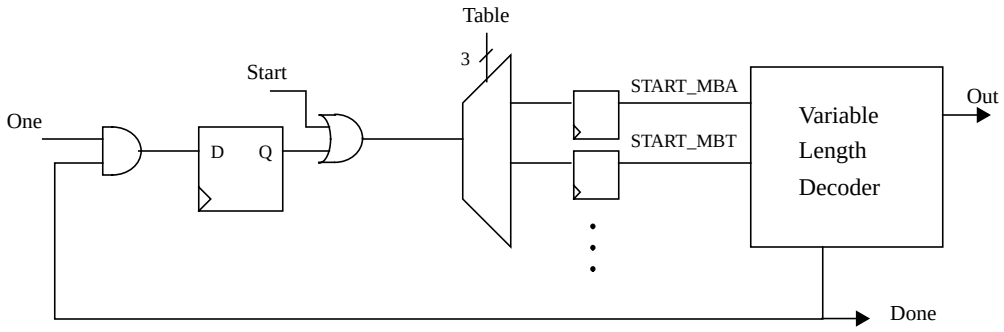
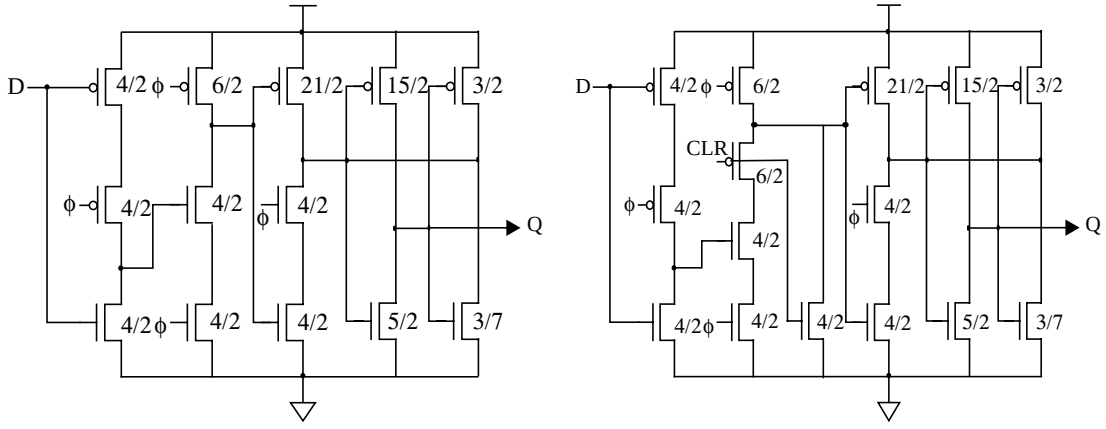


FIGURE 5.4: Control Logic of the Variable Length Decoder

Static TSPC

The following figure shows a static TSPC latch, sized to operate under 1.3V. The same implementation with asynchronous clear is also drawn. An interesting to notice in this circuit is the cross coupled inverter at the output of the TSPC latch. Without the cross coupled inverters, the output node is at a floating state. When the input clock is gated, the clock may not be switching for a long time. This causes a voltage drop at the output node due to leakage. The cross coupled inverter prevents this leakage voltage drop from happening.



(a) Static D-ff **(b) Static D-ff with Asynchronous Clear**

FIGURE 5.5: Static True Single Phase Clock D Flip Flop (Courtesy of Thucydides Xanthopolous)

5.3.2 Layout

Figure 5.6 shows the layout of the chip. The chip core is surrounded by a power-ground ring to supply the power and ground.

5.3.3 Results

The results are obtained from powermill. A 0.7mm CMOS process is used. The threshold voltage for NMOS and PMOS is about 0.7V and 0.9V, respectively.

power	area	frequency	leakage	throughput[a]
0.462mW @1.4V	5.24mm^2(core) 16.54mm^2(full_chip)	10Mhz	5.08% @1.4V	39 M samples/sec

TABLE 5.5: Simulation Results

a. throughput of DCT run/level pair (samples/sec)

5.4 Summary

A low power variable length decoder chip was implemented. It decodes all 15 MPEG-2 variable length codes and provides 39 Msamples/sec output rate to the IDCT module. The chip consumes 0.46mW at 1.4V.

