A Low-Power Design of Motion Estimation Blocks for Low Bit-Rate Wireless Video Communications

Steve Richmond

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Electrical Engineering

Dr. Dong S. Ha, Chairman
Dr. James R. Armstrong
Dr. Jeffrey H. Reed

March 2001
Blacksburg, Virginia

Keywords: Motion estimation, H.263, Low power, Four-step search

A Low-Power Design of Motion Estimation Blocks for Low Bit-Rate Wireless Video Communications

Steve Richmond
Dr. Dong S. Ha, Chairman
Bradley Department of Electrical and Computer Engineering

(ABSTRACT)

Motion estimation and motion compensation comprise one of the most important compression methods for video communications. We propose a low-power design of a motion estimation block for a low bit-rate video codec standard H.263. Since the motion estimation is computationally intensive to result in large power consumption, a low-power design is essential for portable or mobile systems. Our block employs the Four-Step Search (4SS) method as its primary algorithm. The design and the algorithm have been optimized to provide adequate results for low-quality video at low-power consumption. The model is developed in VHDL and synthesized using a 0.35 um CMOS library. Power consumption of both gate-level circuits and memory-accesses have been considered. Gate-level simulation shows the proposed design offers a 38% power reduction over a "baseline" implementation of a 4SS model and a 60% power reduction over a baseline Three-Step Search (TSS) model. Power savings through reduction of memory access is 26% over the TSS model and 32% over the 4SS model. The total power consumption of the proposed motion estimation block ranges from 7 - 9 mW and is dependent on the type of video being motion estimated.

Acknowledgements

I would like to thank Dr. Dong Ha for his guidance in conducting this research and making my graduate school experience a very worthwhile and enjoyable one. It has been a great pleasure working with him over the past couple of years. I would like to also express my appreciation to Dr. James Armstrong and Dr. Jeffrey Reed for agreeing to serve on my graduate advising committee.

I have enjoyed working with Nate August, Suk Won Kim, Jos Sulistyo, and all of my fellow researchers in the Virginia Tech VLSI for Telcommunications (VTVT) Laboratory. It has been a tremendous learning experience collaborating with them and I wish them the best of luck in their future endeavors.

I simply would not have accomplished all that I have without the support of my parents. I will forever be in debt to them for unconditional love and support. My deep appreciation also to my brothers, Michael and Matthew, for being great friends and always knowing how to get me to laugh.

And a special thank you to my girlfriend, Tori Johnson, for making these last few months a little easier. Her understanding and patience have been amazing and have provided much needed confidence and inspiration in finishing this work.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

The transmission of video over portable devices has yet to penetrate the mainstream technology markets. This is primarily due to the fact of the high bandwidth required by video in its raw form. To reduce the bandwidth requirements, video compression must be performed. Recent advances in video compression methods recommended in MPEG-4 and H.261/263 have made feasible video transmission at low bit-rates.

Applications of a low bit-rate video transmission system range from portable videophones to wireless surveillance systems to mobile patrols. They have the similar operating requirement of continuous operation for a long period of time without recharging the battery. This is a critical necessity for mobile patrol and surveillance equipment. Since video compression is computationally intensive, low-power implementations are critical for widespread acceptance and use of these devices.

The applications mentioned above do not require perfect picture quality, nor is it feasible in current technology. Instead, the quality of video can be degraded to ease the bandwidth requirements. As such, the H.263 recommendation is the target of our system investigated in this thesis. The H.263 standard allows for degradation in video quality to meet the low-bandwidth requirement.

One of the most important techniques used to reduce the bandwidth requirements of the data stream is Motion Estimation and Motion Compensation. Motion Estimation is defined as the encoder searching a previous block of pixels to determine where a block of pixels in the current frame has moved. If successful, the encoder via Motion Compensation can then transmit only the difference information between the two frames across the transmission channel. This saves a significant amount of channel bandwidth.

The problem with Motion Estimation lies in the size of the problem. Even modest areas to search increase in computational complexity rapidly, which leads to large increases in power consumption. A variety of algorithms and techniques have been developed to find the optimal motion vector given the large throughput demands of the operation [2] [6] [15] [16] [17] [18] [19] [20] [21] [22] [23] [24] [25] [26]. Other

research has focused on computing relatively accurate motion vectors as efficiently as possible, in terms of the number of computations or circuit complexity. New algorithms that avoid searching large areas of the previous frame pixel-by-pixel accomplish these goals [8] [9] [10] [11] [12] [13] [14]. Some of these algorithms are applicable to low-power computing as they reduce significantly the number of computations needed and therefore the power consumption [10] [11] [12]. Other techniques have been proposed to reduce the size of arithmetic circuits required, eliminate unnecessary computation, and optimize the algorithm for the types of video being encoded [5] [10] [11] [12] [44] [45] [46] [47] [48] [49] [50] [51].

This thesis investigates development of a Motion Estimation block suitable for low bit-rate video encoding. The algorithm selection, model development, and power consumption results are explored as well as techniques for reducing the power consumption of the block. This research is built upon the wealth of work previously done in low-throughput motion estimation blocks.

The thesis is organized as follows. Chapter 2 describes video encoding motion estimation in detail. This includes motion estimation algorithms, implementations, and power-saving enhancements found in the literature. Chapter 3 describes the implementation of basic models. Chapter 4 presents the power-savings techniques utilized. Chapter 5 presents experimental results of the proposed system. The chapter includes a description of the design flow and the techniques to measure power consumption. Measured power consumption and resulting video quality are also presented in this chapter. Finally, Chapter 6 concludes the thesis and suggests a future research area.

Chapter 2
Low Bit-Rate Video Encoding and Motion Estimation

In this chapter, we review some basic concepts of the H.263 recommendation necessary to understand the terms and ideas presented in this thesis. We also review basic concepts of video compression, with an emphasis on Motion Estimation and Motion Compensation. Then, we present some popular Motion Estimation algorithms with some examples and analysis of their usage and applications. Previous research into implementations of these algorithms and some low-power techniques are included in the discussion. Finally, we present an analysis of the test videos targeted by this research is presented to justify low-power enhancements presented later.

2.1 H.263 Overview

The H.263 recommendation does not define the structure of a video encoding system, but only the format the compressed bitstream should follow. The specification also defines how a video frame should be partitioned for transmission, bitrate maximums, and other details (such as allowed quantization factors).

2.1.1 H.263 Basics

The H.263 recommendation allows certain sizes of video to be transmitted. For the purposes of this research, the QCIF size of 176 pixels x 144 pixels was selected for the video size. This size was determined to be an adequate physical size for videophones, mobile surveillance systems, and mobile patrol units to both fit physically in small devices and deliver adequate resolutions for the intended applications. The maximum bitrate allowed for QCIF is 64K bits per second [1].

The recommendation also specifies that the actual video information be converted from typical RGB (Red, Green, and Blue) color information to the YCbCr color space [1]. Y stands for the luminance portion of the color space, and Cb and Cr represent blue and red chrominance of the pixel, respectively. The following equations define the transformations from RGB to YCbCr [2].

$$\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.169 & -0.331 & 0.500 \\ 0.500 & -0.419 & -0.081 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \quad (1)$$

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 & 0.0 & 1.4021 \\ 1 & -0.3441 & -0.7142 \\ 1 & 1.7718 & 0.0 \end{bmatrix} \begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} \quad (2)$$

In the actual video representation used by H.263, the luminance samples are one-for-one based upon the RGB pixels. Therefore, for QCIF video at 176x144 pixels, there are 176x144 luminance samples. However, the red and blue chrominance values are subsampled by 2 in both the x and y directions. This means that there are 88x72 red and blue chrominance samples for a frame of video. This achieves some compression in the video by reducing the number of 8-bit samples required to store one frame of video from 176x144x3, or 76,032, 8-bit samples to 176x144+88x72x2, or 38,016 8-bit samples. Figure 2.1 illustrates the subsampled chrominance images and how they fit with the luminance samples, which lie "on" the pixels of the original RGB image.

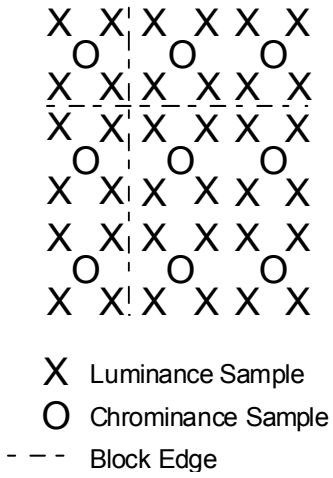

Figure 2.1 - Luminance and Chrominance Sampling [1]

As mentioned above, the H.263 recommendation also specifies how pictures should be divided for compatibility with the bitstream formats. The divisions are hierarchical for each frame. The top-level division is the Group of Blocks (GOBs). A GOB consists of k*16 lines, where k is dependent on the video size and is 1 for the QCIF size frame. The GOB is divided into macroblocks. Each macroblock is 16x16 pixels and

there are 11 macroblocks per GOB in a QCIF frame. Figure 2.2 illustrates this division for a QCIF sized video.

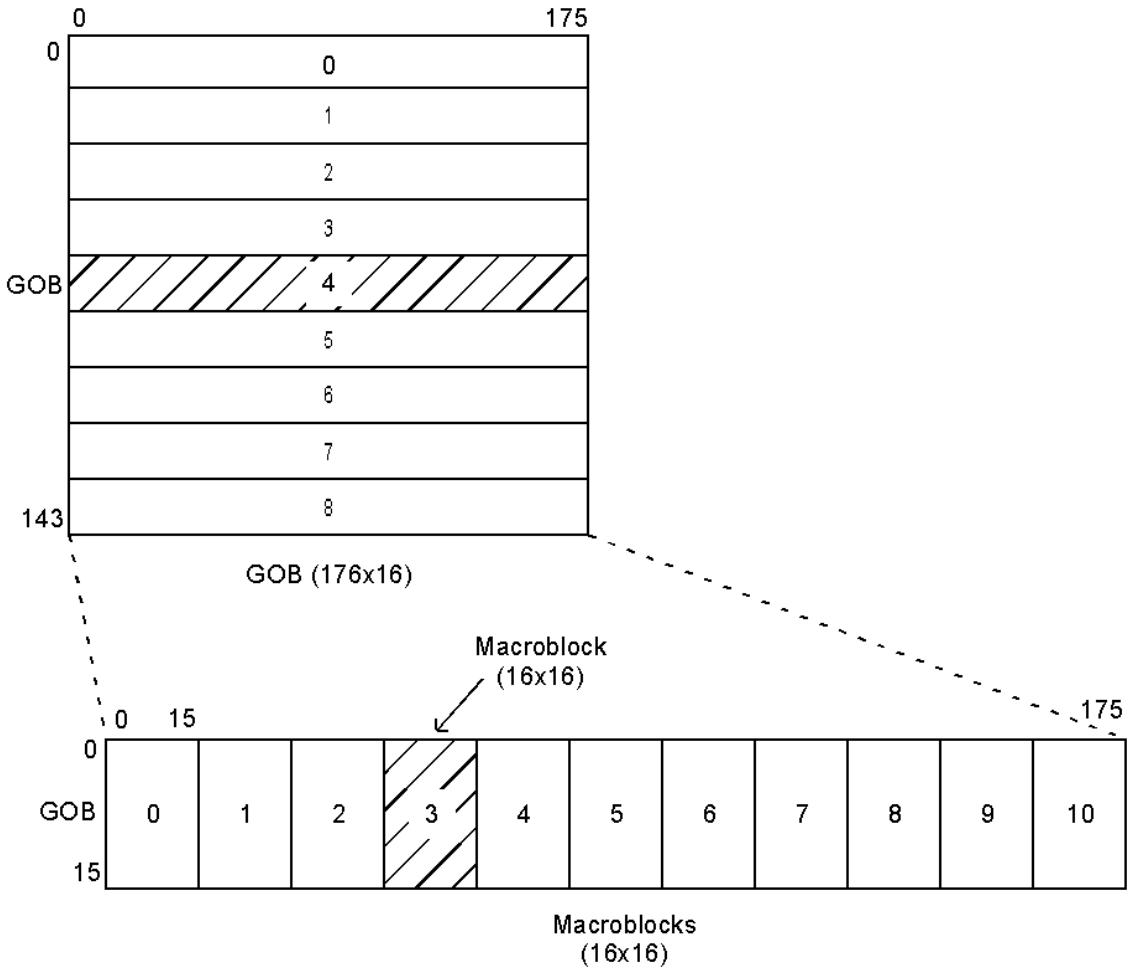

Figure 2.2 - H.263 Division of QCIF Video [1]

A macroblock consists of 16x16 pixels of luminance (Y) and 8x8 samples of blue and of red chrominance (Cb and Cr) corresponding to those luminance samples [1]. The dotted line in Figure 2.1 refers to a macroblock edge, which will not cross a chrominance sample. Blocks are 8x8 pixels of data. Thus, there are four blocks of luminance to a macroblock and 1 block of red and blue chrominance each to a macroblock. For the general case of video transmission without prediction and Motion Compensation, a macroblock of picture data is transmitted via 6 blocks in the order according to Figure 2.3. From these 6 blocks of data, a 16x16 pixel block of RGB data can be reconstructed at the decoder. Most compression of the actual video takes place at the block level, as will be seen later.

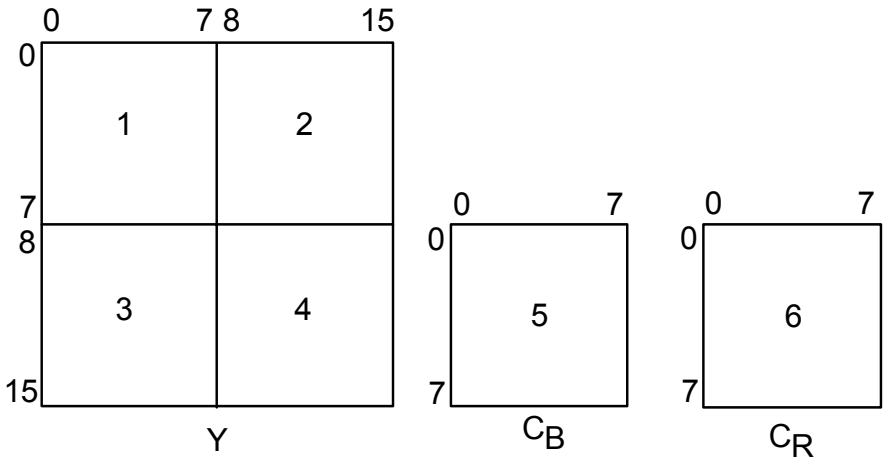

Figure 2.3 - Macroblock Transmission Order with Luminance and Chrominance Blocks [1]

2.1.2 H.263 Coder Overview

A typical low bitrate video encoding system employing the H.263 format takes the following form. This is a generalized block diagram taken directly from the H.263 recommendation (Figure 2.4).

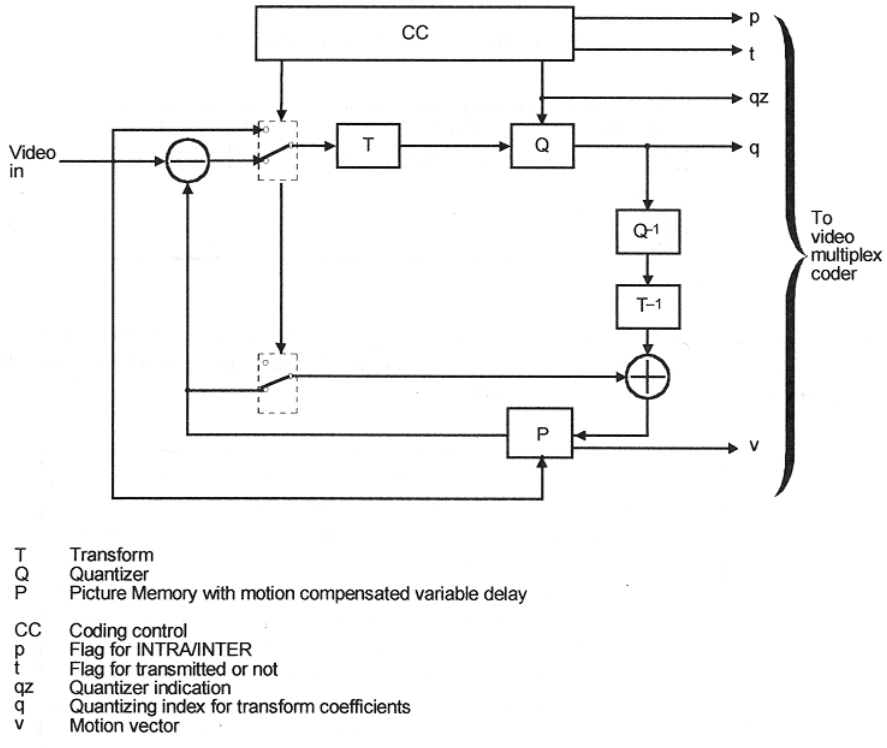

Figure 2.4 - Video Coder Overview from H.263 Recommendation [1]

From the figure, the most important elements of the video coder are the Transform block, the Quantizer block, and the Prediction block. Each block reduces the amount of information that must be transmitted across the channel. The Coding Control block controls the system to maintain the channel bitrate. The Quantizer is adjusted by the Coding Control to adjust picture quality according to the amount of information being sent across the channel.

The Transform block uses some type of transformation to typically take a block of pixel data and transform it to a set of coefficients in the frequency domain. With the aid of quantization, this reduces the number of coefficients necessary to transmit the necessary information. An inverse transform in the decoder restores the original pixel coefficients. Under H.263 8x8 blocks of pixels (corresponding to those found in Figure 2.3) are used as inputs to the Transform block since compression tends to be most efficient with this block size [1] [2]. The Discrete Cosine Transform (DCT) is often used as the transform function since it has an efficient implementation in terms of hardware and computational complexity [2].

The Quantizer block, as mentioned above, is responsible for quantizing the DCT coefficients. This is simply a division of the DCT coefficients by a pre-selected factor. During division some of the information is lost (e.g. division by 8 would effectively truncate 3 of the least-significant bits (LSB) of a coefficient). The quantization factor and the DCT coefficient are transmitted in the bitstream to the decoding block, which multiplies the quantized factor to the transmitted coefficient, a process referred to as *inverse quantization*, to get the original coefficient, albeit with some of the LSB missing. An inverse transform is applied to retrieve the original block. The coding control of the video encoder can increase the quantization factor to effectively decrease the number of bits a block of DCT coefficients required to be transmitted. In this way, the bitrate of compressed video transmitted over the transmission channel can be controlled. The consequence, of course, is video degradation at the video decoder.

The previous two blocks, the Transform and Quantizer, serve to reduce the spatial redundancy present in the picture information. That is, they compress the information present in a single block of data for one frame. Since a video consists of a sequence of pictures, it also makes sense to exploit the temporal redundancy often present in video,

especially considering that a particular frame is often not entirely different from the previous one. Many parts, or blocks of a frame, are identical or closely related to a corresponding block from the previous frame. Using temporal redundancy between blocks of different frames to compress required video information is referred to as Prediction.

Prediction consists of two processes - motion estimation and motion compensation [2]. Motion estimation attempts to find the best match of a block of pixels with the previous frame. Motion estimation's goal is to find a vector that points to a block of pixels in the previous frame that correspond to the current block of pixels in question. In the case of H.263, the blocks are macroblock sized (16x16). Only luminance values (256 values per macroblock) are usually checked during motion estimation. Motion compensation then uses this motion vector to create a motion compensated version of the current block of pixels. This new macroblock is basically the difference of the two "matching" macroblocks and is defined by the following equation: [2]

$$e(x, y, t) = I(x, y, t) - I(x - u, y - u, t - 1) \quad (3)$$

In equation 3, (u, v) respectively represent the x and y coordinates of the motion vector found during motion estimation. The third dimension is time. Hence I(x, y, t) represents the intensity at position x and y at time t. Thus, the previous frame block is being subtracted from the current block of pixels. The output of the motion compensation block can then be Transformed using the DCT and Quantized and transmitted across the channel along with the motion vector. The decoder can inverse this operation by performing its own motion compensation to retrieve the current block of pixels, albeit degraded by the quantization factor and any losses from the DCT inverse transform [2]. (Refer to Figure 2.5 for a picture of the encoder and decoders). Compression is achieved if the blocks "match up" well, so maximum compression depends on the success of motion estimation. The question of motion estimation algorithms, techniques and implementations will be discussed in the next section and the rest of this paper.

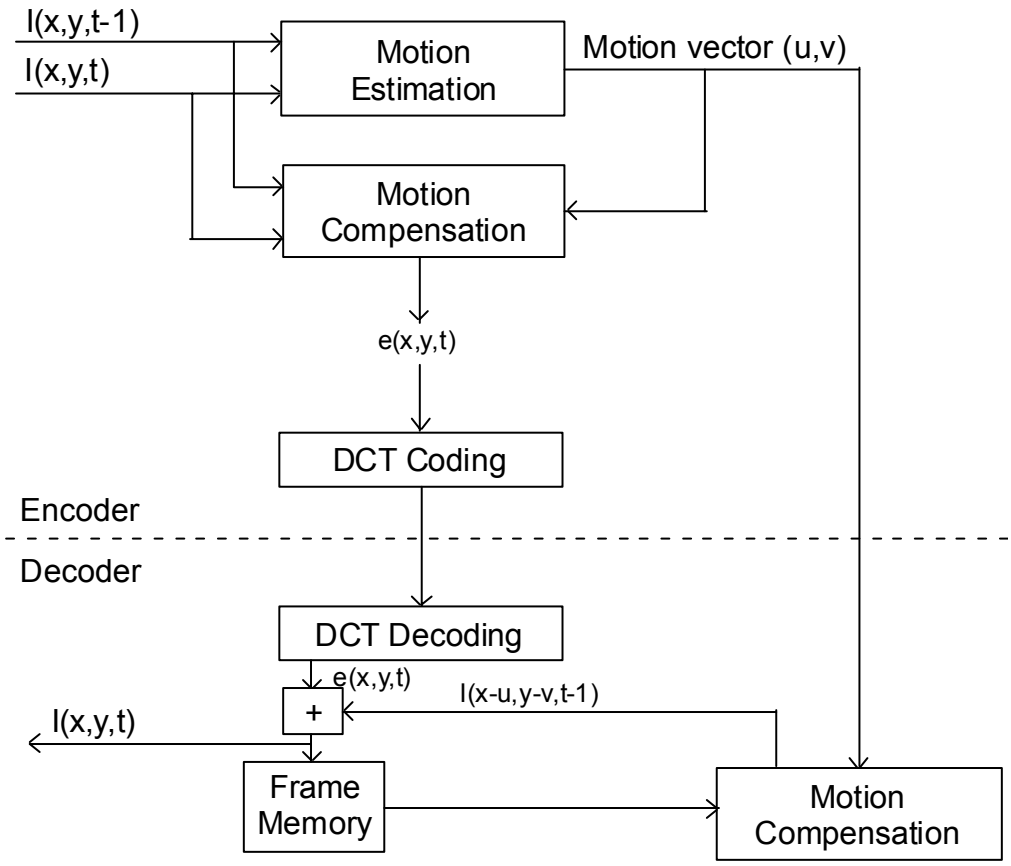

Figure 2.5 - Generic Video Encoder and Decoder Supporting Motion Estimation/Compensation [2]

H.263 requires the use of both INTER and INTRA mode encoding for macroblocks. INTRA refers to the transform and quantize method of compression, where only spatial redundancy is compressed. The raw block of pixels is fed to the DCT engines and the coefficients quantized and sent over the channel. INTER mode is when the macroblock is placed through the motion estimation and motion compensation engines. In most cases, for all video except those blocks in the first frame, motion estimation is performed on the macroblock and the rating of the quality of the motion vector is used to determine the INTER/INTRA decision. Adjacent macroblocks in a frame can be either INTRA or INTER encoded depending upon the success of the motion estimation operation.

From the above discussions, it becomes fairly obvious that compression can be traded off with video quality on a number of fronts. These methods include:

- Threshold for selecting INTRA or INTER for a block. Efficiency of the motion estimation algorithm to find a quality match for the macroblock in question.
- Quantization factor used in reducing the size of transformed coefficients.

These are true for a number of reasons. First, not every motion estimation operation is perfect (nor is expected to be perfect) in finding a block that completely matches the original, thus there are some errors in the pixel values for an INTER motion compensated block. Secondly, since only luminance values are used for motion estimation, further errors can be seen after translation to the RGB color space. INTRA encoded blocks do not suffer these problems-they are always an "exact" translation of the original block minus quantization and transform effects. The tradeoff comes with usually more bits being translated for an INTRA block. For example, if an INTER encoded block does find a perfect match, then no pixels (or "zero" DCT coefficients) need be transmitted in the bit stream, only the motion vector is of consequence.

An important consideration is the quality of video produced by a codec. Since this is, in essence, a subjective question dependent upon the viewer, a quantitative means for measuring video quality is highly desirable. The Peak-Signal-to-Noise ratio (PSNR) was introduced as a measure of video signal quality for a frame of video. The equation for the PSNR is given as [2]:

$$PSNR = 10\log_{10}\frac{255^2}{\frac{1}{rows \cdot cols}\sum_{j=1}^{rows}\sum_{i=1}^{cols}(Y - Yreconstructed)^2} \quad (4)$$

where Y is the luminance pixels of a frame of video (144x176 for the QCIF case) and $Y_{reconstructed}$ represents the pixels of the reconstructed frame after motion estimation and motion compensation (i.e. what the decoder will "see"). Also note that the reconstructed frame is stored at the encoder and is used as the basis for the next frame's motion estimation procedure. While the PSNR is not useful as a measure of good video in and of itself, it is most useful as a comparison between two related videos to measure relative video quality improvement or degradation [2]. For the purposes of this research, it will be used in this fashion to measure the effects of different motion estimation algorithms and implementations on test videos.

2.2 Motion Estimation Algorithms

With the importance of a good motion estimation algorithm established to enable motion compensated prediction to work well, the problem turns into one of finding a good algorithm for discovering good matches between macroblocks from different frames. This section will explore the fundamentals of motion estimation algorithms, including defining the problem of motion estimation, algorithm classifications and ways to "rate" motion vectors.

Ideally, a video encoder would take full advantage of the compression offered by motion compensation by scouring the entire previous frame for a match to find the best possible match for the macroblock in question. However, that is not feasible. Refer to Figure 2.6 for an illustration. Consider a macroblock size of 16x16 on a QCIF-sized video frame. Also assume that any candidate macroblock must fit entirely within the previous frame (i.e. not hang over the edge) and that a motion vector is the relative coordinates between the upper-left-hand corner of the reference block and the candidate block. Then, the upper left-hand pixel of a candidate macroblock must fit within the area denoted in gray in the figure. A suitable candidate macroblock would then be any macroblock that fits the area shown in the figure with its upper left-hand corner pixel within the gray area. This gives a possibility of 161x129, or **20,769** motion vectors to be checked per macroblock. With 99 macroblocks per frame, there are over **2 million** motion vectors to be checked and evaluated per a single frame of video.

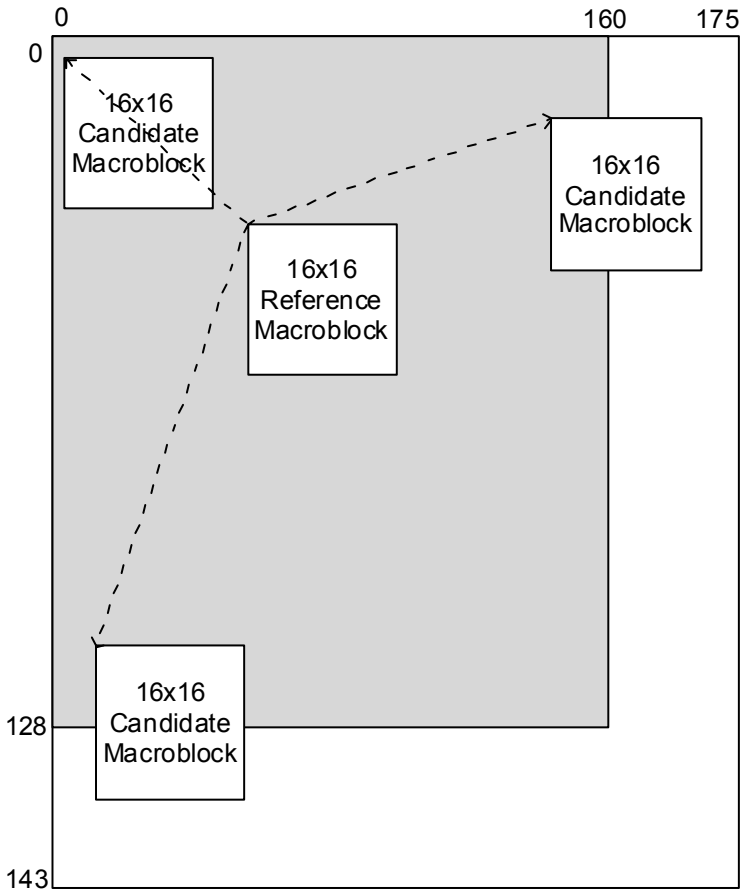

Figure 2.6 - Illustration of Motion Estimation Computational Complexity for Entire QCIF Frame

Usually then, the area of motion estimation is restricted to what is called a search area. This greatly limits the number of possible candidate blocks from the impractical statistics mentioned previously. For example, in H.263 the search area is limited to [-16, +15] under normal operation [1]. Figure 2.7 illustrates this case. Again, the upper-left hand corner of the reference block can be "shifted" anywhere inside the gray area. The entire search area is 47x47 pixels to accommodate all the motion vectors. Now there are only 31x31, or **961** motion vectors to be checked per macroblock and **95,139** per frame.

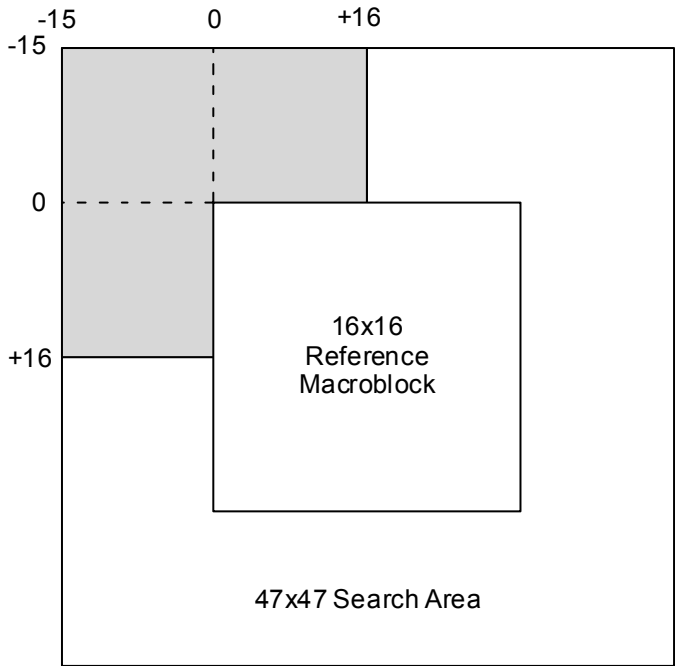

Figure 2.7 - Illustration of Motion Estimation Under H.263

The result of any motion estimation operation should include not only a motion vector, but also a way to enumerate the "fitness" or quality of that motion vector. Note that this is extremely important in video encoders that support INTER/INTRA encoding such as H.263. The fitness rating will be used by the control block of the encoder to determine whether to use the motion compensated INTER-coded block or the INTRA-coded block.

From the literature, the sum-of-absolute difference rating (SAD), the mean-squared error (MSE), and the cross-correlation function (CCF) ratings are widely used [3] [4] [5]. The SAD, MSE, and CCF ratings for a given motion vector are given by: (assuming 16x16 blocks) [3]

$$SAD(i,j) = \sum_{m=0}^{15}\sum_{n=0}^{15}\left|y(n,m) - x(n+i,m+j)\right| \tag{4}$$

$$MSE(i,j) = \tfrac{1}{16\cdot 16}\sum_{m=0}^{15}\sum_{n=0}^{15}\left(y(n,m) - x(n+i,m+j)\right)^2 \tag{5}$$

$$CCF(i,j) = \frac{\sum_{m=0}^{15}\sum_{n=0}^{15} y(n,m)x(n+i,m+j)}{\left[\sum_{m=0}^{15}\sum_{n=0}^{15} y(n,m)^2\right]^{\frac{1}{2}}\left[\sum_{m=0}^{15}\sum_{n=0}^{15} x(n+i,m+j)^2\right]^{\frac{1}{2}}} \tag{6}$$

The best matching criterion is found by using the CCF, at the expense of greater computational complexity [4]. The SAD and MSE give similar results for quality of motion vectors [3]. However, the SAD is the easiest to compute of the three, especially in hardware. Hence, SAD is used in the proceeding discussions of the various motion estimation algorithms and in the models developed as part of this research. Therefore, for the selected macroblock size of 16x16 pixels, processing one SAD for a motion vector requires 256 8-bit subtractions and 255 accumulate operations (additions). For all motion estimation algorithms discussed hereafter, a motion vector is considered better if its associated SAD rating is lower than another motion vector's SAD rating.

The next sections detail some of the widely used motion estimation algorithms. The basics of each algorithm are discussed, including computational complexity, accuracy, and any advantages and disadvantages to implementing such an algorithm in hardware.

2.2.1 Full-Search Block-Matching Algorithm

The Full-Search Block-Matching (FSBM) algorithm is the most straightforward motion estimation operation. Simply speaking, it calls for searching the entire allowable search area for the best motion vector-the one with the best SAD rating. Obviously, this algorithm always finds the optimal motion vector for the given search area. It should also be obvious this algorithm suffers from the largest computational complexity. For example, considering the search area of [-7, +7], there are 225 SAD calculations that must be performed for each macroblock in a frame. Again with 99 macroblocks in a QCIF-sized frame, that translates to **22,275** SAD calculations per frame of video to perform motion estimation.

The advantages of implementing FSBM as the motion estimation algorithm include both the guaranteed optimality of the solution and the regularity of a hardware implementation. This regularity stems from the fact that consecutive motion vectors share many of the pixel values in calculation. This lends itself to using large array processors to efficiently implement a FSBM motion estimation processor with relatively simple control logic [6]. Specific implementations of FSBM motion estimation processors are explored in more detail in the next section regarding previous research.

Disadvantages of the FSBM algorithm lie in the large amounts of area required to implement the systolic arrays, the high throughput requirements to calculate all the SADs and the large amount of power required to drive the systolic arrays [7].

2.2.2 Three-Step Search

The Three-Step Search (TSS) algorithm as well as the next two algorithms (the New Three-Step Search and the Four-Step Search) belong to the family of algorithms known as the Two-Dimensional Logarithmic Search, where the search hones in on the optimum motion vector by dividing the search space into portions and sampling motion vectors at key points [2]. These algorithms "work" on the basis of the Uniform Error Surface Assumption (UESA). This concept states that the error surface created by taking the SAD ratings at every possible motion vector point should reduce monotonically the closer the motion vector is to the true minimum [3] [8] [9].

The TSS algorithm is defined by the following three steps [8]:

1. Search in a box pattern (9 points) located at the motion vectors: (-4, -4), (0, -4), (+4, -4), (-4, 0), (0, 0), (+4, 0), (-4, +4), (0, +4), and (+4, +4). This search pattern will be referred to as a 9x9 search box. Find the minimum SAD rating.
2. Take the previous "winning SAD" as the new center point. Then search in a similar (box-like) fashion, but at a distance of 2. This is a 5x5 search box.
3. Move the "center" to the next SAD winner, but this time search again using a box-distance of 1, or a 3x3 search box. The winning SAD from this step is the winner.

The TSS is also illustrated by the following figure. Two separate operations are illustrated in the figure. For both searches the starting point is (0, 0) with the box pattern of nine points. The first search box (9x9) has been highlighted with shading. Suppose that the best vector found in the first step is (+4, +4). The next step of the algorithm proceeds with a smaller search box center at that new position. (+4, -6) wins that step. The final step searches all immediate points around (+4, -6) with the lowest SAD found, (+3, -7) being selected as the optimum motion vector. The left-bottom search illustrates the situation when the winning SAD in the first step is (-4, +4) and proceeding in the same fashion as described above. In both cases, notice how the size of the search box is decreasing with each step of the algorithm.

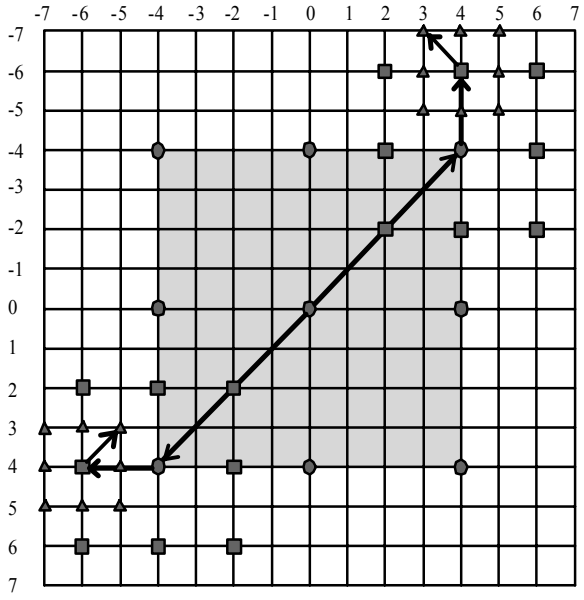

Figure 2.8 - TSS Example Searches

The algorithm has some very desirable characteristics. For the [-7, +7] search area considered, the number of SAD calculations is **27** for each motion estimation operation. Since the middle point of steps 2 and 3 are computed in previous steps, there are really only 25 unique SAD calculations per motion estimation operation. This translates to **2,475** SAD calculations per frame (QCIF), a much more favorable computational complexity than with the FSBM case.

Some consequences exist with using the simplified TSS algorithm. The hardware implementation becomes less straightforward and includes some considerations that preclude usage of a simply array processor [10]. While the steps of the algorithm are regular in relation to each other, the overall control logic is decidedly more complex than with a the FSBM implementations. As before, implementations of the TSS in the literature will be presented later.

Other problems with the algorithm include the lack of accuracy that occurs with not checking every possible point. A variety of problems can cause the UESA to not apply to a given set of pixels [9]:

- The aperture problem
- Local block image changes
- Luminance change between frames

- Inconsistent movement between foreground and background in an image sequence

Finally, another problem is seen with video that has either no motion or small amounts of motion [9] [11] The TSS algorithm is inflexible and requires all 3 steps of the algorithm and all 25 SAD calculations regardless of where the optimum motion vector may lie. This process is inefficient when many of the macroblocks move a single pixel or two or remain stationary. The next two algorithms discussed attempt to resolve that problem while maintaining many of the advantages (and the form) of the TSS algorithm.

2.2.3 Three-Step Search Hybrid Algorithms

The New Three-Step Search (NTSS) algorithm was proposed in 1994 by Li, et. al [9]. The algorithm attempts to bias the first step of the algorithm to search around the zero vector, to detect small motion more efficiently than with the TSS. The algorithm proceeds as follows: [9]

1. Search the 9 points as usual in the TSS algorithm. However, also include the 8 points directly surrounding the zero (center) motion vector. If the winning vector is the zero vector, then stop. If the winning vector is one of the 8 points surrounding the zero vector, then search the 8 points directly surrounding **that** vector and stop.
2. Otherwise proceed with steps 2 and 3 of the TSS algorithm as usual.

The algorithm is illustrated in the following Figure 2.9 for the vector search ending (+2, -1). In the first step all points highlighted with an oval are searched. Suppose the lowest vector is found at (+1, -1) in the first step. Then all points surrounding that one are searched in the special final step of the algorithm. The lowest SAD found from those points is taken as the optimum, which is (+2, -1) in this case.

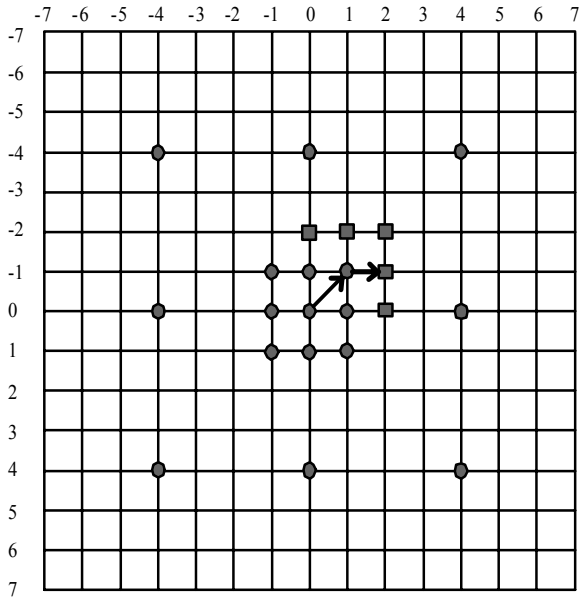

Figure 2.9 - NTSS Example Search

The first step thus has 17 vectors to check. If the zero vector wins, this is the amount of SADs that must be checked. This is obviously the best case for this algorithm. Should the algorithm have to proceed to steps 2 and 3, then 33 unique values must be checked, worse than the original TSS algorithm. Should the alternative case be selected, where one of the 8 values surrounding the zero vector wins step 1, the number of SAD calculations is either 20 or 23, both slightly better than TSS. The hope with using the NTSS is that enough zero vectors and vectors close to the zero vector can be found to result in an overall computation savings. For video with large motion, the NTSS is obviously not a good choice, since more SAD calculations are needed.

The NTSS also further sacrifices some of the regularity in a hardware implementation. This shows up in both the implementation of the SAD calculation hardware and the control logic. There are separate decision steps in the algorithm that must be accounted for, making the control logic more complex than the deterministic TSS implementation.

Another hybrid TSS method is the Enhanced Three-Step Search algorithm [12]. This algorithm only changes the first step of the algorithm so that a smaller search box is used to concentrate the first step towards the center of the search area. It promises to achieve better results than the TSS method by using the same number of search points but with faster convergence to vectors near the center of the search area.

2.2.4 Four-Step Search

The Four-Step Search (4SS) is another algorithm that builds upon the TSS, while being more center-biased. The 4SS differs in that it maintains a more regular search pattern with half-stop techniques employed in the algorithm. The algorithm proceeds as follows [11]:

1. The minimum SAD found using a 5x5 search box. Nine points are checked at (-2, -2), (0, -2), (+2, -2), (-2, 0), (0, 0), (+2, 0), (-2, +2), (0, +2), and (+2, +2). If the minimum is at the zero vector, then proceed to step 4, else go to step 2.
2. The 5x5 search box is maintained, centered at the winner of the previous step. Notice that using the same sized windows creates a lot of overlap, or redundant calculations that were searched in the previous step. If the winning vector is the center of the search box, proceed to step 4. Otherwise, proceed on to Step 3.
3. Using the winner of Step 2, search exactly as mentioned in Step 2, but proceed to Step 4, regardless of winner.
4. Reduce the size of the search box to 3x3. Then search around the previous step's winner for the final motion vector.

To better illustrate the algorithm, the following figure gives an example as to how the search would proceed for the final motion vectors (+3, -7) and (-5, +3). Shading highlights the original 5x5 search box.