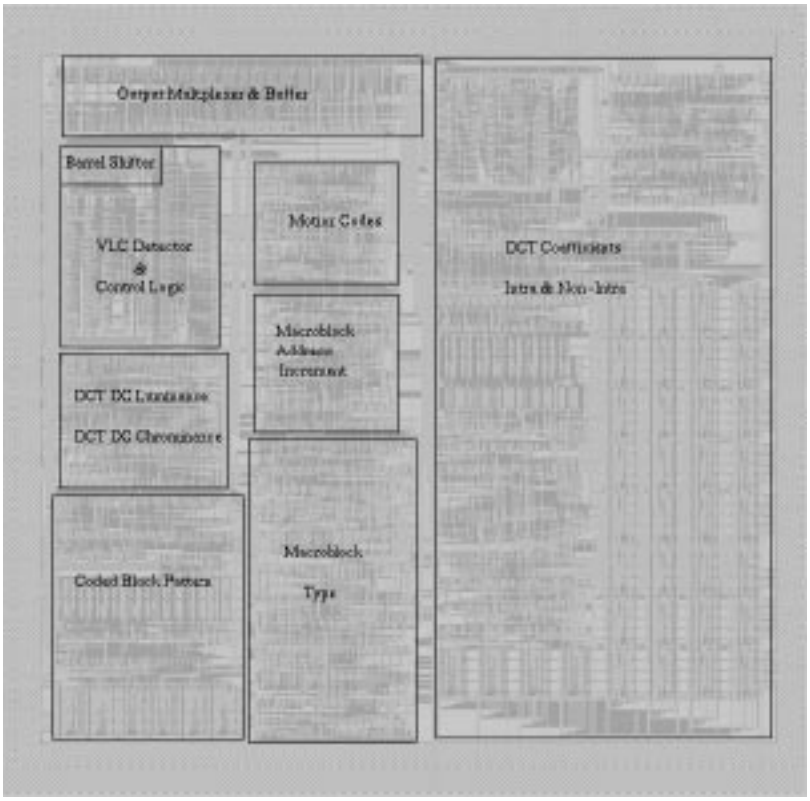


FIGURE 5.6: Chip Layout (Full Chip with Pads, Chip Core)

Chapter 6
Conclusion

A low power variable length decoder has been extensively studied. Power reduction was achieved mainly by architecture level optimizations, where we decomposed the table into several variable size blocks. The statistics of the variable length code (VLC) was exploited in the optimization process. First the variable length decoder (VLD) was decomposed into its functional units: the VLC detector and the lookup table. The power and throughput of each component was studied and minimized.

A parallel method was chosen for the VLC detector, which is capable of processing multiple bits per cycle by using a barrel shifter. It has advantage over the serial method in that it produces higher codeword output rate at the same system clock frequency. In implementing the VLC detector, a regular barrel shifter was used. The accumulator was implemented using a hybrid adder, which is composed of a mirror adder carry and a DCVSL sum.

The lookup table was partitioned into several variable size blocks according to the VLC occurring frequency. The table decomposition was done in three steps. The VLD detector based partition, prefix based partition and fine grain non-uniform table partition. The VLD detector based partition minimized the power consumption of the variable length code detector. Since only short codewords comes in most of the time, having the

VLC detector designed for the maximum length VLC is a waste of power and area. The size of the barrel shifter was optimized to give minimum power under a given throughput. The prefix based table partition greatly reduced the size of the lookup table. By clustering the VLCs according to their common prefixes, the area and power consumption in the lookup table was lowered. Finally, we decomposed the table by allocating optimum number of bits per each table. Simulation results have shown that about 3-5 level decomposition is appropriate for minimum power.

Looking from the circuit level, the lookup table was implemented in static CMOS rather than a PLA. Static CMOS was chosen to a PLA because its power performance was better for small tables.

A chip was implemented which decodes all 15 different MPEG-2 VLC tables. The chip simulation from the extracted layout resulted a 0.46mW at 1.4V with a DCT run level throughput of 39 Msamples/sec. An order magnitude of power reduction was possible compared to the conventional scheme with a single lookup table.