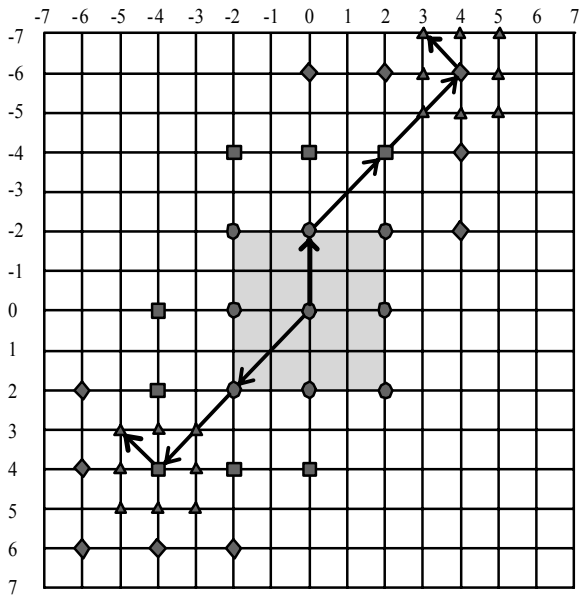

Figure 2.10 - 4SS Example Searches

The 4SS offers a similar best-case scenario for computational complexity when compared to the NTSS (17 searches). The worst-case situation for the 4SS is 27 search points compared to 33 for the NTSS, which is an improvement. The average computational complexity for some real-world images is reported to be better for 4SS than NTSS [11].

The usage of the regular-sized search boxes enables a more regular hardware implementation, as will be seen in the later sections. The hardware complexity is added with enabling the motion estimator to detect the step 4 condition and stop the algorithm when necessary. The flexibility of the algorithm enables it to use fewer searches for small motion, which can translate to savings in the number of computations for small motion or relatively still video. The worst-case number of calculations is a bit worse for the 4SS than the TSS, which is 27 versus 25.

2.2.5 Other Algorithms

Some other algorithms have been proposed for improved performance over those mentioned before. However, they lose regularity and generally are more applicable for a software coder implementation. They include the Center-Biased Diamond Search Algorithm and the Center-Biased Hybrid Search [13] [14].

The Center-Biased Diamond Search uses a diamond-shaped search area to search for the optimum motion vector. It begins with a small diamond around the center and "walks" the diamond pattern outward if the lowest motion vector is found not to be in the center of the diamond. It has a best case of 13 search points and an average of 15.5 search points [13].

The Center-Biased Hybrid Search uses a combination of "X" patterns and diamond-shaped patterns to more efficiently search for motion vectors. It promises a 10.2 average number of search points with an improved best case of 9 search points [14]. However, the increased lack of regularity makes this algorithm an even less suitable choice for a hardware implementation.

2.3 Previous Research on Implementation

This section reviews previous work on motion estimation implementations. The information is presented as a basis for the research work presented later and as an

overview of the issues facing motion estimation block development. The research is divided into sections based upon the algorithms being reviewed.

2.3.1 Full Search Block-Matching (FSBM) Research

The FSBM algorithm calls for searching all possible motion vectors, easing the need to develop complex control logic. However, the problem arises to meet stringent throughput requirements for the motion estimation block, often necessitating the development of special purpose hardware [6]. In conjunction with the large arithmetic calculation requirements, often large memory bandwidths are needed with FSBM blocks [15].

To handle these large computational loads, some sort of a systolic array processor is often used [2] [6] [15] [16] [17] [18] [19] [20] [21] [22] [23] [24] [25] [26]. The systolic array consists of a set of processing elements (PE) connected locally to share data. The advantages include low control overhead, high clock rates, and high processor efficiency (meaning that the PEs can be kept busy for the majority of the time) [6] [25]. The arrays can be either one-dimensional or two-dimensional. An example of a 1-D systolic array is included in Figure 2.11.

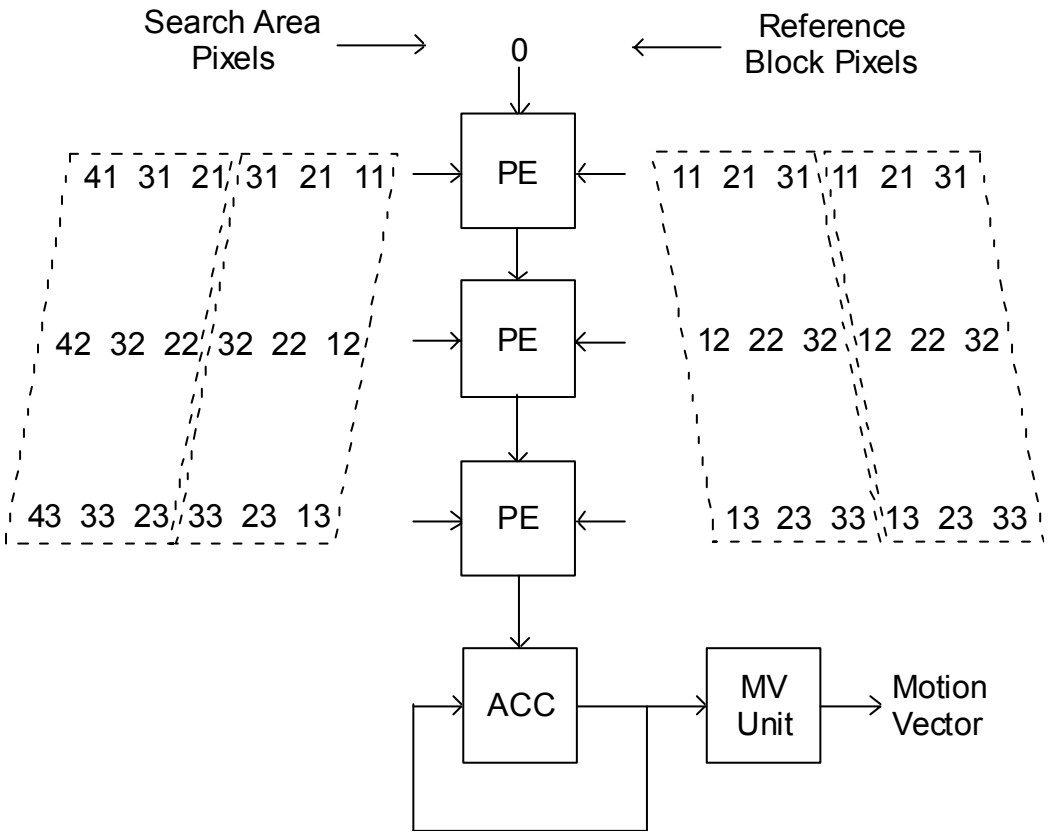

Figure 2.11 - Example 1-D Systolic Array [6]

The tradeoff with increased parallelism and higher throughput occurs with larger memory bandwidth requirements, often requiring a multiple-ported memory to support the PE array operation [6] [18] [25] [26]. In response to this, some architectures utilize large shift registers to store pixel values for PEs farther in the chain [20] [21] [24]. 2-D systolic arrays can further share data and allow even more parallelism by calculating multiple SADs per unit of time [6] [16] [17] [18] [21] [25]. Figure 2.12 shows a 2-D architecture that conserves current frame block memory accesses by using a 16x16 PE array [25]. The current reference block pixels are stored local to each PE; hence no local memory is required for these pixels. This further illustrates the severe trade-offs, in this case between circuit area and memory, necessary with the FSBM implementations.

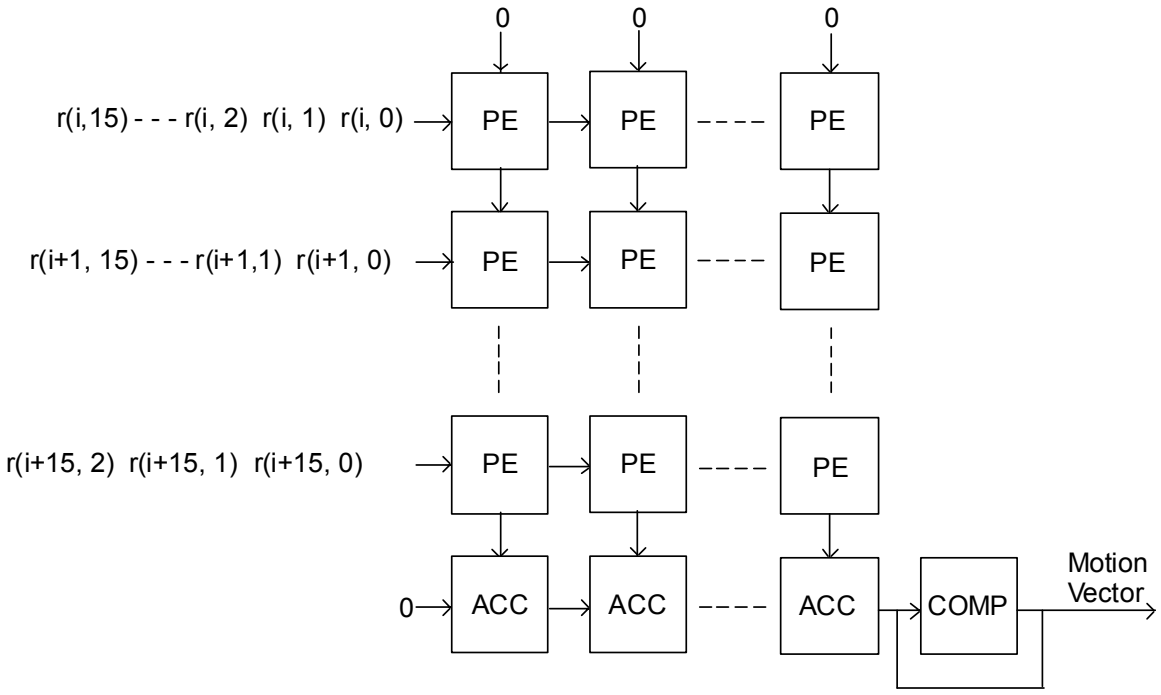

Figure 2.12 - Example 2-D Systolic Array [25]

2.3.2 Three-Step Search Research

Many TSS models have been reported in the literature [10] [27] [28] [29] [30] [31] [32] [33] [34] [35]. As opposed to an overall throughput problem, most TSS models attempt to overcome both the irregularity introduced by the TSS algorithm (when compared to the FSBM) and the less-than-completely-deterministic nature of the algorithm. The second and third steps of the algorithms are dependent upon the previous step.

Some architectures use a 1-D systolic array [10] [29] [30] to achieve as much parallelism as possible. One implementation [10] suffers latency problems due to the step-dependence problem mentioned above. Also, a 3-port memory is required for getting the current and previous search buffers to the correct PE array. The data is passed by x-axis from PE to PE to maintain some benefits of the parallelism of the systolic array after the latency of initialization is met. Two alternative methods for dealing with the latency were presented in [29] and [30]. He and Liou proposed checking 3 points for every single point in the TSS algorithm. These are referred to as checking vectors [29]. Their approach is to use the parallelism of a 1-D array to check extra points to enhance accuracy. The latency penalty thus is traded off with the increased accuracy at little cost

in computations [29]. The other method attacks the latency problem more directly [30]. The SAD accumulation procedures and the comparison structures are implemented in a tree-fashion to reduce the overall latency by a factor of $\log_2$ at the expense of more area. Memory interleaving is used to reduce the overall bandwidth into a single memory module. The proposed scheme uses a single bank to hold each of the 16x16 values, thus there are 256 memory modules for a 16x16 block. This means that each pixel of a SAD calculation can be accessed from a different module, supporting the throughput of the tree of subtractors and accumulators. Pipeline interleaving is used to reduce the number of idle cycles in the calculations [30]

Some parallel architectures were proposed for more efficient implementation of the TSS algorithm [31] [32] [33] [34]. Costa, et. al. introduced a family of parallel architectures [31] that can use 3, 9 or 27 PEs. Each set of PEs work in parallel on a row of a search box. By increasing memory bandwidth, the larger PE sets can work on an entire search box (9 PEs) or a larger search area (27 PEs) at once. Programmable delay units are used to facilitate the parallelism [31]. Additionally, the architecture is modular enough to combine PE structures and control units to create such things as large search area motion estimators and forward-backward predictors [31]. Two of the other proposed implementations [32] [33] use a 9-PE implementation to complete calculations for a search box in parallel. The PEs use data from 9 different memory modules. The data itself is loaded into these modules based upon a residual memory addressing system based upon 2^k (where k=2,1,0 for the different steps of the algorithm), so the PEs should always be addressing one of the 9 memory modules exclusively at any given time [32] [34].

Another architecture was developed using a mesh structure to solve the parallelism problem without incurring high latency [35]. In this implementation, 9 PEs are connected in a mesh fashion, with data connections to all 4 neighbors, similar to a highly parallel general computing network. Instead of each PE computing all 256 calculations for a single SAD value, the PE calculates 16 values and shifts the partial result to a neighbor. Through the proper memory interleaving, each PE can be connected to its own memory module and the results of the 9 required SAD values are computed after 256 clocks. The comparison step is also achieved using “intelligent” shifting of the

data with on-board comparators, such that the final result is achieved after 4 clocks. This reduces the latency problem with each step of the TSS [35].

The NTSS algorithm incorporates the 8 checking points around the zero vector in the first step. He, et. al. proposed an architecture using 3 1-D arrays (48 total PEs) to compute the difference values [36]. Programmable delay units are used to save re-broadcasting of reference block pixels. These also allow pixel values to be shared among the PEs [36]. However, this architecture requires extremely high memory bandwidth to feed the 48 concurrently running PEs with search area data.

In addition to the models and implementations mentioned earlier some algorithmic enhancements have been presented as well [37] [38] [39] [40] [41]. Jong, et. al, suggest using multiple winners in each step of the TSS algorithm (the two lowest SADs as an example) to improve the accuracy of real-world video motion estimation in case the UESA assumption does not completely hold for an image [37]. Also, the technique of pixel subsampling is mentioned as an alternative to reducing input memory bandwidth and computational load. However, the authors have determined that subsampling only the previous frame search area (candidate block) is necessary and maintains good motion vector accuracy [37]. Xu, Po, and Cheung have proposed tracking the motion vectors found in previous macroblocks and taking the ones most used as starting points for subsequent applications of any block-matching algorithm, since motion is consistent over time in most real-world images [38].

Taking the UESA assumption a bit further, searching both directions in the first step of the TSS algorithm can be seen as wasteful, since only one direction will guide the algorithm to the optimal vector. Hence, some algorithms add a step to check only orthogonal directions from the zero-vector to better steer the TSS algorithm in the correct direction based upon this preliminary first step [39] [40] [41]. The algorithms presented by these works, called the simple and efficient search (SES) [39] and the fast three-step search (FTSS) [40] reduce the number of checking points, on average, by half based upon software simulation [39] [40].

2.3.3 Four-Step Search Research

With both the recent advent of the 4SS algorithm and its similarities to the TSS algorithm in structure, relatively few implementations of a 4SS motion estimator were

proposed in the open literature [42] [43]. The architecture introduced by Wu exhibits some low-power characteristics and a representative implementation of the 4SS algorithm [42]. It is presented in the following section, which describes low-power designs.

2.3.4 Low-Power Designs for Motion Estimation

A few attempts have been made at reducing the power consumption of FSBM implementations [44] [45] [46] [47]. As could be implied from the previous discussions on the large throughput requirements of the FSBM block, the motion estimator is responsible for nearly 60% of the power dissipated in certain FSBM video encoders [44]. Thus, any reduction in power in the motion estimator would be critical for inclusion in a portable or other power-conscious device.

Moshnyaga introduced the idea of a SAD criterion for shutting down the PEs after a certain point in the calculation of a motion vector [44]. The proposed block alters the FSBM algorithm by monitoring picture variation and increasing the threshold of other blocks when the picture is changing rapidly (i.e. the SAD values are increasing and the motion vectors are non-zero with an increasing frequency.) The basis for this research is that most vectors are near the zero-vector, and motion is gradual over time. This allows the threshold to be ramped up over time without much loss in motion vector accuracy. The shutting down of PEs occurs by using gated clocks that are shut off when the given threshold is breached. Only ¼ of the calculations of the original FSBM algorithm are necessary with this approach [44].

Do and Sousa presented an alternative strategy to shutting off the PEs [45] [46] [47]. A conservative estimate is calculated for each row of the macroblock. This estimate is simply the sum of all the previous search area pixels minus the sum of all the current search area pixels for the given row [45]. If this estimate is greater than the current SAD rating found so far, then the calculation is skipped for that vector. Hardware structures are also presented to accomplish the estimation and the comparison and stop operations. A blocking latch is used to “disable” PEs when determined to be necessary by the power-reduction algorithm. As long as the power to compute the estimate is conserved by the power saved by removing normal SAD calculations, an overall power gain is achieved. A savings of 50% of calculations is reported [44]. Alternative

implementations to this algorithm based upon the same concepts of a conservative SAD estimate calculation and comparison have been presented in [45] and [46].

Wu introduced an efficient VLSI implementation of the 4SS [42]. A block diagram of Wu's motion estimator for a block size of 3x3 is given in Figure 2.13. The PE structure proposed is a 3xN 2-D systolic array, where N is the height of the block size being considered. Each column calculates a SAD for one column of the search box. Therefore, 3 SADs are being calculated at any given time. (This is one row of the current search box.) Notice that for each PE in a row, the local memory is delivering the correct row of data. Programmable delay units (PDUs) delay the sharing of current reference block pixel data between adjacent PE columns by either 1 or 2 clocks. This implements either a 5x5 search box (Steps 1, 2, and 3) or a 3x3 search box (Step 4), respectively. Power consumption is reduced in Wu's model by recognizing the overlapping calculations of SADs in the second and third step and disabling PDUs and PEs when necessary. 40% of power was saved with this enhancement [42].

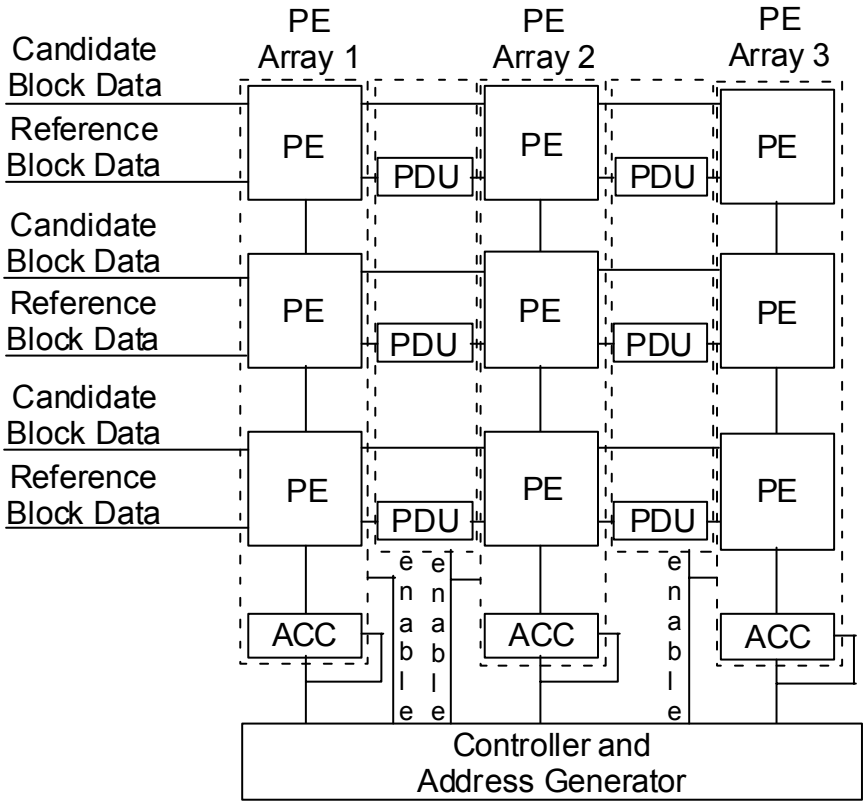

Figure 2.13 - 4SS Motion Estimation Implementation [42]

Two papers present ideas of using less than the full 8-bits of the pixels for motion estimation [48] [49]. Moshynaga proposed truncating the MSBs dynamically [48]. Basically, the two input pixel values are looked examined from the MSB to the LSB. As long as the MSB of the two values are equal, those MSBs are thrown out. The idea is to save the power toggling inherent in converting to 2's complement and sign extending the subtracted value after a result before sending it to the absolute value logic. The paper shows that 2-3 bits are usually truncated on average for most test videos without compromising PSNR. More importantly, the method reduces the switching activities of the processing elements by 50%, which saves dynamic power dissipation by the same amount. He and Liou proposed cutting off the LSBs of the pixel values and performing SAD calculations on a predetermined number of the MSBs only [49]. Simulation results show that over 50% of the gate count in the processing elements is reduced with a 4-bit reduction in pixel values, while the PSNR is reduced only by 2%.

Two other techniques attempt to reduce the motion estimation procedure to that of a binary matching procedure (i.e. 1-bit matching via filtering) [50] [51]. Mizuki, et. al used a binary edge detector to reduce the 8-bit value image to a 1-bit edge detected image, where edge colors are represented with a 1, and "non-edges" with a 0. Thus, the PE is simplified greatly to a simple XOR and increment operation [50]. Obviously, this PE is much simpler than one with an 8-bit subtractor, 16-bit adder, and a large register used in SAD-calculating implementations. As long as the filtering operation for converting the pixels down to a single-bit consumes less power than the SAD-calculating PE would, a large amount of power can be saved. Natarajan, et. al. used a special filter to compute the 1-bit value field that is matched for the motion vector computation. The filter is a bandpass image that represents the mid-frequency content of the images. Again the PE is greatly simplified to an XOR with an increment operation [51]. The primary advantage of both architectures is that the hardware area associated with the processing elements, and hence the power dissipation, is reduced greatly. Yeo and Hu proposed a mesh-based systolic array for FSBM and TSS based binary block-matching architectures [5].

2.4 Analysis of Targeted Videos

The intended application for our low bit rate motion video system is for portable usage with very low bandwidth available for video transmission. As mentioned in the Introduction, these applications can include cellular phones, video surveillance, and mobile patrols. Often, this type of video consists of "head and shoulders" shots of a person talking, a still picture, or other "low motion" video. This section examines some test videos for information related to SAD ratings and motion vector distribution. The information will be used later to justify some algorithmic alterations in the basic algorithms that directly lead to low-power enhancements.