Appendix A

Optimum Table Partitioning Program

A. *C* Source Code of Optimum Table Partitioning

```
/************************************************
 * Finding the Optimum Number of Table Partitioning
 * for Low Power Variable Length Decoder
 *
 * Low Power MPEG2 Project
 *
 * SeongHwan Cho May 20, 1997
 * Massachusetts Institute of Technology
 * chosta@mit.edu
 ************************************************/

# include “math.h”
# include “stdio.h”
# define TOTAL 11

float prob[TOTAL];
static int b[TOTAL][TOTAL];
static int a[TOTAL][TOTAL];
static int throughput[TOTAL];
int num_of_part, num_of_rec;
float opt_en0[TOTAL];
float opt_en1[TOTAL];
static int opt_part0[TOTAL][TOTAL];
static int opt_part1[TOTAL][TOTAL];

float pcum(int x, int y);
void recur(int p, int T);
float thrput(int x[TOTAL][TOTAL], int y[TOTAL][TOTAL]);
float energy0(int x[TOTAL][TOTAL], int y[TOTAL][TOTAL]);
float energy1(int x[TOTAL][TOTAL], int y[TOTAL][TOTAL]);
float en(int x, int y);
int g(int k);

void main(void)
{

  int i,j;


prob[1]=0.381,prob[2]=0.149,prob[3]=0.128,prob[4]=0.075,prob[5]=0.072,prob[6]=.0691,prob[7]=0.0458,prob[8]=0.0
314,prob[9]=0.032,prob[10]=0.025;


  for(i=0;i<=TOTAL;i++){
    a[i][1]=1;
    opt_en0[i]=999999;
    opt_en1[i]=999999;
  }

  printf(“\n”);

  for(num_of_part=2;num_of_part<=TOTAL;num_of_part++){
    num_of_rec=0;
    recur(num_of_part,TOTAL-1);
  }
  printf(“\n 1 %f %f”,(160*en(1,10)+234)/1000,(160*en(1,10)+234)/1000);
  for(i=2;i<TOTAL;i++)
    printf(“\n %d %f %f “,i,opt_en0[i],opt_en1[i]);

  printf(“\n for matlab input”);
```

```
  printf("\n opt_en0 = [");
  for(i=2;i<TOTAL;i++)
    printf(" %f ",opt_en0[i]);
  printf("]");

  printf("\n opt_en1 = [");
  for(i=2;i<TOTAL;i++)
    printf(" %f ",opt_en1[i]);
  printf("] \n");

}

/* p : number of partitioning>1 , T : total number of codewords in the blk */

void recur(int p, int T)
{
  int i,j,k;
  float tmp0,tmp1;
  if (p==2){
    for(i=1;i<T;i++){
      b[num_of_part][num_of_rec+1]=i;
      b[num_of_part][num_of_rec+2]=T-i;

      for(j=2;j<num_of_rec+3;j++)
            a[num_of_part][j]=a[num_of_part][j-1]+b[num_of_part][j-1];

      printf("\n (");
      for(k=1;k<TOTAL;k++)
            printf(" %d ",b[num_of_part][k]);
      printf(")");
      printf("\n (");
      for(k=1;k<TOTAL;k++)
            printf(" %d ",a[num_of_part][k]);
      printf(")");

      tmp0=energy0(a,b)*thrput(a,b);
      tmp1=energy1(a,b)*thrput(a,b);
      if(tmp0<opt_en0[num_of_part]){
            opt_en0[num_of_part]=tmp0;
      }
      if(tmp1<opt_en1[num_of_part]){
            opt_en1[num_of_part]=tmp1;
      }
    }
  }
  else {
    num_of_rec++;
    for(i=1;i<T-p+2;i++){
      b[num_of_part][num_of_rec]=i;
      recur(p-1,T-i);
    }
      num_of_rec--;
  }
}

float pcum(int x, int y)
{
  int i;
  float cum;
  cum=0;
  for(i=x;i<x+y;i++)
    cum +=prob[i];

  return cum;
```

```
}

float thrput(int x[TOTAL][TOTAL], int y[TOTAL][TOTAL])
{
 int i;
 float thr;
 thr=0;
 for(i=1;i<TOTAL;i++)
   thr = thr + i*pcum(x[num_of_part][i],y[num_of_part][i]);
 return thr;
}

float energy0(int x[TOTAL][TOTAL], int y[TOTAL][TOTAL])
{
 int i;
 float eng;
 eng=0;
 for(i=1;i<TOTAL;i++)
   eng = eng + pcum(x[num_of_part][i],y[num_of_part][i])*en(x[num_of_part][i],y[num_of_part][i]);

 return (160*eng+234+0*num_of_part*60)/1000;
}

float energy1(int x[TOTAL][TOTAL], int y[TOTAL][TOTAL])
{
 int i,j;
 float eng,esum;
 eng=0;
 for(i=1;i<TOTAL;i++){
   esum=0;
   for(j=1;j<=i;j++)
     esum+=en(x[num_of_part][j],y[num_of_part][j]);
   eng = eng + pcum(x[num_of_part][i],y[num_of_part][i])*esum;
 }
 return (160*eng+234+0*num_of_part*60)/1000;
}

float en(int x, int y)
{
 int i,j;
 float eng;
 eng=0;
 if (y)
   for(i=x;i< y+1;i++)
             eng=eng+g(i);
 return eng;
}

int g(int k)
{
 switch (k)
   {
   case 1: return 1; break;
   case 2: return 1; break;
   case 3: return 1; break;
   case 4: return 2; break;
   case 5: return 2; break;
   case 6: return 2; break;
   case 7: return 2; break;
   default: return 3; break;
   }
}
```

Appendix B

Circuit Schematics

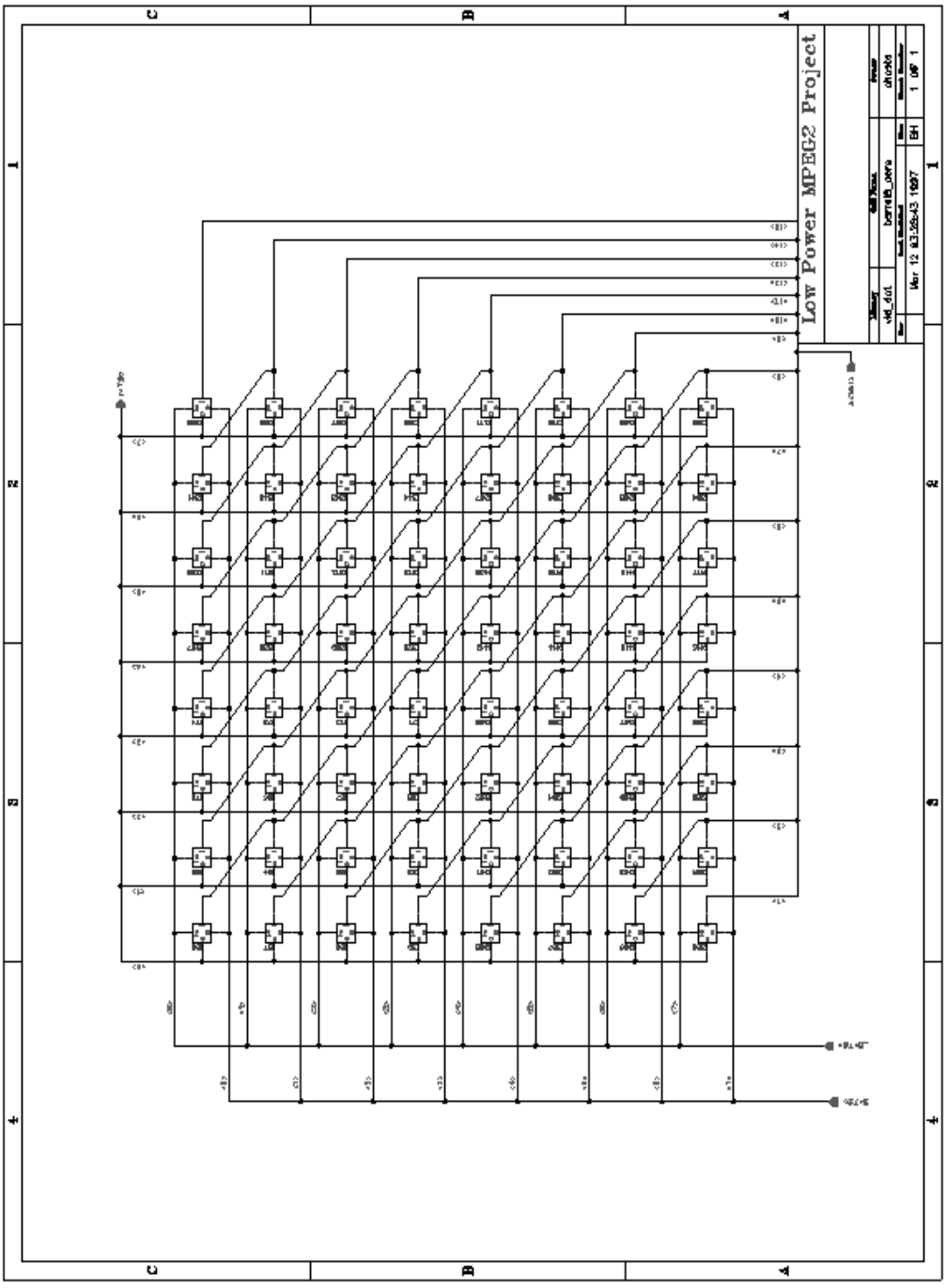


FIGURE B.1: 8 Bit Barrel Shifter

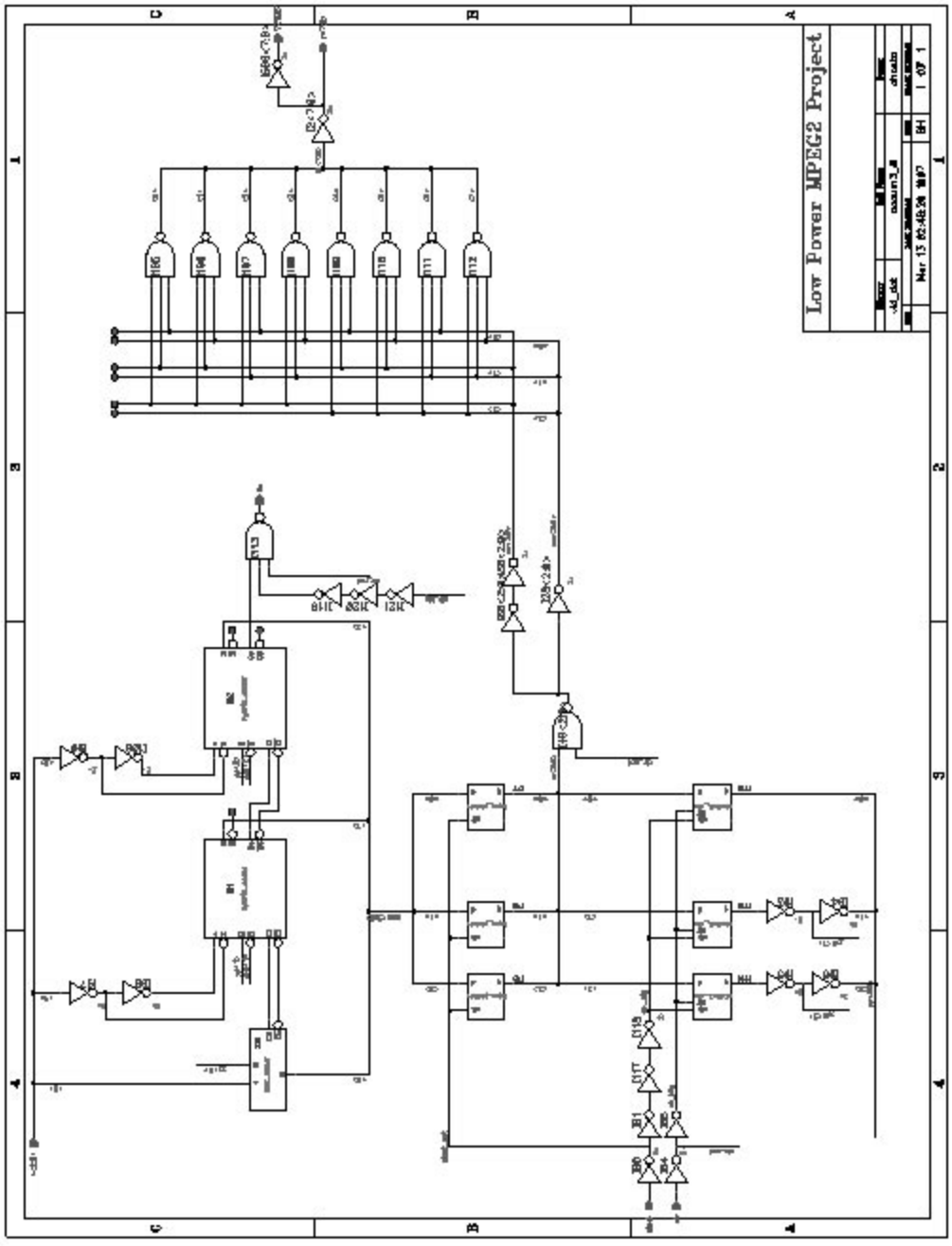


FIGURE B.2: Accumulator

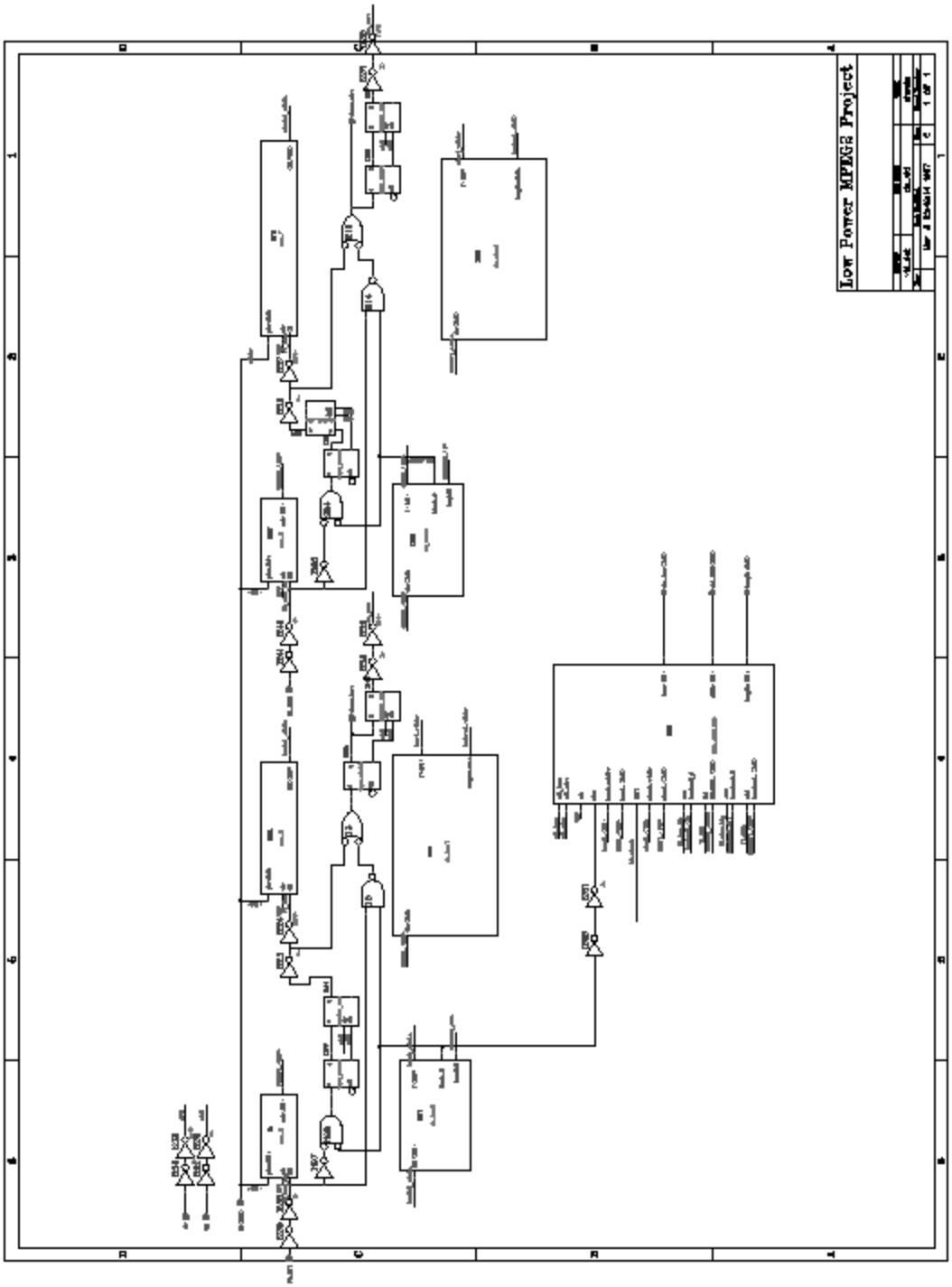


FIGURE B.3: DCT DC Luminance & Chrominance VLD

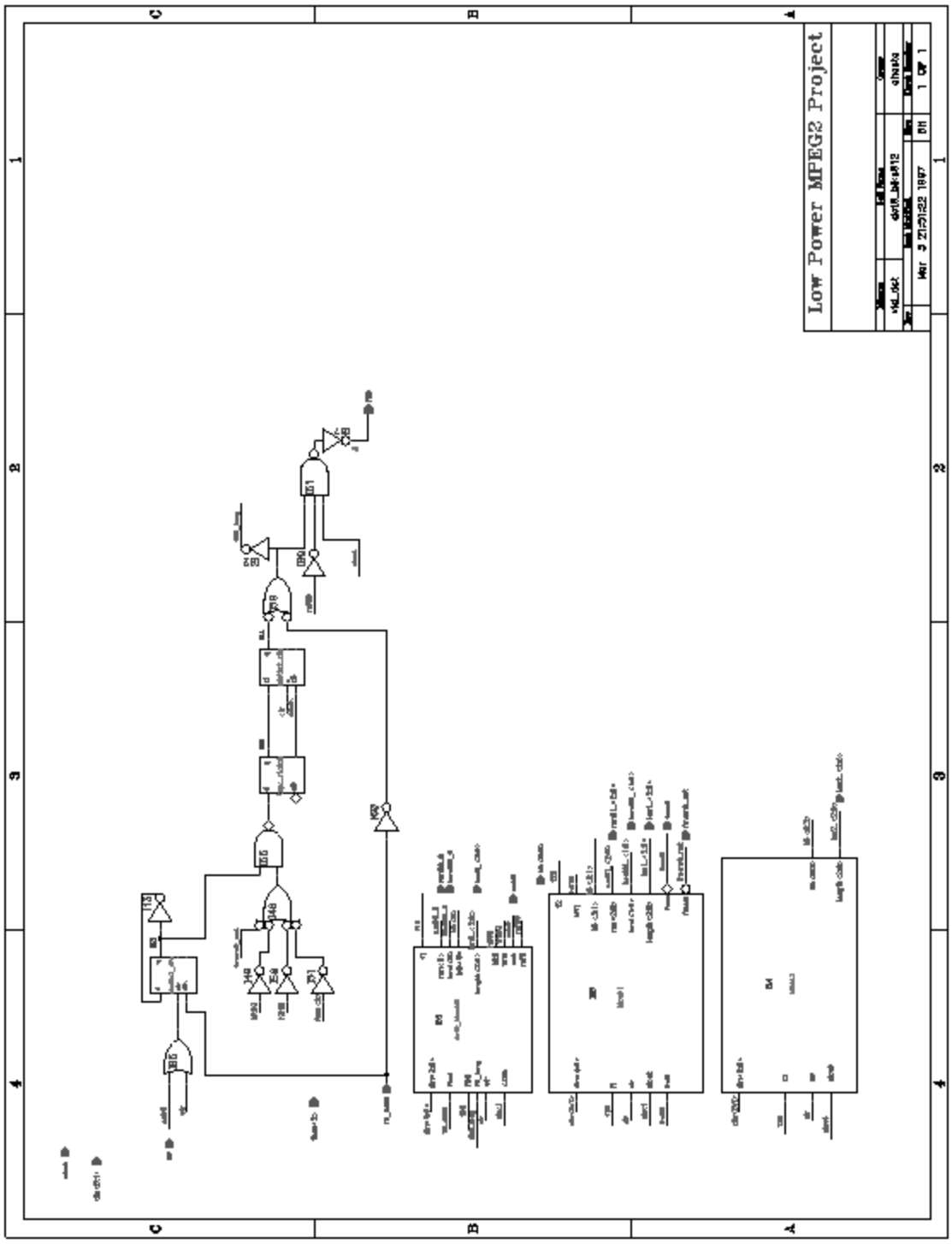


FIGURE B.4: Block Diagram of Table Partitioned DCT VLD

Bibliography

[C-c95] C-cube Microsystems, CL9100 Multimode Video Decoder User's Manual, *C-Cube Microsystems*, 1995

[CL94] S.B. Choi and M.H. Lee, "High Speed Pattern Matching for a Fast Huffman Decoder," *IEEE Transactions on Consumer Electronics*, vol. 41, no. 1, pp.97-103, Feb. 1995.