Three sample video clips were selected to represent real-world video clips. These videos are summarized in Table 2.1 by their content and relative amount of motion.

Table 2.1 - Summary of Test Videos

Video	Description	Motion
Suzie	Woman talking on the telephone	Still head and shoulders (Low motion)
Carphone	Man in a car talking on a videophone with a moving background.	Animated facial motions and background motion (medium Motion)
Foreman	Head and shoulders of a man talking with a large camera pan of a construction site.	Head and shoulders with a large scene change during camera pan (High motion)

The first issue studied with the videos was the distributions of the SADs. This can be important in a hardware solution so that the registers used to store SADs are of sufficient size to achieve correct results but not to waste extra hardware on bits that were not often used. To find these distributions, the video clips from Table 2.1 were simulated for best SAD values found using the 4SS algorithm. We implemented a software H.263 encoder/decoder to obtain the distribution of SAD solutions for these video clips. Table 2.2 shows the bit-lengths of the best SADs for each macroblock.

Table 2.2 - SAD Statistics for Basic Motion Estimation Algorithms

	Suzie		Carphone		Foreman	
SAD Bit-length	*TSS*	*4SS*	*TSS*	*4SS*	*TSS*	*4SS*
< 11 Bits	9138 (61.9%)	9134 (61.9%)	12947 (34.3%)	13052 (34.6 %)	7655 (20.6 %)	7658 (20.6 %)
11 Bits	5141 (34.8%)	5138 (34.8%)	16182 (42.9%)	15996 (42.4 %)	18217 (49.1 %)	18355 (49.4 %)
12 Bits	433 (2.93%)	437 (2.96%)	7443 (19.7%)	7517 (19.9 %)	10338 (27.8 %)	10094 (27.2 %)
13 Bits	42 (0.29%)	45 (3.05%)	1046 (2.78 %)	1051 (2.79 %)	873 (2.35 %)	960 (2.59 %)
14 Bits	0	0	98 (0.26 %)	101 (0.27 %)	45 (0.12 %)	61 (0.16 %)
15 Bits	0	0	0	0	0	0
16 Bits	0	0	3 (0.00 %)	5 (0.00 %)	0	0

From Table 2.2, it can be seen that most SAD ratings fall beneath the 13-bit (8192) range. A majority of SAD ratings fall even further below this into the 12-bit, 11-bit, and fewer ranges. For example, nearly 97% of the SAD solutions found in the *Suzie* video sequence can be represented with 11 bits and fewer. For the *Carphone* and *Foreman* sequences, 97% of SAD solutions can be represented using 12 bits and fewer. This suggests that the upper bits of the SAD are not necessary for all calculations and some hardware savings can be made in the sizes of registers, comparators, and associated logic in the processing elements and the control unit. These issues will be explored further in Chapter 4, which details low-power enhancements incorporated in the final motion estimation block design.

The second area of analysis deals with the actual motion vectors that are used by the video. With the types of video targeted for this system, small amounts of motion are expected within the pictures, meaning that most motion vectors found by the motion estimation unit should be close to the zero-vector. Table 2.3 shows the results of software simulations using the 4SS algorithm on the test video sequences. For the low-motion videos nearly 90% of all motion vectors are either the zero vector or one of the 8 vectors immediately surrounding it.

Table 2.3 - Partial Motion Vector Distributions for Test Video Sequences

	Zero Vector	Distance 1 from Zero Vector	Distance 2 from Zero Vector	Other
Suzie	68.8 %	21.1 %	4.83 %	5.27 %
Carphone	63.2 %	22.9 %	6.67 %	7.23 %
Foreman	44.8 %	26.2 %	10.7 %	18.3 %

These findings suggest that the 4SS algorithm would indeed be more effective as a motion estimation algorithm for these types of low-motion video. The suggestion can also be made that the 4SS algorithm can be further enhanced by biasing the first step of the algorithm toward the center vector and still get the correct vector the vast majority of the time. These observations are exploited in our low-power design for motion estimation block.

Chapter 3
Motion Estimation Block Design

The previous chapter introduced algorithms that have been used in motion estimation block designs along with advantages and disadvantages on using each algorithm. This chapter describes the designs based on some of those algorithms that were developed and simulated as part of this research.

Three algorithms were implemented to examine their applicability to a dedicated hardware solution and to perform comparative analysis with final results. The first algorithm considered is a Full-Search Block Matching design from [2]. The algorithm is included mostly for comparative purposes with our low-power design. As mentioned in the previous chapter, the full search algorithm performs too many calculations and takes up too much area, which renders it impractical for low-power design.

Secondly, a Three-Step Search (TSS) algorithm was implemented based upon the minimum hardware solution presented in [15]. The implementation will be compared and contrasted with the third algorithm implemented, the Four-Step Search (4SS) algorithm. The 4SS design is heavily based upon the TSS engine. These algorithms were selected for their applicability to a low-power solution for low-motion video.

This chapter presents the design and operation of these three algorithms. The three models are considered as the "baseline" models, to be used for comparison with each other and with later enhancements. The next chapter details the low-power enhancements made to the 4SS model based on observations made earlier and some previous research.

3.1 System Requirements

The desired performance characteristics of a motion estimation block should meet the following requirements.

- Perform motion estimation on 16x16 pixel macroblocks.
- Use a search area of [-7,+7].
- Perform motion estimation for 99 macroblocks per frame (QCIF-size) for 15 frames/sec or 1,485 macroblocks/second.

With these requirements the search area must include the previous 16x16 block plus 7 pixels to the left and right and 7 pixels above and below that block from the previous frame. This implies a 30x30 search area, which will be used for all the motion estimation blocks described in this chapter. The current frame search block and the previous frame search area are stored in local (to the motion estimation block) SRAMs. The partitioning of this memory can be different for different models and will be discussed in each respective section. An overview of the motion estimation process, including an illustration of the search area and the current frame reference block is included in Figure 3.1. The motion estimation blocks calculate SAD ratings for a set of motion vectors depending upon the algorithm being implemented. The lowest SAD and motion vector found are the outputs of the block. Table 3.1 shows the inputs and outputs of each motion estimation block along with a brief description of their operation. The usage of each signal inside the respective models will be discussed in the sections describing the models in detail.

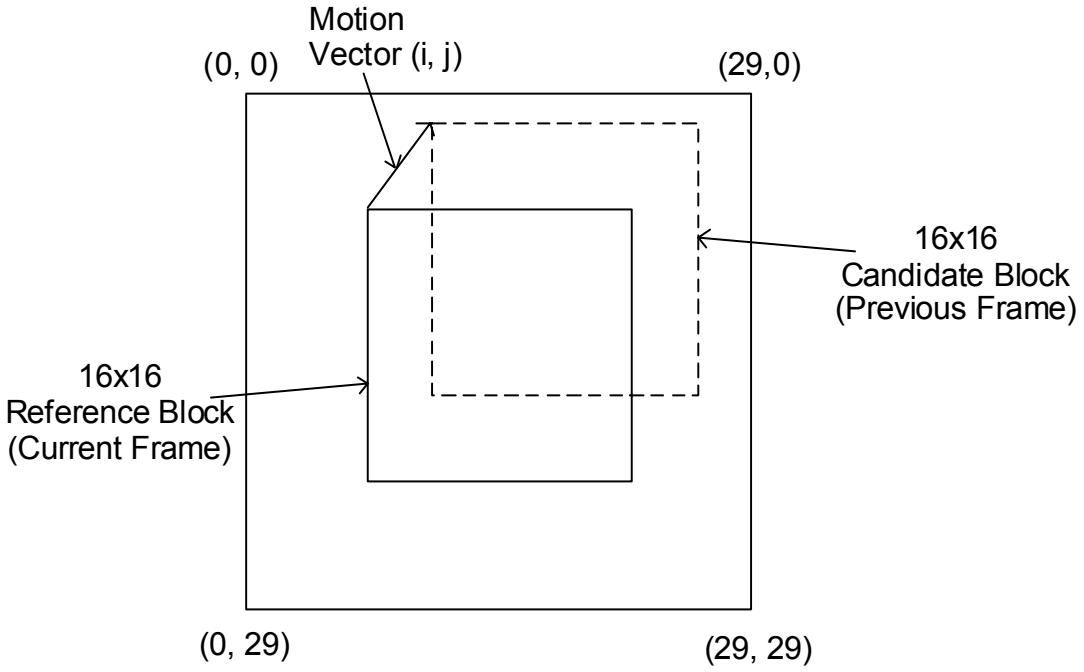

Figure 3.1 - Motion Estimation Overview

Table 3.1 - Input/Output Signals for All Models

Signal	Input/Output	Description
CLOCK	*Input*	Global clock for a model. Each model uses a single clock for all synchronous logic (registers, counters, etc.)
ENABLE	*Input*	Begins the circuit operation. The circuit will hold state if the ENABLE line is de-asserted.
RESET	*Input*	Global RESET. Will RESET all states, registers, and counters in the circuit.
FINISHED	*Output*	Informs external control circuitry to that MVX, MVY, and SAD are valid.
MVX	*Output*	Four bit-signed value that signifies the output motion vector in the x-direction. Valid values are from -7 to +7. Not valid unless FINISHED is also asserted.
MVY	*Output*	4-bit signed value that signifies the output motion vector in the y-direction. Valid values are from -7 to +7. Not valid unless FINISHED is also asserted.
SAD	*Output*	16-bit unsigned value that represents the SAD rating of the motion vector given by MVX and MVY. Not valid unless FINISHED is also asserted.

The initialization of local memories is not included in the design considerations or the power consumption numbers presented within. It is assumed that this process is uniform for the models and therefore not an important design objective. The focus of this research is on a comparative analysis of normal operation of the different motion estimation block designs.

3.2 Baseline Models

3.2.1 FSBM Baseline Model

As stated in the introduction to the chapter, the FSBM baseline model is based upon the one presented in [2]. More specifically, the arithmetic unit design and memory partitioning are used from the specifications found in that work. This model is implemented for later comparative analysis with the other implementations more suitable for low bit-rate video encoding. It also serves to illustrate the regularity and superior throughput of a FSBM solution for large-scale, computationally intensive motion estimation operations.

This model performs the Full-Search Block Matching motion estimation algorithm on 16x16 macroblocks within a search area of [-7, +7]. The design includes 15 processing elements (PEs), 3 local memories (one for the reference block and 2 for the search area), a comparator unit, and counters to control addressing into the memories. A block diagram of the entire system is shown in Figure 3.2.

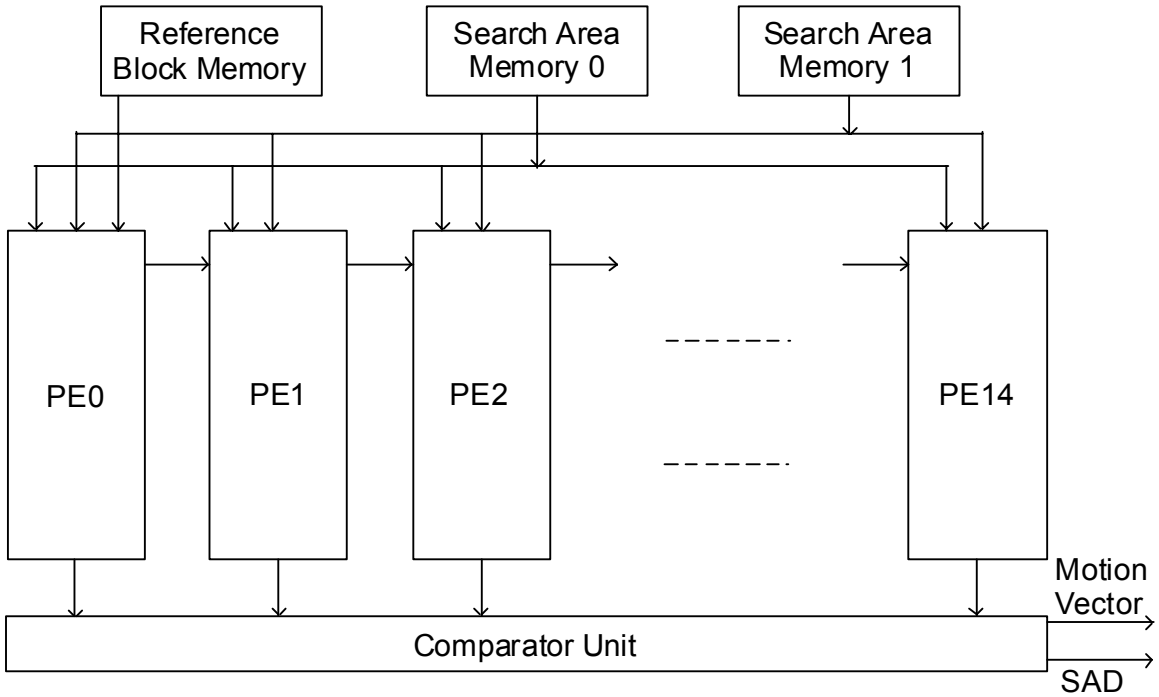

Figure 3.2 - Block Diagram of FSBM Model

The basic operation of the model is straightforward. Refer to the figure of the PE in Figure 3.3 and the pixel-flow table during operation in Table 3.2 for the following discussion. The PEs operate on a row of motion vectors at a time. Thus, the SADs for

motion vectors (-7, -7), (-7, -6), (-7, -5)…(-7, +7) are computed in the first operation of the circuit. Each PE receives three pixel values for each clock cycle, two search area pixels and a reference block pixel. The reference block pixels are shared between PEs through a delay register inside each PE. Note that only the first PE (PE0) is connected directly to the reference block memory. Then, the reference block pixel is shifted to its adjacent PE on the next clock. Each PE performs the basic SAD operation of computing the *absolute* difference of the two 8-bit input pixel values and accumulating it via an adder and a 16-bit register. All PEs are seeded and starting calculation after 14 clocks from first enabling the circuit. The first PE has a valid SAD ready for processing after 256 clocks. Each clock after that an adjacent PE has a valid SAD. Then, external logic resets the PE, and it is available for the next SAD. The comparator decides within a clock cycle if the newly computed SAD is lower than the lowest SAD found so far. If so, then the system saves it off including motion vector information. Otherwise, the system disregards the SAD and proceeds. When the counters indicate that all SADs have been calculated, the system asserts FINISHED and stops.

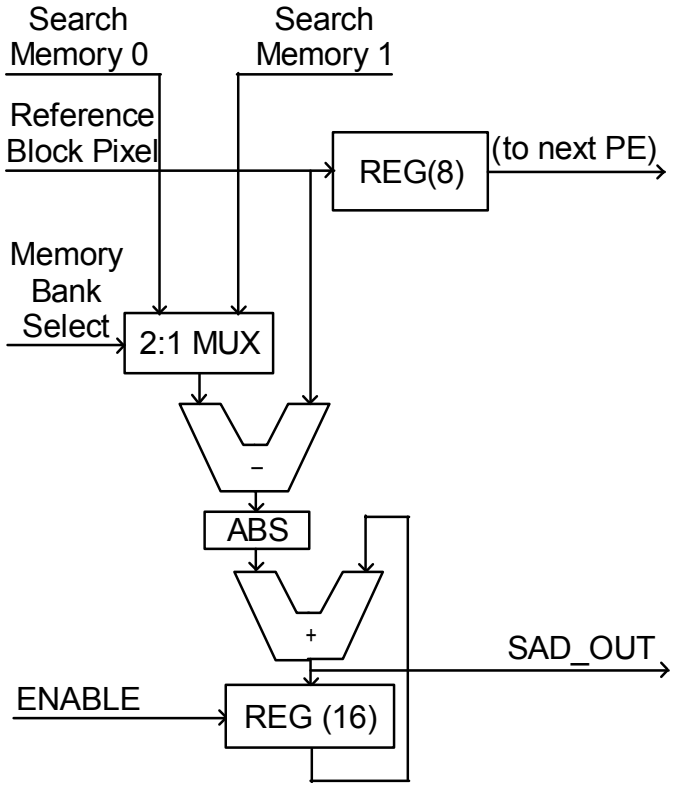

Figure 3.3 - FSBM Processing Element

Table 3.2 - Pixel Flow for FSBM Model [2]

Cycle	Input Data			Processor Inputs			
	Reference	Candidate 0	Candidate 1	PE0	PE1		PE14
0	$r_{0,0}$	$c_{0,0}$		$r_{0,0}$ $c_{0,0}$			
1	$r_{0,1}$	$c_{0,1}$		$r_{0,1}$ $c_{0,1}$	$r_{0,0}$ $c_{0,1}$		
2	$r_{0,2}$	$c_{0,2}$		$r_{0,2}$ $c_{0,2}$	$r_{0,1}$ $c_{0,2}$		
3	$r_{0,3}$	$c_{0,3}$		$r_{0,3}$ $c_{0,3}$	$r_{0,2}$ $c_{0,3}$		
-							
13	$r_{0,13}$	$c_{0,13}$		$r_{0,13}$ $c_{0,13}$	$r_{0,12}$ $c_{0,13}$		
14	$r_{0,14}$	$c_{0,14}$		$r_{0,14}$ $c_{0,14}$	$r_{0,13}$ $c_{0,14}$		$r_{0,0}$ $c_{0,14}$
15	$r_{0,15}$	$c_{0,15}$		$r_{0,15}$ $c_{0,15}$	$r_{0,14}$ $c_{0,15}$		$r_{0,1}$ $c_{0,15}$
16	$r_{1,0}$	$c_{1,0}$	$c_{0,16}$	$r_{1,0}$ $c_{1,0}$	$r_{0,15}$ $c_{0,16}$		$r_{0,2}$ $c_{0,16}$
17	$r_{1,1}$	$c_{1,1}$	$c_{0,17}$	$r_{1,1}$ $c_{1,1}$	$r_{1,0}$ $c_{1,1}$		$r_{0,3}$ $c_{0,17}$
18	$r_{1,2}$	$c_{1,2}$	$c_{0,18}$	$r_{1,2}$ $c_{1,2}$	$r_{1,1}$ $c_{1,2}$		$r_{0,4}$ $c_{0,18}$
-							
30	$r_{1,14}$	$c_{1,14}$	$c_{0,30}$	$r_{1,14}$ $c_{1,14}$	$r_{1,13}$ $c_{1,14}$		$r_{1,0}$ $c_{1,14}$
31	$r_{1,15}$	$c_{1,15}$	N/A	$r_{1,15}$ $c_{1,15}$	$r_{1,14}$ $c_{1,15}$		$r_{1,1}$ $c_{1,15}$
-							
255	$r_{15,15}$	$c_{15,15}$	N/A	$r_{15,15}$ $c_{15,15}$	$r_{15,14}$ $c_{15,15}$		$r_{15,1}$ $c_{15,15}$
256			$c_{15,16}$		$r_{15,15}$ $c_{15,16}$		$r_{15,2}$ $c_{15,16}$
257			$c_{15,17}$				$r_{15,3}$ $c_{15,17}$
-							
269			$c_{15,30}$				$r_{15,15}$ $c_{15,30}$

The next few paragraphs discuss the important pieces of the system in more detail. The discussions include the local memory, the address counters, the reset logic, and the comparator unit. Local memory is split up a bit differently than might be expected for this model. The reference block is maintained as a single memory of 256 bytes. The reference block pixels are arranged in the row-major order. The candidate block area memory, however, is split up to cover two 512 bytes blocks of memory. The memory partitioning is illustrated in Figure 3.4. Note that the leftmost 16x30 pixels are included in the left-hand memory (Area 0) and the remaining 14x30 pixels are included in the right-hand memory (Area 1). This enables the continuous operation of the PEs as illustrated above. Some of the larger memories are not completely utilized. These are marked with hatch markings inside the figure.

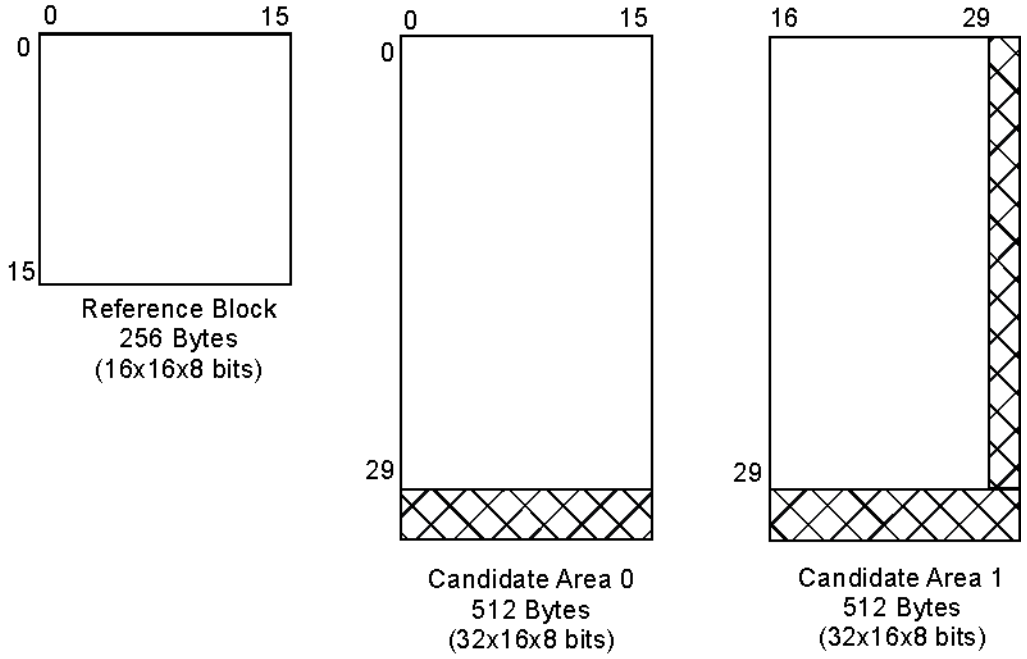

Figure 3.4 - Memory Partitioning for FSBM Model

The address counters act as the control logic for the system. Logic at the counter outputs dictates activity in most of the system. Clearly, the addresses also dictate the pixel values that flow from the memories to the PEs as described above. There are three memory addresses that must be created: a reference block address and two search area memory blocks.

Figure 3.5 shows the reference block address generation logic. Referring to Table 3.2, the reference block simply broadcasts the pixels in-order for the PE array to operate correctly. This means that as soon as the ENABLE line is asserted, this logic acts as a simple 8-bit counter. Then the reference block pixels are presented to PE0 in order and move through the delay chain to reach the rest of the PEs to achieve the pixel flow in Table 3.2.

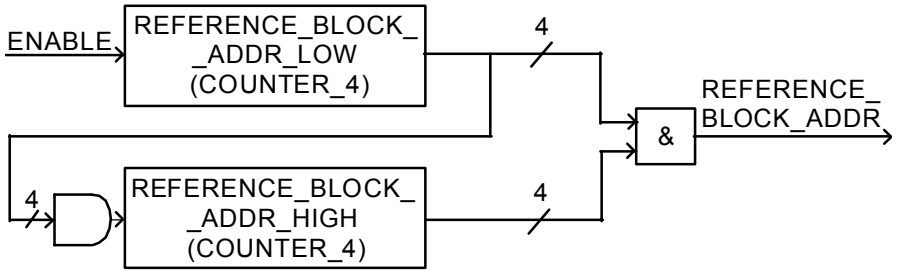

Figure 3.5 - Reference Block Address Generation Logic for FSBM Model

By necessity, the candidate search area logic is a bit more complex. Two addresses are computed, one for the left-hand memory (Area 0) and one for the right-hand memory (Area 1). The lower 4 bits of both addresses are created by the four-bit CANDIDATE_AREA_ADDR_X counter. This counter counts as long as the ENABLE line is asserted. To understand the derivation of the upper 5 bits, refer again to Table 3.2. For each row of motion vectors being calculated, there are 16 rows of the search area that must be calculated. These are represented by the CANDIDATE_AREA_ADDR_Y counter. This counter is reset for every row of motion vectors to calculate. (That condition is detected when the CANDIDATE_AREA_ADDR_X counter rolls over as seen in the block diagram.) CANDIDATE_AREA_ADDR_MVY is another counter that represents the scaling of the upper 5 bits in the Y direction as the Y-coordinate of the motion vector increases. This counter is incremented when both the ADDR_X and ADDR_Y counters roll over. ADDR_Y and ADDR_MVY are added to from the upper 5 bits of the left-hand memory address. According to Table 3.2, the right hand address shares some similarities with the left-hand address. The lower 4-bits (or relative x-offsets to the memory bank) are identical as mentioned previously. However, the y-offset of the right-hand address is a delayed version of the y-offset of the left-hand offset. Hence, the 5-bit register is included to save off the old y-offset in the logic below , as the x-counter rolls over. The output of this register is used as the upper 5 bits of the right-hand address.

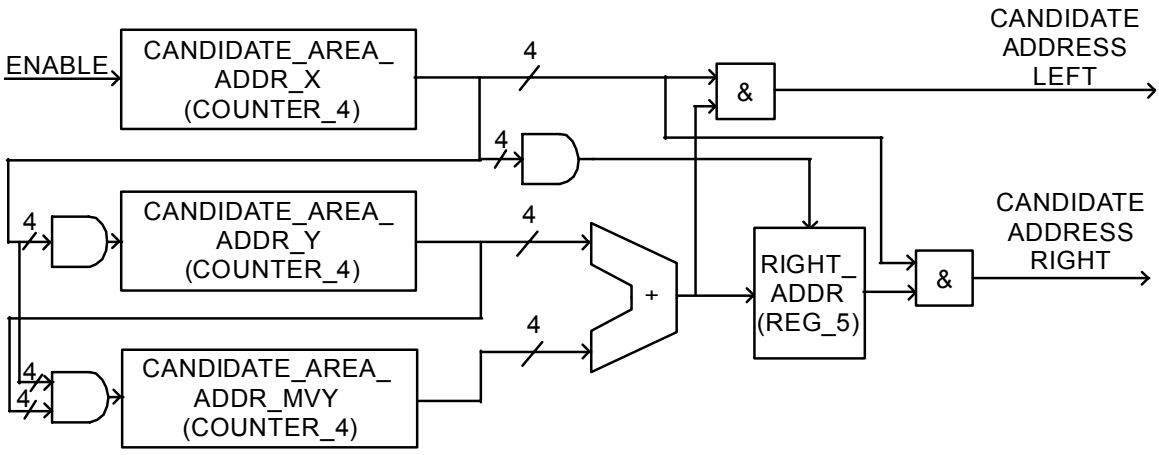

Figure 3.6 - Candidate Block Addresses Generation Logic for FSBM Model

Two important pieces of "control logic" are directly derived from these counters and are necessary for the correct operation of the circuit. The first is the generation of the RESET lines. Figure 3.7 illustrates the logic block. It consists of a 14-bit shift-register.

The ENABLE_MVY line is the ENABLE line for the ADDR_MVY counter above. As this counter is being enabled, the PEs are finishing off their respective SAD calculations. This bit is shifted through the shift register to reset the PEs in order. This RESET_PE value is also used to activate the comparator.

The other major control logic is the line to each PE to select which side of the candidate memory to use (Memory Bank Select signal of the PE in Figure 3.3). It turns out that this logic can be based entirely on the 4-bit ADDR_X counter. Referring back to Table 3.2, the pattern of which memory to use repeats every 16 clocks, coinciding with this counter. Table 3.3 gives the truth table of this logic for every PE.

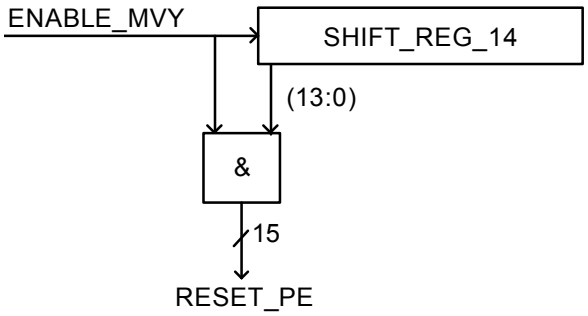

Figure 3.7 - PE Reset Logic for FSBM Model

Table 3.3 - Memory Select Truth Table

ADDR_X Counter	**Processing Element Memory Bank Select**														
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1
2	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1
3	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1
4	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1
5	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
6	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
7	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1
8	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1
9	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1
10	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
11	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
12	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

The final major portion of the FSBM model is the Comparator Unit. As mentioned in the description above, this unit is responsible for determining which SAD is the lowest value and saving both the SAD and the motion vector information. Figure 3.8 gives the block diagram for the Comparator Unit logic. The RESET_PE lines described above signal when a SAD is ready in the respective PE. The encoder encodes this code for the 15:1 MUX. The inputs to the 15:1 MUX are the 16 bit SAD values from the PEs. The SAD from the proper PE is selected and passed to the 16-bit comparator. If this is lower than the SAD stored in the 16-bit SAD_OUT register, then the register is enabled and the new SAD will be loaded at the next clock. In order to ensure that this SAD is valid, both the comparison must be valid (asserted) and one of the SAD PEs active (OR-gate in the logic). Also, if the comparison is valid, the motion vector registers (MVX and MVY) reload a new motion vector. The X-coordinate of the motion vector is derived from the encoded RESET_PE lines, and the Y-coordinate of the motion vector is derived from the CANDIDATE_AREA_ADDR_MVY counter.

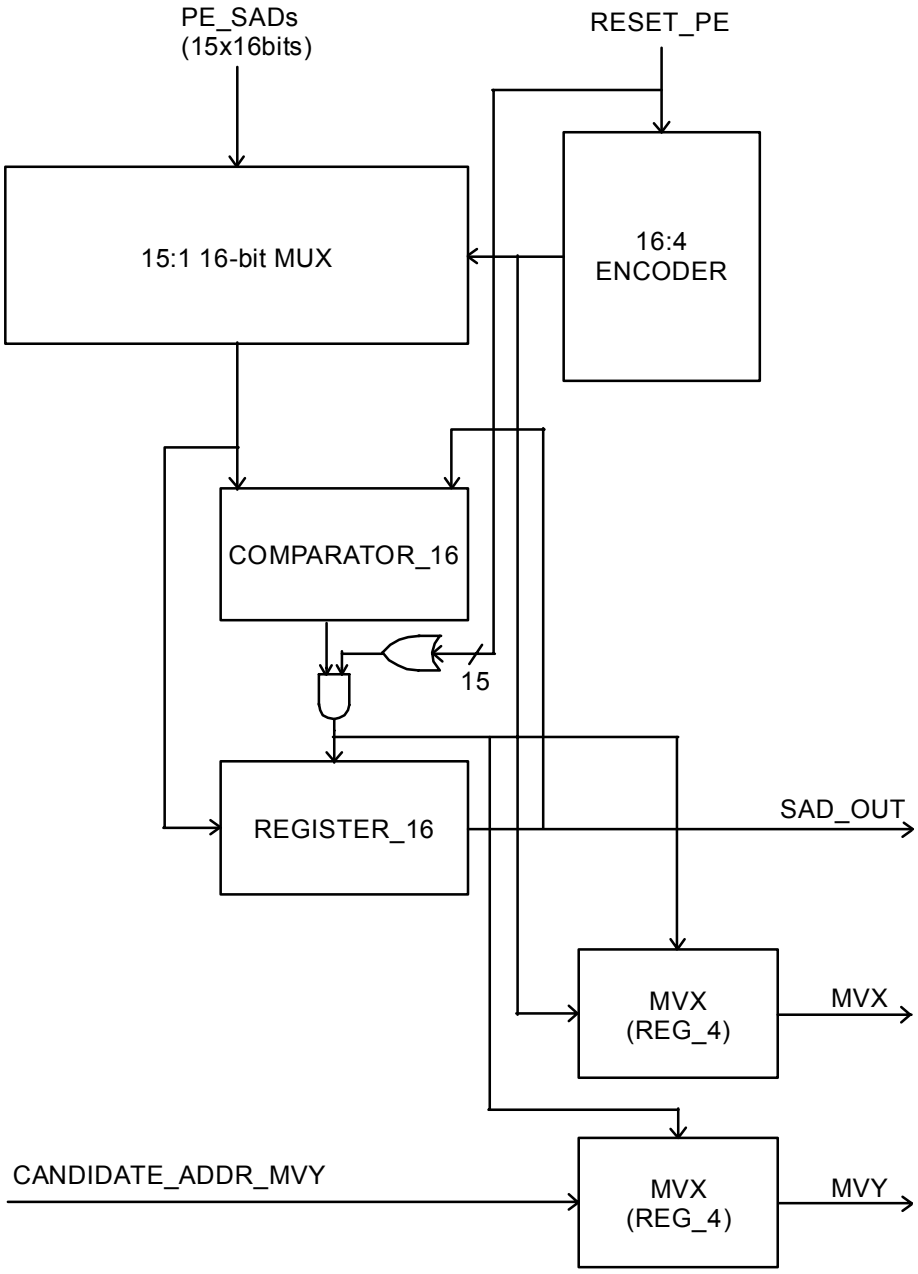

Figure 3.8 - Comparator Block for FSBM Model

To illustrate the timing of the SAD collection and RESET processes, a simple timing diagram is presented in Figure 3.9.

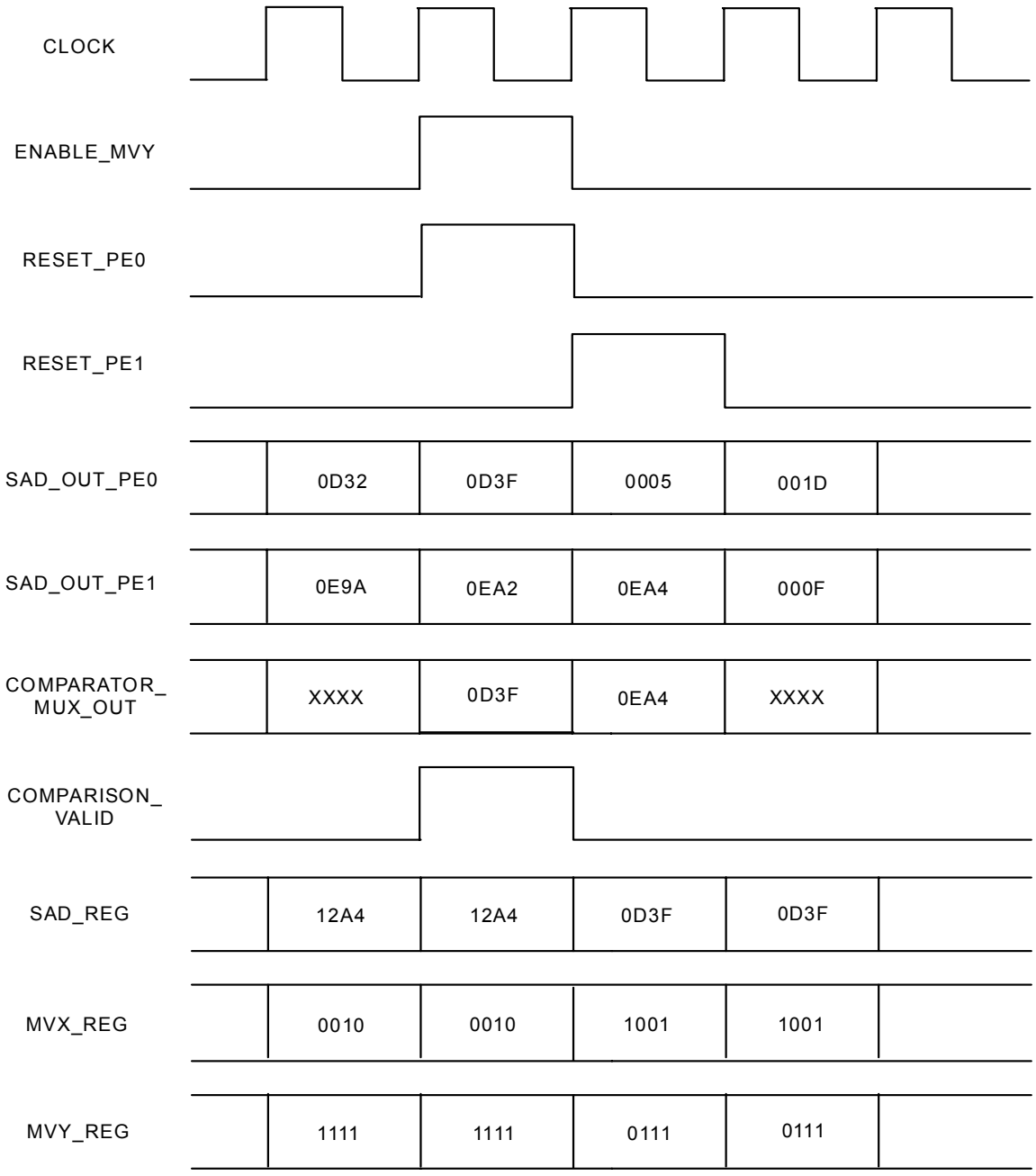

Figure 3.9 - Timing Diagram for Comparator Unit and PE Resets

This diagram depicts a typical situation where the counters dictate that the PEs are finishing a round of SADs and the Comparator Unit decides if the SADs beat the best SAD found so far. In this situation, the lowest SAD found so far is "12A4" at the motion vector (+2, -1). The process of comparison is triggered by the ENABLE_MVY line to enable the MVY counter to increment. At this point the SAD_OUT from PE0 is valid.

(Notice from Figure 3.3 that SAD_OUT is taken from the input to the accumulator register, allowing the SAD to be taken a cycle before it would appear at the register output and allowing more efficient use of the PE.) RESET_PE0 is now enabled as a result. Note that the register in the PE has a synchronous RESET operation with respect to this input, therefore, the reset will not occur until the start of the next clock cycle, preserving the SAD being used in this comparison. The MUX logic in the comparator passes 0D3F to the comparator logic, where it triggers a favorable comparison (COMPARISON_VALID enabled). Therefore, on the next cycle, the SAD_REG is loaded with the new SAD and the motion vector registers update with the new motion vector (-7, +7) for this example, since PE0 only figures SADs for motion vectors with a "-7" X-coordinate. The next clock cycle, these registers are updated. Also PE0 resets and begins calculating the next motion vector. The shift register now shifts the RESET_PE signal and RESET_PE1 is asserted. This SAD passes through the MUX logic but is larger than the new lowest SAD, so it does not trigger a favorable comparison. The next cycle occurs and the Comparator Unit registers are preserved, PE1 resets and begins its next calculation, and PE2's SAD is compared, and so on.

The parallel nature of the system allows generous margins in the clocking of the system for the necessary throughput goals of the motion estimator. Figuring the number of clocks to complete a motion vector is straightforward by referring to Table 3.2 regarding pixel flow. Covering a search area of (-7,+7) requires 225 SADs to be calculated. Each run through the pixel flow in Table 3.2 calculates 15 of those SADs. Thus, there are 15 of those operations to complete. Since the PEs are completely utilized without wasting cycles between SAD comparison and reset, these operations require 256 cycles for each of the 14 rows of motion vectors. The final row of motion vectors must "wait" until the last SAD is completed. Thus, this row of motion vectors takes 269 cycles to complete. This is a total of **3,853** clocks to compute a single motion vector. Computing the goal of 1,485 macroblocks per second requires 5,721,705 clocks per second or a clock rate of about **5.72 MHz**. These results are summarized in Table 3.4, including the number of memory accesses required during calculations.

Table 3.4 - Performance Statistics for FSBM Model

FSBM Statistic	Value
Clocks per Macroblock	***3,853***
Minimum Clock Rate for QCIF @ 15 fps	***~5.72 MHz***
Reference Block Memory Accesses per Macroblock	***3840***
Candidate Search Area Memory Accesses per Macroblock (Area 0)	***3840***
Candidate Search Area Memory Accesses per Macroblock (Area 1)	***3840***

3.2.2 TSS Baseline Model

The TSS baseline model is presented here as a more plausible approach to the motion estimation low channel bandwidth and low power budget. The model developed and described in this section is related to the minimum hardware-cost model presented by Costa, et. al in terms of PE structure and data flow [31]. We redesigned the control logic to simplify it to a single state machine.

Figure 3.10 gives an outline of the TSS baseline model. This model performs the TSS algorithm as described in [8]. The macroblock size is 16x16 and the search area considered is [-7, +7]. This design includes 3 processing elements (PEs), 2 local memories, and a control unit. The control unit includes the state machine to control the circuit, memory addressing counters, and a comparator unit.

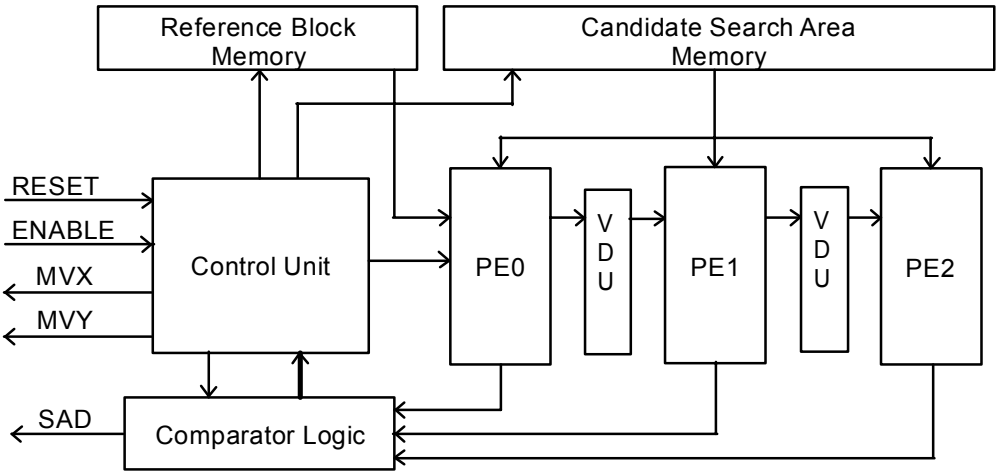

Figure 3.10 - Overview of TSS/4SS Baseline Model [31]

The circuit computes motion vectors and SADs according to the TSS algorithm as follows.

1. The memory counters are reset to the defaults of a zero vector and a FFFF SAD rating.

2. The PEs operate on a set of three motion vectors at a time, representing a row of vectors on a search box. For example, to being the TSS algorithm, PE0 begins calculations for (-4, -4), PE1 starts at (-4, 0), and PE2 starts at (-4, +4). Information on pixel flow and actual SAD calculation follows this overview.
3. As the PEs finish, the control unit enables the comparator to check each SAD. If it wins, the motion vector information is saved off for the next step.
4. The PEs begin work on the next 3 motion vectors of a search box. The overall order of calculating SADs of a search box is given in Figure 3.11 below.
5. If the search box is finished, the control logic determines if the algorithm is finished or if the next step of the algorithm is required. If the next step is required, the control unit changes the search box size and resets the memory counters to point to the origin of the new search box.
6. Repeat 2-5 until the algorithm is finished. Then the control unit asserts FINISHED and the SAD, MVX, and MVY outputs are valid.