[CM92] S. F. Chang and D. G. Messerschmitt, "Designing high-throughput VLC decoder Part I - Concurrent VLSI architectures," *IEEE Transactions on Circuits and Systems for Video Technol*ogy, vol. 2, no. 2, pp. 187-196, June 1992.

[CXC97] SeongHwan Cho, Thucycides Xanthopolous, and Anatha P. Chandrakasan, "Design of a Variable Length Decoder using Fine Grain Non-Uniform Table Partitioning," *IEEE International Symposium on Circuits and Systems*, June 1997.

[Erc85] M.D. Ercegovac, *Digital Systems and Hardware/Firmware Algorithms*, New York Wiley, 1985.

[Hash94] R. Hashemian, "Design and hardware construction of a high speed and memory efficient Huffman decoding," *IEEE International Conference on Consumer Electronics*, pp.74-75,1994.

[Huf52] D.A. Huffman, "A method for the construction of minimum redundancy codes," *Proceedings of IRE*, vol. 40, no. 10, pp. 1098-1101, Sept. 1952.

[Jan92] Y. J. Jan, "A high speed variable length decoder for digital HDTV systems," *Signal Processing of HDTV IV*, pp. 333-340, 1992.

[Ker93] Roger G. Kermode, Requirements for real time digital video compression. Technical Report, M.I.T. Media Laboratory, July 1993.