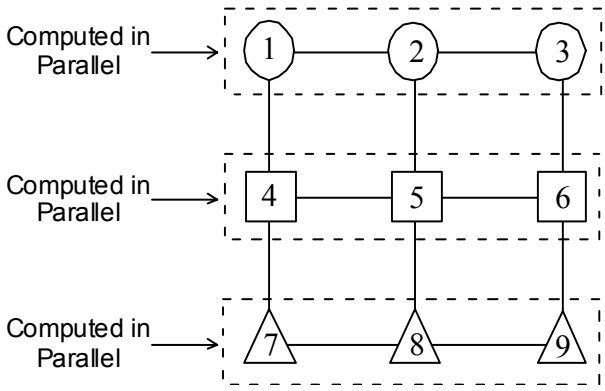

Figure 3.11 - Search Order for PEs in TSS/4SS Models [31]

The operation of the PEs is critical to the understanding of the circuit. The major pieces of logic are the PE itself and the variable delay units (VDUs). The VDUs enable the sharing of reference block pixels, in the same way as the delay latch enables the parallelism in the FSBM model. Block diagrams of the PE and the VDUs are included in Figures 3.12 and 3.13. Figures 3.14 and 3.15 detail the memory partitioning for this model. Only two local memories are used for the TSS model. One is 256 bytes and includes the reference block, arranged in order as 16 rows of 16 8-bit pixel values. The candidate search area is included in a single 1kB memory, arranged as 30 rows of 30 8-

bit pixel values to enable easier memory addressing, as will be described later. Table 3.5 reviews the pixel flow for the PEs during the first step of the algorithm, computing (-4,-4), (0, -4), and (+4, -4), as an example. The first candidate pixel addressed is (3,3), since (7, 7) represents the start of the zero vector block and the first motion vector is (-4, -4).

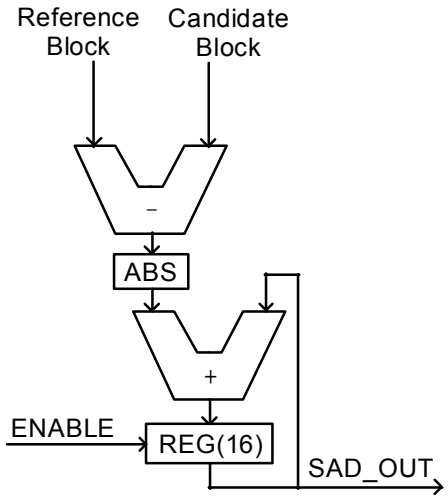

Figure 3.12 - TSS/4SS Baseline Model Processing Element

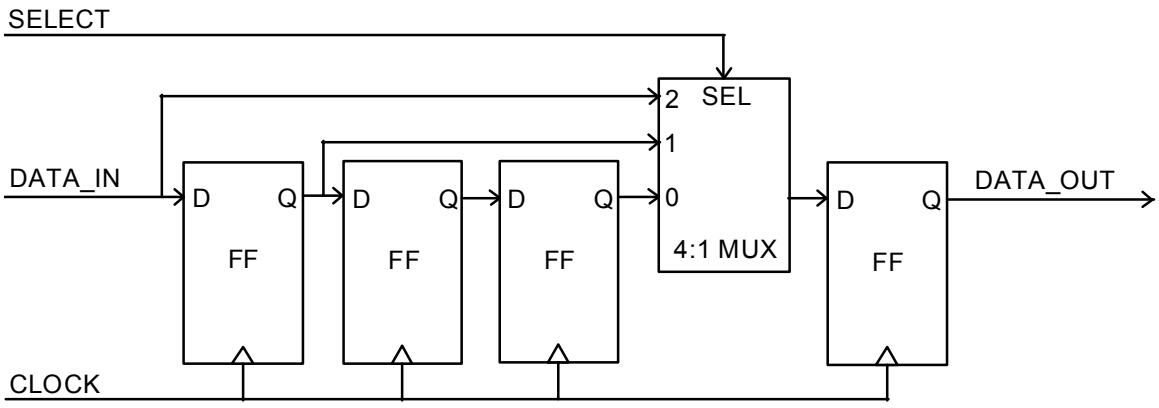

Figure 3.13 - TSS Variable Delay Unit

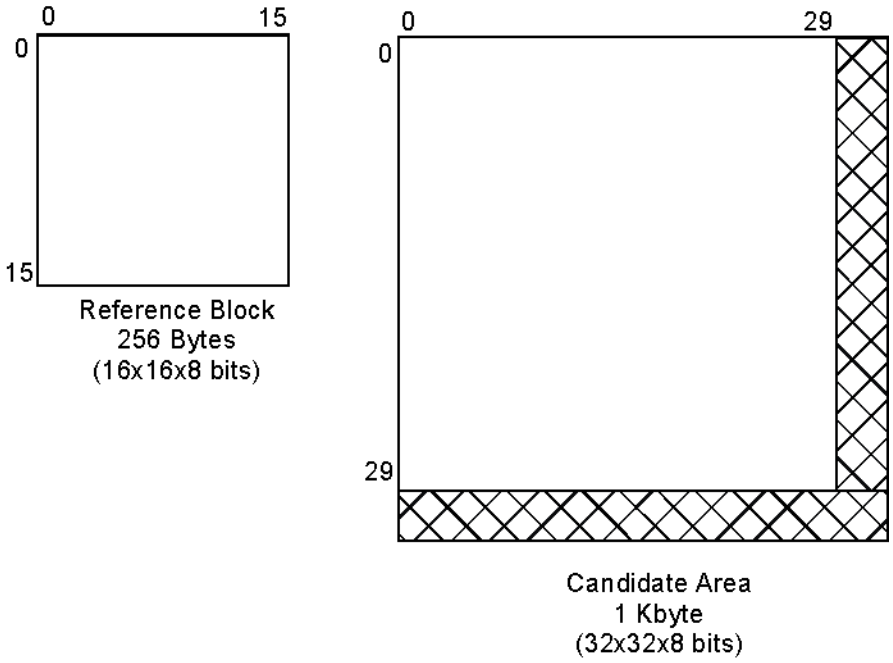

Figure 3.14 - TSS/4SS Memory Partitioning

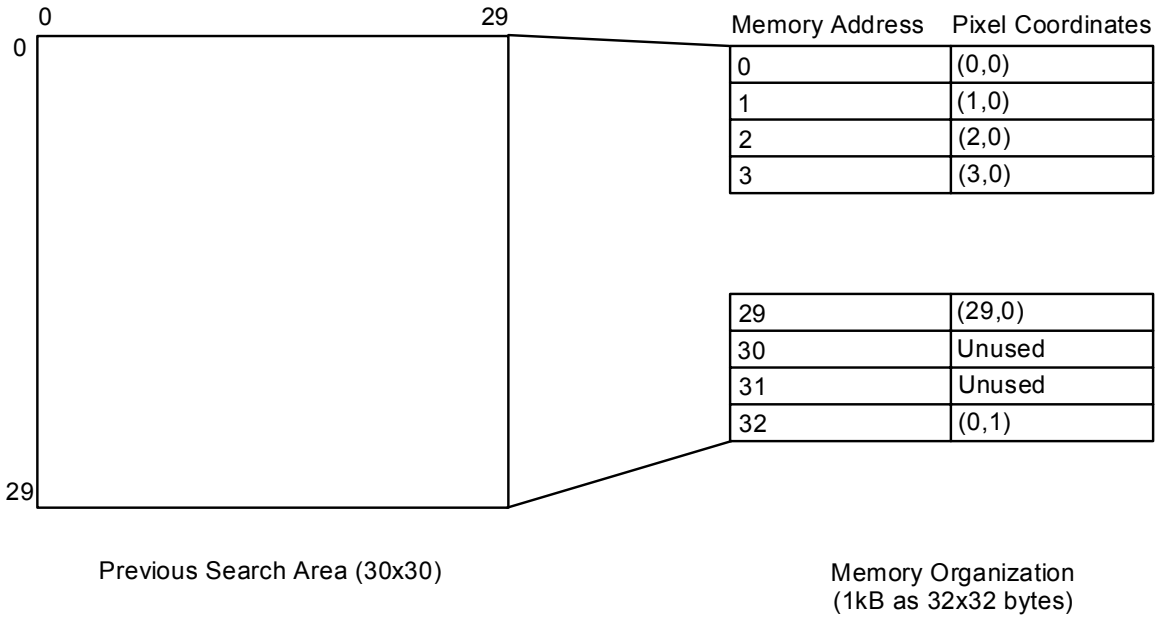

Figure 3.15 - TSS/4SS Memory Organization by Rows

Table 3.5 - Pixel Flow to PEs in TSS/4SS Baseline Models

Clock	Reference Memory	Candidate Memory	PE0	PE1	PE2
0	$r_{0,0}$	$c_{3,3}$	$r_{0,0}$ $c_{3,3}$	Disabled	Disabled
1	$r_{0,1}$	$c_{3,4}$	$r_{0,1}$ $c_{3,4}$	Disabled	Disabled
2	$r_{0,2}$	$c_{3,5}$	$r_{0,2}$ $c_{3,5}$	Disabled	Disabled
3	$r_{0,3}$	$c_{3,6}$	$r_{0,3}$ $c_{3,6}$	Disabled	Disabled
4	$r_{0,4}$	$c_{3,7}$	$r_{0,4}$ $c_{3,7}$	$r_{0,0}$ $c_{3,7}$	Disabled
5	$r_{0,5}$	$c_{3,8}$	$r_{0,5}$ $c_{3,8}$	$r_{0,1}$ $c_{3,8}$	Disabled
6	$r_{0,6}$	$c_{3,9}$	$r_{0,6}$ $c_{3,9}$	$r_{0,2}$ $c_{3,9}$	Disabled
7	$r_{0,7}$	$c_{3,10}$	$r_{0,7}$ $c_{3,10}$	$r_{0,3}$ $c_{3,10}$	Disabled
8	$r_{0,8}$	$c_{3,11}$	$r_{0,8}$ $c_{3,11}$	$r_{0,4}$ $c_{3,11}$	$r_{0,0}$ $c_{3,11}$
9	$r_{0,9}$	$c_{3,12}$	$r_{0,9}$ $c_{3,12}$	$r_{0,5}$ $c_{3,12}$	$r_{0,1}$ $c_{3,11}$
-					
14	$r_{0,14}$	$c_{3,17}$	$r_{0,14}$ $c_{3,17}$	$r_{0,10}$ $c_{3,17}$	$r_{0,6}$ $c_{3,17}$
15	$r_{0,15}$	$c_{3,18}$	$r_{0,15}$ $c_{3,18}$	$r_{0,11}$ $c_{3,18}$	$r_{0,7}$ $c_{3,18}$
16		$c_{3,19}$	Disabled	$r_{0,12}$ $c_{3,19}$	$r_{0,8}$ $c_{3,19}$
17		$c_{3,20}$	Disabled	$r_{0,13}$ $c_{3,20}$	$r_{0,9}$ $c_{3,20}$
18		$c_{3,21}$	Disabled	$r_{0,14}$ $c_{3,21}$	$r_{0,10}$ $c_{3,21}$
19		$c_{3,22}$	Disabled	$r_{0,15}$ $c_{3,22}$	$r_{0,11}$ $c_{3,22}$
20		$c_{3,23}$	Disabled	Disabled	$r_{0,12}$ $c_{3,23}$
21		$c_{3,24}$	Disabled	Disabled	$r_{0,13}$ $c_{3,24}$
22		$c_{3,25}$	Disabled	Disabled	$r_{0,14}$ $c_{3,25}$
23		$c_{3,26}$	Disabled	Disabled	$r_{0,15}$ $c_{3,26}$
24	$r_{1,0}$	$c_{4,3}$	$r_{1,0}$ $c_{4,3}$	Disabled	Disabled
25	$r_{1,1}$	$c_{4,4}$	$r_{1,1}$ $c_{4,4}$	Disabled	Disabled
-					
382			SAD Ready	SAD Ready	$r_{15,14}$ $c_{18,25}$
383			SAD Ready	SAD Ready	$r_{15,15}$ $c_{18,26}$
384			SAD Ready	SAD Ready	SAD Ready

In this example, the VDU would be set for a four-cycle delay on the reference block pixels. This effectively implements the search box for step 1 of the TSS algorithm. Other delay modes (2 and 1 cycle delay) implement different sized search boxes for steps 2 and 3 of the TSS algorithm. The other two important considerations with maintaining the circuit include restarting the reference block counters and disabling the PEs at the correct points in time. The first consideration is handled by the control unit and will be described in more detail below. The ability to enable or disable a PE is another important

consideration. This is accomplished by adding another bit to the VDUs for an ENABLE line. By enabling the first PE for the first 16 clocks, then sending PEs 1 and 2 a "delayed" version of this, PE1 will be enabled for clocks 4-19, and PE 2 will be enabled for clocks 8-23. This strategy of using the VDUs also scales down as the search boxes get smaller.

The control unit contains addressing counters, system state machine, and comparator logic. A block diagram of the system is included in Figure 3.16. The finite state machine generates the control signals for the system. The XCOUNTER and YCOUNTER are used to address the memory and inform the finite state machine when certain states are finished. The finite state machine, comparator unit, and address generation logic are described in more detail in the following.

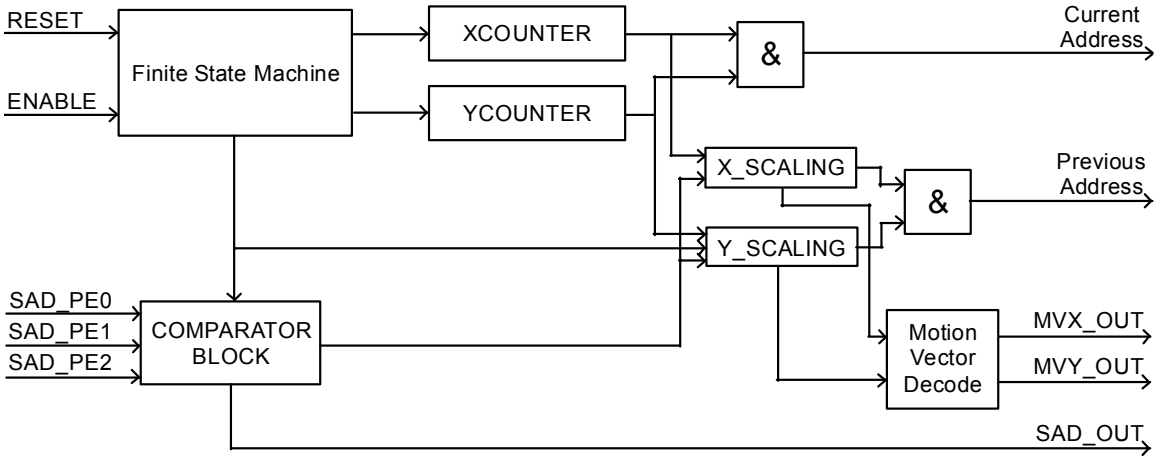

Figure 3.16 - TSS/4SS Control Logic Overview

The most important signals of the TSS control unit are the DELAY and STEP signals. The STEP signal is a 2-bit counter that informs the control unit which row of a current search box is being executed. The DELAY signal is a 2-bit counter that informs the control unit which step of the TSS algorithm is being executed. The reason for it being labeled DELAY is that this signal is used as the input to the VDUs. This counter counts up from 0 to 1 to 2 for the three steps of the TSS algorithm. Figure 3.17 shows the logic to generate these two important signals. ENABLE_DELAY and ENABLE_STEP are generated by the finite state machine. RESET_STEP is used to reset the STEP counter (when a new search box is being started) and is also generated by the finite state machine.

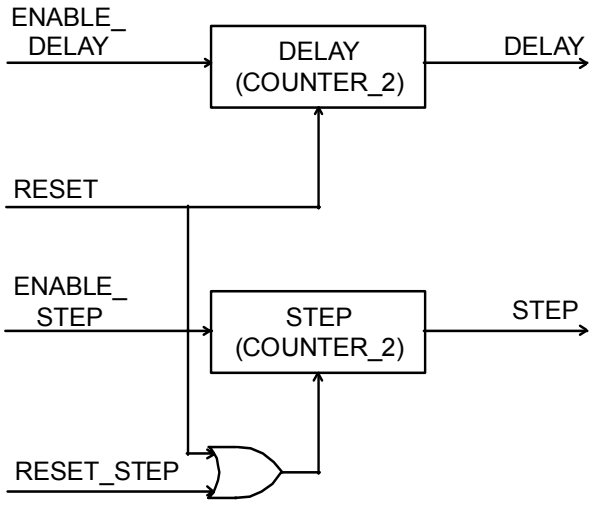

Figure 3.17 - TSS DELAY and STEP Generation Logic

The finite state machine acts as the controlling device for the system, generating all control signals. It is responsible for the timing of the control signals for the address counters, the comparator unit, the processing elements and all other logic blocks in the system. The state machine has 8 distinct states, described in Table 3.6 along with the conditions that will switch state. The following paragraphs describe each of the states of the finite state machine in more detail.

Table 3.6 - State Descriptions for TSS/4SS Models

State	State Name	Operation	Next State Condition
0	Reset	Initialize all counters, scaling factors, best SAD rating.	If ENABLE asserted, proceed to State 1. Otherwise hold.
1	Count	Enable counters and PEs.	XCOUNTER reaches end of row of pixels, then proceed to State 2. If XCOUNTER has not reached threshold, then repeat State.
2	Counter Reset	Reset address counters.	YCOUNTER determines if SAD calculations are finished. If not, proceed to State 1. Otherwise proceed to State 3.
3	Evaluate PE0	Activate PE0 input to Comparator.	Always proceed to State 4.
4	Evaluate PE1	Activate PE1 input to Comparator.	Always proceed to State 5.
5	Evaluate PE2	Activate PE2 input to Comparator.	Always proceed to State 6.
6	Decision	Reset PEs. Update counters for step of algorithm and/or row of search box. Update scaling factors if a step of TSS/4SS algorithm completed.	If algorithm finished, go to State 7. Otherwise proceed to State 1.
7	Finished	Enable Motion Vector output logic, hold SAD output.	If RESET asserted, proceed to State 0. Otherwise hold.

The Count state (State 1) is the state in which the address counters are activated and counting and the PEs enabled. This is the only state in which SAD calculation takes place. More specifically, the XCOUNTER is activated to enable the x-directional counting to achieve the pixel flow of Table 3.5. Only SADs for one row of pixels are calculated at a time for an "iteration" of State 1. The next state from State 1 is State 2, where the counters are reset. However, the threshold for switching from State 1 to State 2 depends upon which Step of the TSS algorithm is being computed. Note that the different size of the search box requires a different number of pixels to compute the entire row. Table 3.7 summarizes the values of XCOUNTER that move the state machine from State 1 to State 2.

Table 3.7 - XCOUNTER Threshold for Moving From State 1 to State 2 for TSS Algorithm

TSS Algorithm Step	Search Box Size	XCOUNTER Threshold
1	9x9	23
2	5x5	19
3	3x3	15

The Counter Reset state (State 2) resets the XCOUNTER to enabling moving back to the left-hand side of a row of pixels. This state enables the YCOUNTER, so that the SAD calculations move +1 pixel in the y-direction. Counter Reset usually then proceeds back to the Count state (State 1) to enable PE operation again, unless the SAD calculations for the row of the search box are completed. The completion condition, thus, is that YCOUNTER = 15, indicating all rows of the SADs were calculated. Then PEs are ready for comparison by the comparator unit, leading to the next states.

States 3, 4, and 5, the "Evaluate PE0", "Evaluate PE1", and "Evaluate PE2" states respectively, simply enabling that PE's input to the Comparator Unit. During these states, the comparator logic determines if any of these SADs beat the current best SAD found. Figure 3.18 shows a block diagram of the TSS/4SS comparator logic. The PE_VALID_n inputs generated by the finite state machine are used to enable the MUX that enables visibility of the correct PE's SAD. During State 3, PE_VALID0 is asserted. For State 4, PE_VALID1 is asserted and so on. If the SAD wins, then the new motion vector information and SAD are saved off as new scaling factors and COMPARISON_VALID is asserted.

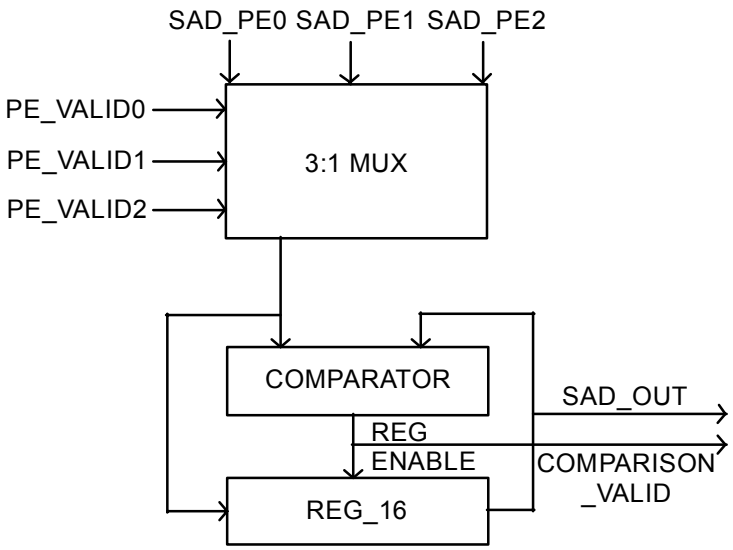

Figure 3.18 - TSS/4SS Comparator Unit

The most critical state is State 6, the Decision State. The logic must determine in this state the following questions:

- Is the algorithm completely finished?
- Or is the search box finished, meaning a step of the algorithm has been completed?
- Or is the search box incomplete and requires at least one more step of processing?

Based upon these questions a pseudocode for the logic can be derived, as in the following:

```
if (STEP == 2 and DELAY == 2) then
     /* algorithm is finished */
     proceed to state 7; /* Finished step */
elsif (STEP == 2) then
     /* search box is finished, go to next step of
algorithm */
     increment DELAY; /*Assert ENABLE_DELAY */
     reset STEP; /* Assert STEP_RESET */
     proceed to state 1; /* Count state */
else
     /* search box not finished */
     increment STEP; /* Assert STEP_ENABLE */
     proceed to state 1; /* Count state */
end if;
```

Figure 3.19 - Pseudocode for Decision State Logic of TSS Baseline Model

At stage 7, the motion estimation operation is complete. All counters in the circuit are disabled, and computation stops. The SAD being asserted by the comparator unit is the optimum according to the TSS algorithm. The MVX and MVY outputs are valid. MVX and MVY are calculated via a combinational logic block that translates the BASEX and BASEY outputs to their MVX and MVY equivalents.

Table 3.8 - Translation from BASEX and BASEY to MVX and MVY

BASEX, BASEY	MVX, MVY
0000 (0)	1001 (-7)
0001 (1)	1010 (-6)
0010 (2)	1011 (-5)
0011 (3)	1100 (-4)
0100 (4)	1101 (-3)
0101 (5)	1110 (-2)
0110 (6)	1111 (-1)
0111 (7)	0000 (0)
1000 (8)	0001 (1)
1001 (9)	0010 (2)
1010 (10)	0011 (3)
1011 (11)	0100 (4)
1100 (12)	0101 (5)
1101 (13)	0110 (6)
1110 (14)	0111 (7)
1111 (15)	XXXX

The address generation logic must address both the current frame reference block memory and previous frame candidate area memory. The reference block address generation is easy. Referring back to Table 3.5, the reference pixels must be presented a row at a time as the Count state begins. Therefore, as shown in Figure 3.16, the reference block address is simply the concatenation of the lowest 4 bits of the YCOUNTER and the lowest 4 bits of the XCOUNTER. As YCOUNTER is implemented with each time that State 2 is reached, all reference block pixels are reached for each row of the search box.

The candidate area memory address is more complex. There are two "fields" of the candidate area address, the row indicator, or upper five bits that index the row of the search area, and the column indicator, or lower five bits that index one of the 30 pixel positions within a row. Figure 3.20 illustrates the components of the candidate area address. The YCOUNTER and XCOUNTER are described above and represent the stepping through the candidate areas as the PEs calculate SADs. The derivation of the STEPY and STEPX are mentioned below.

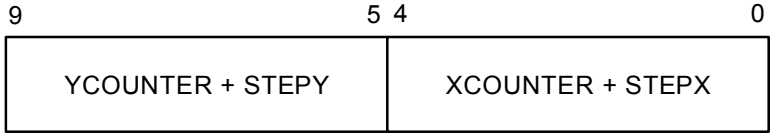

Figure 3.20 - Formation of a Candidate Area Address

However, the question remains of how to start the address in the correct position. This is where the "SCALING" logic blocks presented earlier come into use. Figure 3.19 illustrates the scaling logic. The BASEX and BASEY registers hold the coordinates representing the center of the search box in the candidate area. Therefore, on reset, they are set to 7 (0111), which are the starting candidate areas coordinates for the motion vector (0, 0). There are two important functions to be considered with the scaling block. The first is the scaling of the base addresses during normal operation, and the second is the procedure by which a new base address is assigned should a new motion vector win during operation. These are described in the next two paragraphs.

The "Base Address Scaling Logic" block performs address scaling based upon the BASEX, BASEY, STEP, and DELAY inputs. The X-coordinate, BASEX, is shifted to the left (subtracted) by either 4, 2, or 1 positions depending upon which step of the algorithm is being executed (i.e. input signal DELAY). The rules for Y-coordinate shifting depend also upon which row of the search box is being executed (i.e. input signal STEP). If the first row is being executed, a factor should be subtracted from the BASEY. For the second row, no change to BASEY is needed. For the final row, the factor should be added to BASEY. The factor mentioned in these cases is also 4, 2, or 1 depending upon input signal DELAY. The logic for this block has been synthesized based upon the preceding rules. The resulting factors are STEPX and STEPY. When these are added to the XCOUNTER and YCOUNTER, respectively, they form the two fields for the candidate area address mentioned in Figure 3.18.

The scaling for the new addresses use the rest of the logic shown in Figure 3.21. NEWBASEX and NEWBASEY hold the new values for BASEX and BASEY. BASEX and BASEY load these values only when ENABLE_BASE is asserted by the state machine. (This occurs when a new step of the algorithm is being executed, i.e. when STEP = 2 and the Decision state reached.) The NEWBASEX and NEWBASEY are loaded with new values when the COMPARISON_VALID signal is asserted signifying a new low value from the comparator unit. First, look at the NEWBASEX input. It can either represent BASEX plus or minus a factor based upon the step of the algorithm (4, 2, or 1). Notice that these scaling factors are prefigured for each BASEX. The results are passed through the 3:1 multiplexer to the NEWBASEX register depending on the

STATE. The STATE is decoded to determine which PE (and therefore which coordinate of the search box) is being accessed. The Y-shifting occurs in almost the same fashion. Except this time, the STEP input is used in the mutliplexer to determine which row of the search box is being looked at. That will determine which direction to shift the BASEY value.

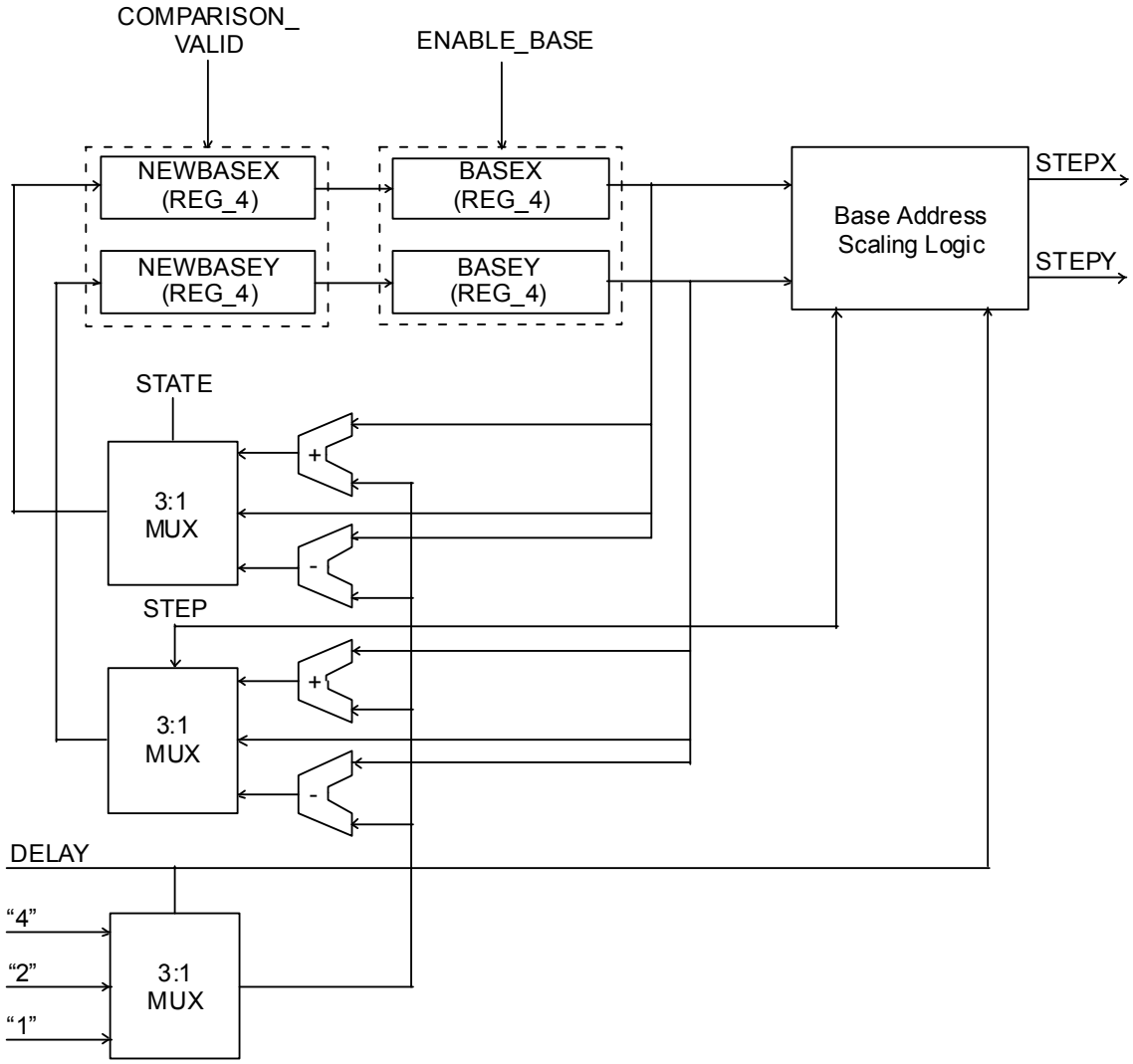

Figure 3.21 - Block Diagram of TSS Scaling Unit

The parallel computation of the PEs and reduced workload from the TSS allow a minimum clocking requirement close to that of the FSBM model. The algorithm must complete 3 steps of the algorithm. Table 3.9 shows exactly how many cycles of each state must be executed for each step of the TSS algorithm.

Table 3.9 - Number of Cycles for Each State Per Step of TSS Algorithm

State	State Name	Cycles for Step 1	Cycles for Step 2	Cycles for TSS Step 3
0	Reset	1	0	0
1	Count	24x16x3 (1152)	20x16x3 (960)	18x16x3 (864)
2	Counter Reset	1x16x3 (48)	1x16x3 (48)	1x16x3 (48)
3	Evaluate PE0	1x3 (3)	1x3 (3)	1x3 (3)
4	Evaluate PE1	1x3 (3)	1x3 (3)	1x3 (3)
5	Evaluate PE2	1x3 (3)	1x3 (3)	1x3 (3)
6	Decision	1x3 (3)	1x3 (3)	1x3 (3)
7	Finished	0	0	1
Total		**1213**	**1020**	**925**

Therefore, **3,158** clock cycles are required for a single motion estimation operation. At the desired throughput of the system, this demands **4,689,630** clocks per second or a frequency of about **4.7 MHz** as the minimum clock frequency.

The issue of memory accesses becomes somewhat more complex to answer due to the nature of the different size search boxes in the algorithms. The simple logic exists inside the model to turn off memory when it is not needed. An example can be seen in Table 3.5 for the Reference memory from clocks 16-23 and so on. For the reference memory, only the 16x16 reference block is read per iteration of a row of motion vectors of the search box. This is repeated then 3 times for a search box. Considering another factor of 3 for the steps of the TSS algorithm and the number of reference memory accesses is given by 16x16x3x3 = **2,304 accesses**. The number of candidate area memory accesses differs according to each step of the algorithm. For the first step, 24 accesses are required for each row, for the second step, 20 accesses, and 18 for the third. Therefore, multiplying by the 16 and 3 factors for each step of the algorithm yields:

- 24x16x3 = 1,152 accesses for Step 1.
- 20x16x3 = 960 accesses for Step 2.
- 18x16x3 = 864 accesses for Step 3.

The total number of candidate area memory accesses for the TSS model is **2,976 accesses.**

Table 3.10 summarizes the performance characteristics for the baseline TSS model derived above.

Table 3.10 - Performance Statistics for TSS Model

TSS Statistic	Value
Clocks per Macroblock	***3,158***
Minimum Clock Rate for QCIF @ 15 fps	***~4.7 MHz***
Reference Block Memory Accesses per Macroblock	***2304***
Candidate Search Area Memory Accesses per Macroblock	***2976***

3.2.3 4SS Baseline Model

The 4SS baseline model performs the 4SS motion estimation algorithm on a group of pixels as defined by Po and Ma [11]. The implementation of this model is nearly identical to the TSS model. The general structure is exactly the same. The major differences between these models are highlighted in this section. Also, performance and clocking requirements differ so those are presented in this section as well.

The following steps summarize the operation of this model as it performs a motion estimation operation (differences from the TSS model are highlighted in italics).

1. The memory counters are reset to the defaults of a zero vector and a FFFF SAD rating.
2. The PEs operate on a set of three motion vectors at a time, representing a row of vectors on a search box. *For example, PE0 begins calculations for (-2, -2), PE1 starts at (-2, 0), and PE2 starts at (-2, +2).*
3. As the PEs finish, the control unit enables the comparator to check each SAD. If it wins, the motion vector information is saved off for the next step.
4. The PEs begin work on the next 3 motion vectors of a search box. The overall order of calculating SADs of a search box is the same as in the TSS model (Figure 3.11 above).
5. If the search box is finished, the control logic determines if the algorithm is finished or if the next step of the algorithm is required. If the next step is required, the control unit changes the search box size and resets the memory counters to point to the origin of the new search box. *This step will differ from the*

TSS model in that the algorithm must proceed to the fourth step if the center vector of a search box wins.

6. Repeat 2-5 until the algorithm is finished. Then the control unit asserts FINISHED and the SAD, MVX, and MVY outputs are valid.

Obviously, one of the major differences between the TSS and 4SS models is that the 9x9 search box is not used in the 4SS algorithm. Steps 1-3 of the 4SS algorithm use a 5x5 search box and the 4th step uses the 3x3 search box. Recall that the VDU selects the size of the search box being used for each step of the algorithm. Therefore, the 4-cycle delay mode to implement the 9x9 search box is unnecessary in the 4SS model. This simplifies the 4SS VDU to a device that delays pixel values by either 1 or 2 clocks as shown in Figure 3.22.

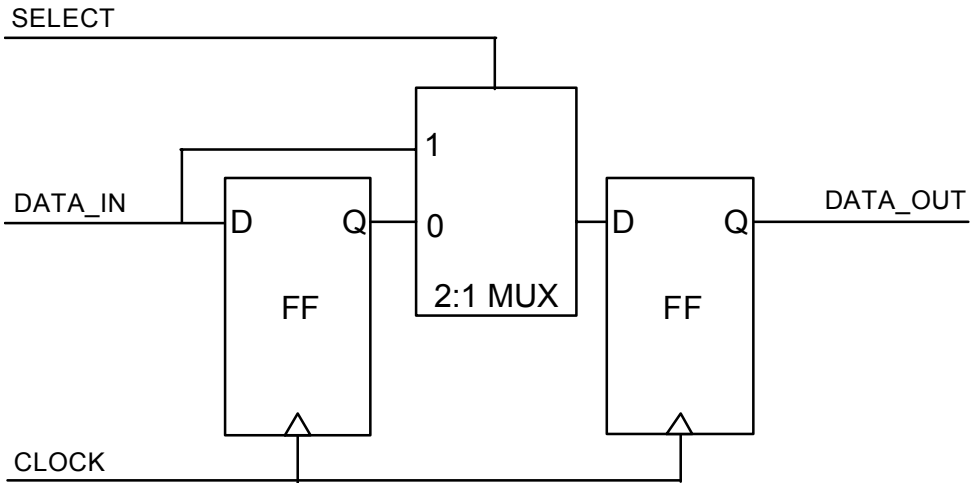

Figure 3.22 - 4SS Model VDU

The DELAY input still represents the size of the search box, but now it cannot be used to tell when the algorithm is completed. In the TSS model, the DELAY could be incremented by asserting ENABLE_DELAY every time the STEP counter reads 2 at the Decision state, indicating that the search box was finished. In this case however, the search box size (DELAY) being incremented represents moving to Step 4 of the 4SS algorithm. This can only happen if

1. Three search boxes have already been searched, or
2. The previous search box winner was the center of the search box.

Figure 3.23 shows the DELAY and STEP generation logic for the 4SS baseline model reflecting these new restrictions. STEP is generated as before, incremented or reset at the

Decision state by the finite state machine. DELAY, however, is not a counter, but a 1-bit register. The register is reset to 0 upon reset, setting the VDUs to implement a 5x5 search box. A 2-bit counter (COUNT in the figure) is used to keep track of the number of iterations of the 5x5 search box. When this hits "2" and STEP hits "2", the algorithm must jump to Step 4 so the DELAY_REG input is asserted. The other way to assert that input line is via the GO_STEP_4 signal. GO_STEP_4 is a piece of combinational logic that asserts under the following condition:

$$(BASEX = NEWBASEX) \cdot (BASEY = NEWBASEY) \quad (1)$$

In other words, this signal shows that there is no change in the new base coordinates about to be loaded into the BASEX and BASEY registers. If this condition is true when the STEP input resets, or a new "step" of the algorithm is entered, then the center of the previous search box has won and the algorithm should proceed to step 4.

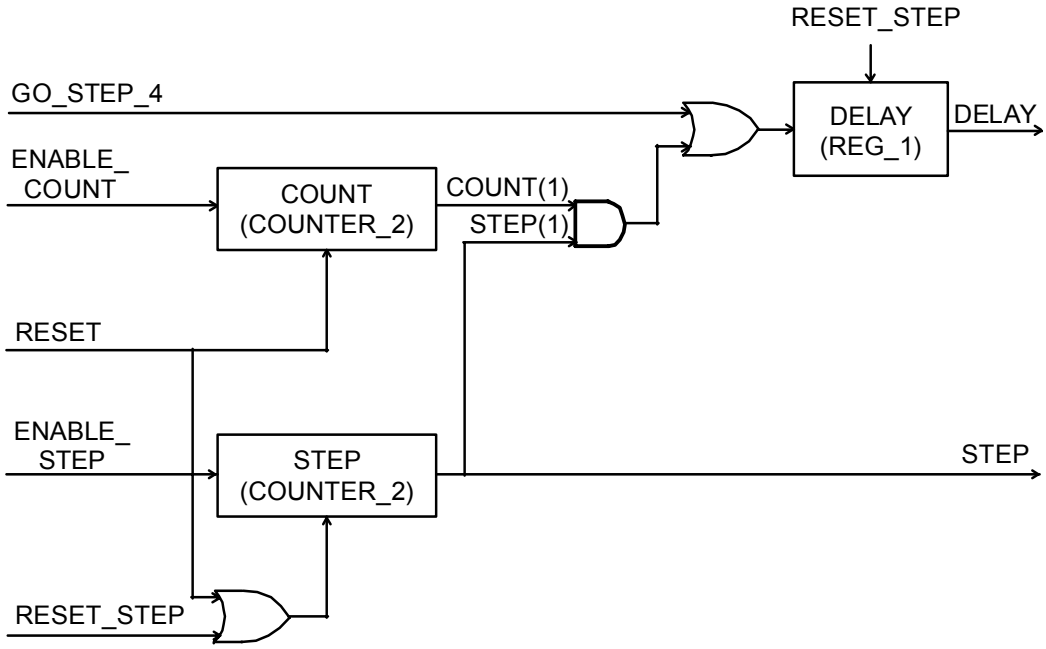

Figure 3.23 - 4SS DELAY and STEP Generation Logic

Another difference shows up in the address scaling logic. A block diagram of this logic for the 4SS baseline model is presented in Figure 3.24. Notice that its operation is identical to that of the TSS model with one simplification. The scaling factors used to add or subtract from the base addresses in both the new vector scaling logic and the "Base Address Scaling Logic" only require factors of either 2 or 1 depending on the

DELAY signal input. The reason is again due to the fact that the 9x9 search box with a search point distance of 4 is not used in the 4SS model.

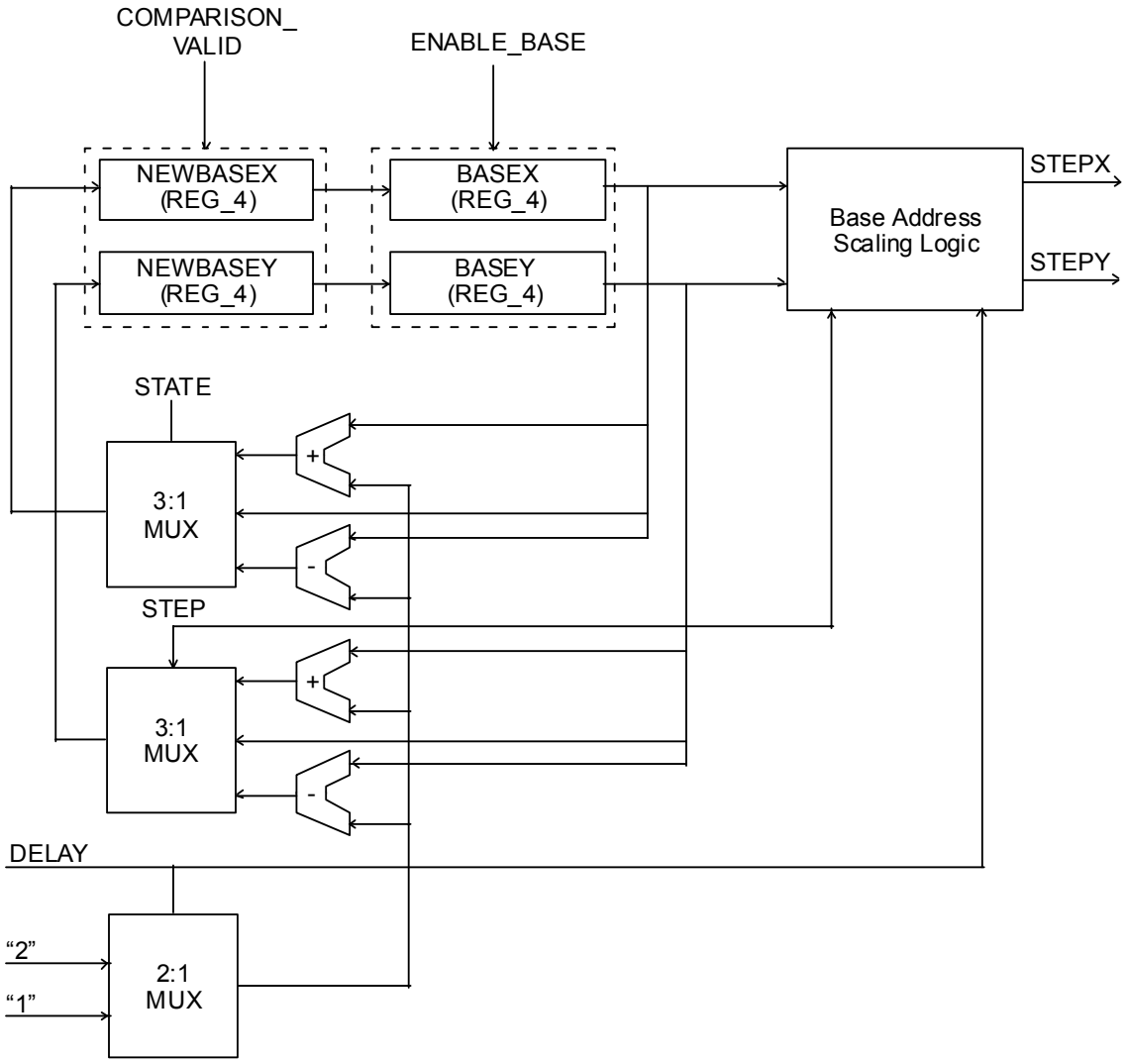

Figure 3.24 - Block Diagram of TSS Scaling Unit

Finally, a change must be made in the finite state machine for determining what should happen upon reaching the Decision state. As mentioned previously, the logic for the DELAY register handles switching between the search box sizes and therefore handles which "Step" of the 4SS algorithm the circuit currently is executing. This can simplify the pseudocode for the next state logic at the Decision state. This is summarized in the following figure (Figure 3.25).

```
if (STEP == 2 and DELAY == 1) then
     /* algorithm is finished */
```

```
        proceed to state 7; /* Finished step */
    elsif (STEP == 2) then
        /* search box is finished, go to next step of
    algorithm */
        reset STEP; /* Assert STEP_RESET */
        increment COUNT; /* Assert ENABLE_COUNT */
        proceed to state 1; /* Count state */
    else
        /* search box not finished */
        increment STEP; /* Assert STEP_ENABLE */
        proceed to state 1; /* Count state */
    end if;
```

Figure 3.25 - Pseudocode for Decision State Logic of 4SS Baseline Model

When calculating performance statistics and minimum clock requirements for the 4SS model, the fact that the length of time of computation is not set as in the previous two algorithms. There are 3 distinct lengths of time that can be observed, depending upon whether the algorithms aborts early in the 1st or 2nd steps or runs through its entirety. Table 3.11 lists the number of cycles for each step of the 4SS algorithm with this implementation. Notice that basically these are the same as those lengths of time for the TSS algorithm steps 2 and 3 for the 5x5 and the 3x3 search boxes.