[KS94] E. Komoto and M. Seguchi, "A 110Mhz MPEG-2 variable length decoder LSI," *Proceedings of Symposium on VLSI circuits*, pp. 71-72, 1994.

[LS91] S. M. Lei and M. T. Sun, "An entropy coding system for digital HDTV ap-

plications," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 1, pp.147-155, Mar. 1991.

[MC80] Carver Mead and Lynn Conway, *Introduction to VLSI systems*, Addison-Wesley, 1980

[MPEG-2] Recommendation H.262, ISO/IEC 13818, "Generic coding of moving picture and associated audio," *Draft international standard*.

[MRB91] A.Mukherjee, N. Ranganathan, and M. Bassiouni, "Efficient VLSI designs for data transformations of tree-based codes," *IEEE Transactions on Circuits & Systems*, pp. 306-314, 1991.

[OTD94] Yasushi Ooi, Atsushi Taniguchi, Shigeki Demura, "A 162 Mbit/s Variable Length Decoding Circuit using an Adaptive Tree Search Technique," *IEEE Custom Integrated Circuits Conference*, pp.107-110, 1994.

[Rab95] Rabaey, *Digital Integrated Circuits: A Design Perspective*, Prentice Hall, 1995.

[RW96] Mikael Rudberg, Lars Wanhammar, "New Approaches to High Speed Huffman Decoding," *IEEE International Symposium on Circuits and Systems*, pp.149-152, 1996.