Table 3.11 - Number of Cycles for Each State Per Step of 4SS Algorithm

State	State Name	Cycles for Step 1	Cycles for Step 2	Cycles for Step 3	Cycles for Step 4
0	Reset	1	0	0	0
1	Count	20x16x3 (1152)	20x16x3 (960)	20x16x3 (960)	18x16x3 (864)
2	Counter Reset	1x16x3 (48)	1x16x3 (48)	1x16x3 (48)	1x16x3 (48)
3	Evaluate PE0	1x3 (3)	1x3 (3)	1x3 (3)	1x3 (3)
4	Evaluate PE1	1x3 (3)	1x3 (3)	1x3 (3)	1x3 (3)
5	Evaluate PE2	1x3 (3)	1x3 (3)	1x3 (3)	1x3 (3)
6	Decision	1x3 (3)	1x3 (3)	1x3 (3)	1x3 (3)
7	Finished	0	0	0	1
Total	----------	**1021**	**1020**	**1020**	**925**

Therefore, for a single motion estimation operation **1,946, 2966,** or **3986** clock cycles are required. The worst-case condition must be used to define the minimum clocking for this algorithm. At the desired throughput of the system, **5,919,210** clocks per second are needed or a frequency of about **5.9 MHz** as the minimum clock frequency.

Memory accesses can be solved using the examples given for the TSS baseline model above. The reference area memory will always be accessed 16x16x3 = 768 times for each step of the 4SS algorithm, regardless of the search box size. For the candidate area, since steps 1-3 of the 4SS algorithm directly correlate to Step 2 of the TSS algorithm with the 5x5 search box, steps 1-3 require 20x16x3 = 960 accesses. The fourth step of the 4SS algorithm uses a 3x3 search box, therefore 18x16x3 = 864 accesses are required. The total number of candidate area memory accesses for the 4SS model can be **1824, 2784 or 3744 accesses**.

Table 3.12 summarizes the performance characteristics for the baseline TSS model derived above. Note that the table is split according to how many steps of the 4SS algorithm are used for each motion estimation operation.

Table 3.12 - Performance Statistics for 4SS Model

4SS Statistic	2 Steps	3 Steps	4 Steps
Clocks per Macroblock	***1,946***	***2,966***	***3,986***
Minimum Clock Rate for QCIF @ 15 fps	***~5.9 MHz***	***~5.9 MHz***	***~5.9 MHz***
Reference Block Memory Accesses per Macroblock	***1,536***	***2,304***	***3,072***
Candidate Search Area Memory Accesses per Macroblock	***1,824***	***2,784***	***3,744***

Chapter 4
Low-Power Enhancements

The motion estimation block can consume a large amount of power. The necessary components for a motion estimator include counters to address memory, large memories to store pixel values, arithmetic units to calculate SADs, comparators, registers, and control logic. The complex block demands a good deal of power. To make the problem worse, the motion estimation block is used nearly all the time. After the first frame of a video sequence, motion estimation is applied to nearly every macroblock of every frame of the video. In fact, high-throughput motion estimation blocks have been estimated to consume nearly 50% of the power dissipated in a video encoder [6].

While the motion estimation block used for the targeted low-bit rate video encoder is not as high-throughput as one for a high quality, high frame rate video system, the demand of portable operation places a strain on the power dissipation of the block. The proposed motion estimation design presented in this thesis must consume as little power as possible. Optimizations are measured by reduction of power consumed by the block or in reduction of memory accesses that occur during the operation.

Based upon power characterizations of the baseline models and the study of targeted videos from Chapter 2, the four-step search model was chosen as the baseline for the low-power enhancements. This chapter presents four major enhancements and some minor design alterations to the baseline 4SS model to reduce power consumption. Although most of these enhancements would work better for any low bit-rate video, the final proposed design is optimized for low-motion video targeted by this system. Those topics will be discussed as well as a rationale for each enhancement and a detailed description of the implementation.

The first two alterations, removing redundant calculations and halting of SAD calculations are purely design-related to the implementation of the 4SS algorithm. In other words, they do not alter the output of the circuit in any way but should result in a net power saving. The final two methods presented in this chapter, zero-biasing the search and reducing bit-length of registers, alter the 4SS algorithm slightly.

4.1 Redundant Search Removal

As the 4SS algorithm proceeds to the 2nd and 3rd steps of the algorithm, many of the motion vector SAD calculations are unnecessary. Those SADs have already been computed and are found to be non-optimal. If those calculations can be removed in subsequent steps, a power savings can be achieved. For an example, refer to Figure 4.1. When moving from the 1st to the 2nd step and from the 2nd to the 3rd step of the algorithm, many of the calculations required by the new search box are redundant. In this example, the motion vector at (0, -2) wins the first step, so the 2nd step is required. Placing the 5x5 search box at the new center of (0, -2) renders **6** of those 9 vectors to be redundant (marked by R1 in the figure). In the best case, illustrated by the 3rd-step search centered at (+2, -4) **4** searches are redundant (marked by R2) in the figure. Generally speaking, 6 searches are redundant when the side of a search box wins, and 4 searches are redundant when selecting the corner of the search box. For larger motion that uses most of the 4 available steps of the 4SS algorithm, this can lead to wasted power and an average computational complexity greater than that of the TSS algorithm.

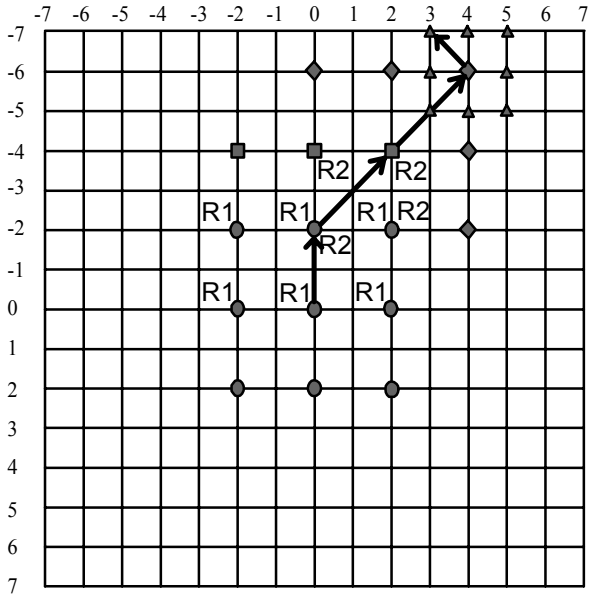

Figure 4.1 - Illustration of 4SS Redundant Calculations

In our low-power design, logic has been added to detect this case and remove the redundant SAD calculations from the subsequent search step. A special bit code is saved off if a PE comparison is valid (i.e. the SAD of that PE is the lowest found so far.) This code is used the next search step to gate the ENABLE fed to each PE. When a PE does

not receives the ENABLE signal, it gates off all inputs to "0" and disables its accumulator register, effectively stopping all switching activity for that SAD calculation for that row of motion vectors. Figure 4.2 shows a block diagram of the logic.

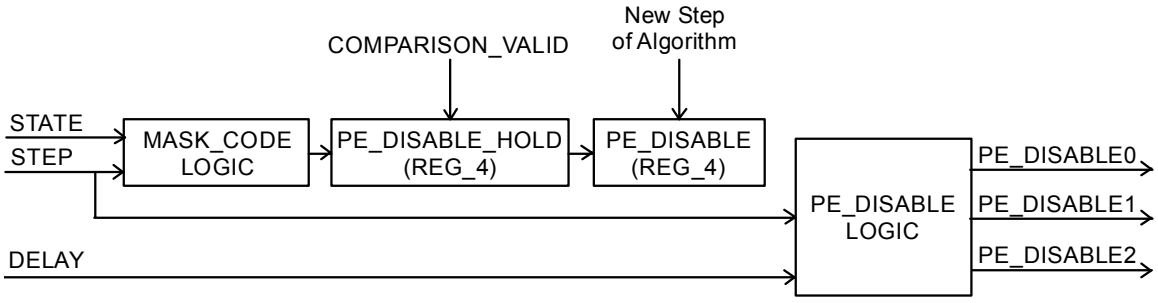

Figure 4.2 - Block Diagram of Logic to Remove Redundant Calculations

As mentioned above, the code for enabling the PE_DISABLE lines is stored in the 4 bit register PE_DISABLE. A two register "pipeline" structure implements this functionality. Recall that COMPARISON_VALID is asserted when a valid comparison has been made. The STATE and STEP information determine exactly which PE in the search box generated the successful comparison, so the MASK_CODE logic can generate the correct 4-bit code. STEP indicates the row of the search box being calculated (the y-coordinate). STATE of the control unit will determine which of the 3 PEs is being active in the comparator unit at that moment (x-coordinate). Figure 4.3 illustrates this indexing of the correct PE. VHDL code was synthesized to produce the proper mask code to identify which PE has won in the previous step. PE_DISABLE_HOLD is then loaded by being enabled with the asserted COMPARISON_VALID signal. This mask code then is loaded into the next register in the chain, PE_DISABLE, when the algorithm shifts to the next step. This logic equation to derive the signal "New Step of Algorithm is given as equation 1 below, the condition being that STATE is 6 (the Decision state) and the STEP is 2 (the last row of the search box).

$$NewStep = STATE2 \cdot STATE1 \cdot \overline{STATE0} \cdot STEP1 \quad (1)$$

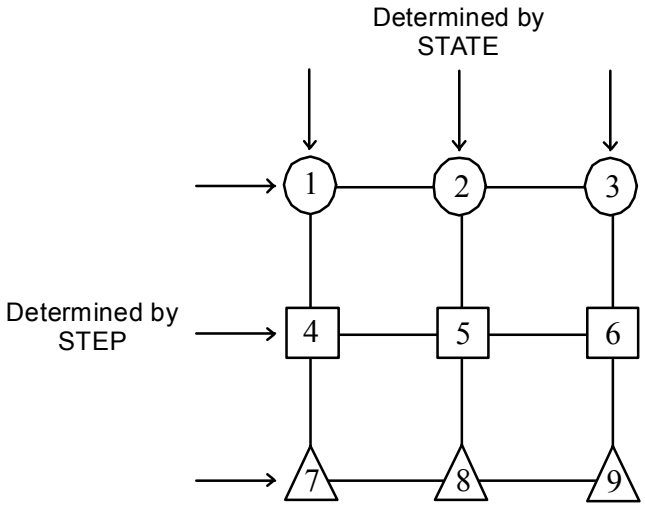

Figure 4.3 - Illustration of Indexing into Search Box to Determine Winning PE for Generation of Redundant Search Removal MASK_CODE

At the next step of the algorithm, this 4-bit code, which gives the winning PE of the previous step, is fed to combinational logic (PE_DISABLE Logic) to produce the final signals that can turn off a PE if necessary. Note again that the DELAY line indicates the size of the search box being used. STEP is also needed since the PE_DISABLE signals are different depending upon which row of the search box is being calculated. For the 3x3 search box (step 4 or DELAY is 1), the PE_DISABLE logic defaults to calculating all motion vectors except the center one. This differs from the 5x5 search box case where turning off different PEs is necessary depending upon the winner. Table 4.1 gives the truth table for the PE_DISABLE logic. Notice that a "1" in the table means that the PE is *enabled*. A "0" means the PE should be shut off.

The PE_DISABLE lines are logically combined with the PE_ENABLE lines generated for each the PEs as shown in Figure 4.4. If a PE_DISABLE line is asserted, then the PE is disabled. The final PE_ENABLE signal is deasserted for the length of that calculation.

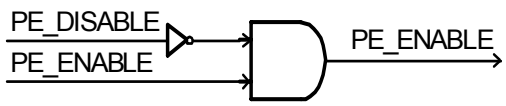

Figure 4.4 - PE_ENABLE Generation Logic with PE_DISABLE Signal

Table 4.1 - Truth Table for PE_DISABLE Logic

		PE_DISABLE (STEP = 0)			PE_DISABLE (STEP = 1)			PE_DISABLE (STEP = 2)		
Winning PE	**DELAY**	*0*	*1*	*2*	*0*	*1*	*2*	*0*	*1*	*2*
1	**0**	1	1	1	1	0	0	1	1	1
2	**0**	1	1	1	0	0	0	0	0	0
3	**0**	1	1	1	0	0	1	0	0	1
4	**0**	1	0	0	1	0	0	1	0	0
5	**0**	1	1	1	1	0	1	1	1	1
6	**0**	0	0	1	0	0	1	0	0	1
7	**0**	1	0	0	1	0	0	1	1	1
8	**0**	0	0	0	0	0	0	1	1	1
9	**0**	0	0	1	0	0	1	1	1	1
X	**1**	1	1	1	1	0	1	1	1	1

A final consideration must be made for the comparator unit. The SADs for the disabled PEs are 0, since all computations are disabled for that PE. The logic for the comparator unit needs to be altered from the baseline model to prevent this SAD from being used. Figure 4.5 shows the logic of the enhanced comparator.

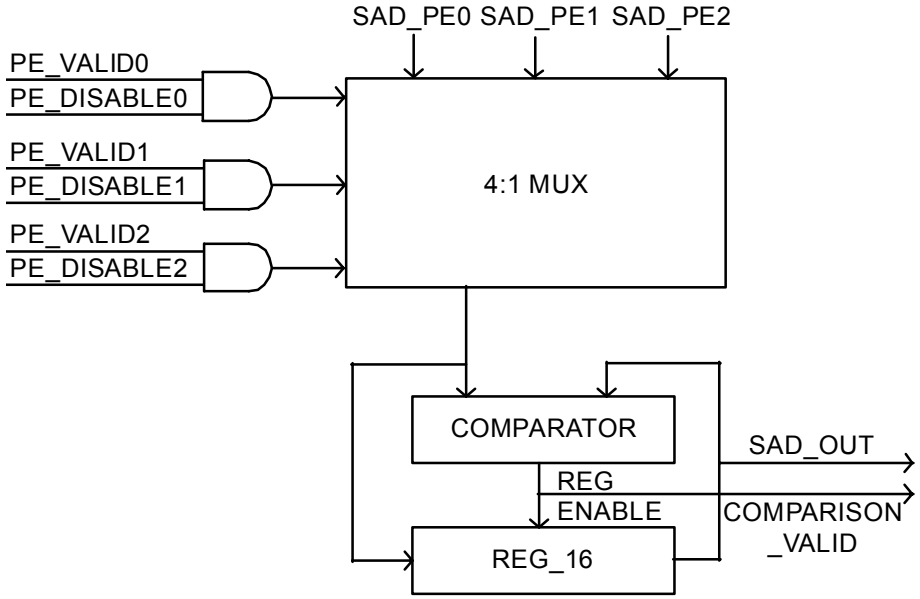

Figure 4.5 - Comparator Unit with Redundant Search Removal Capability

The major difference between the new comparator unit and the old unit lies in the multiplexer and the inputs from the state machine to the mutliplexer. The multiplexer has

been altered so that when none of the three inputs are asserted, the output is FFFF, or the largest possible number, guaranteeing that a positive comparison will NOT occur. The PE_VALID_n signals are gated into the multiplexer by the respective $PE\_DISABLE_n$ signal. The "negative logic" of the PE_DISABLE lines enable this. As an example, look at the case where the PE_DISBALE mask for a particular SAD calculation is "001". This means that PE0 and PE1 are disabled and will have a SAD value of 0 and PE2 is enabled and will have a valid SAD after calculations. During evaluation of the PEs, PE_VALID0 and PE_VALID1 are asserted during states 3 and 4 respectively. However, since the PE_DISABLE signals are deasserted (the PE_DISABLE mask does not "reset" to the next value until Decision state) the multiplexer does not see a valid input and asserts FFFF as the SAD value for those PEs. During state 5, the PE_DISABLE2 and PE_VALID2 are both asserted, and the proper SAD is applied to the comparator to determine if it should be used.

4.2 Comparator SAD Stop

The 4SS algorithm only requires an accurate SAD calculation if that SAD has beaten the best SAD so far. Therefore, if a SAD calculation is obviously not going to beat the best SAD found so far, any additional calculations of that SAD are completely unnecessary. This scheme is most effective for smaller motion video, when an optimum SAD can be found quickly, and more calculations in the PEs can be skipped.

This method is implemented with a simple comparator built-in to each PE. Figure 4.6 shows the new processing element with comparator logic. The "best" SAD found so far is supplied to each PE (represented by the signal SAD_IN in the figure). If the calculated SAD for that PE increases to a value larger than the best SAD found previously, the comparator flags a signal that is held until the PE is reset again. This signal will force the PE to gate its inputs, again effectively shutting off switching activity. Since the SAD value is obviously not going to win, it can be used without modification by the controller during the evaluation steps without concern for altering final results.

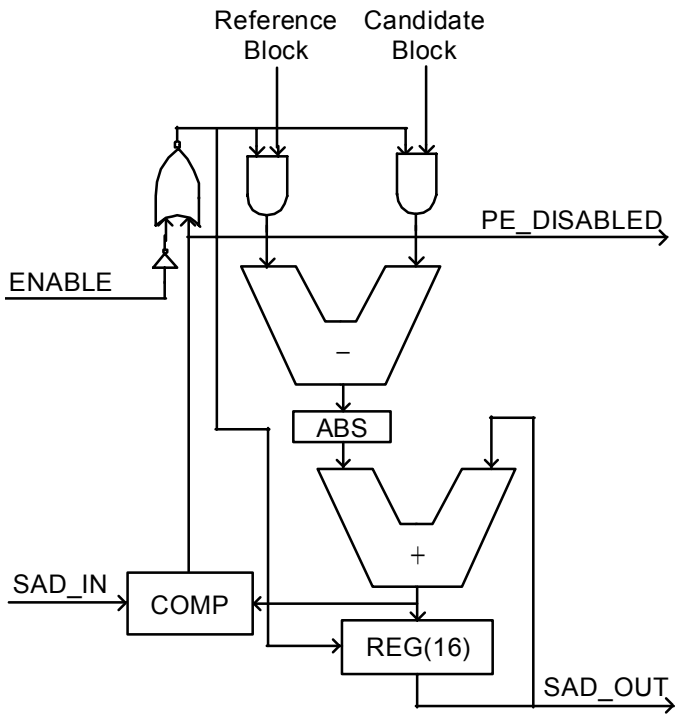

Figure 4.6 - Processing Element with Comparator to Enable SAD Stop

The timing diagram in Figure 4.7 illustrates exactly how the PE is disabled. (Note that all pixel and register values in the diagram are decimal for clarity.) For this situation, the SAD_IN value, representing the best SAD found so far by the system is 232. The PE has calculated a SAD value so far of 220, represented by SAD_OUT. Since, the SAD_OUT is less than that of SAD_IN, the COMPARATOR_OUTPUT signal is deasserted, and the PE_DISABLED signal is high. The AND gates then allow the REFERENCE_BLOCK and CANDIDATE_BLOCK pixel values to pass and be operated upon by the subtractor and absolute value unit. In conjunction with this, the 16-bit accumulator register is enabled. Notice then the second rising clock edge in the timing diagram, where the PE inputs are 135 and 142. This gives an absolute difference of 7, which when added to the previous SAD_OUT gives a new SAD of 233. The comparator flags this and enables its output. This output is *synchronous* with the clock, so the SAD_OUT clocks in the 233 value at the next rising edge. However, when the COMPARATOR_OUT goes high at the next clock, PE_DISABLED is now deasserted. This effectively "gates out" the input pixel values to zero. The absolute difference is now always zero, and the register is disabled from loading new values to help save more power. Effectively, the state of the PE is frozen, until the control unit resets the PE, then

the SAD_OUT and COMPARATOR_OUT resets to 0, re-enabling the PE for the next SAD calculation. Note that the SAD_OUT is higher than the SAD_IN, so this value can be used by the Comparator Unit during SAD evaluation states without any other logic. The logic in the PE guarantees that this SAD, even though not completely accurate, will not "win" and thus can be used as it is.

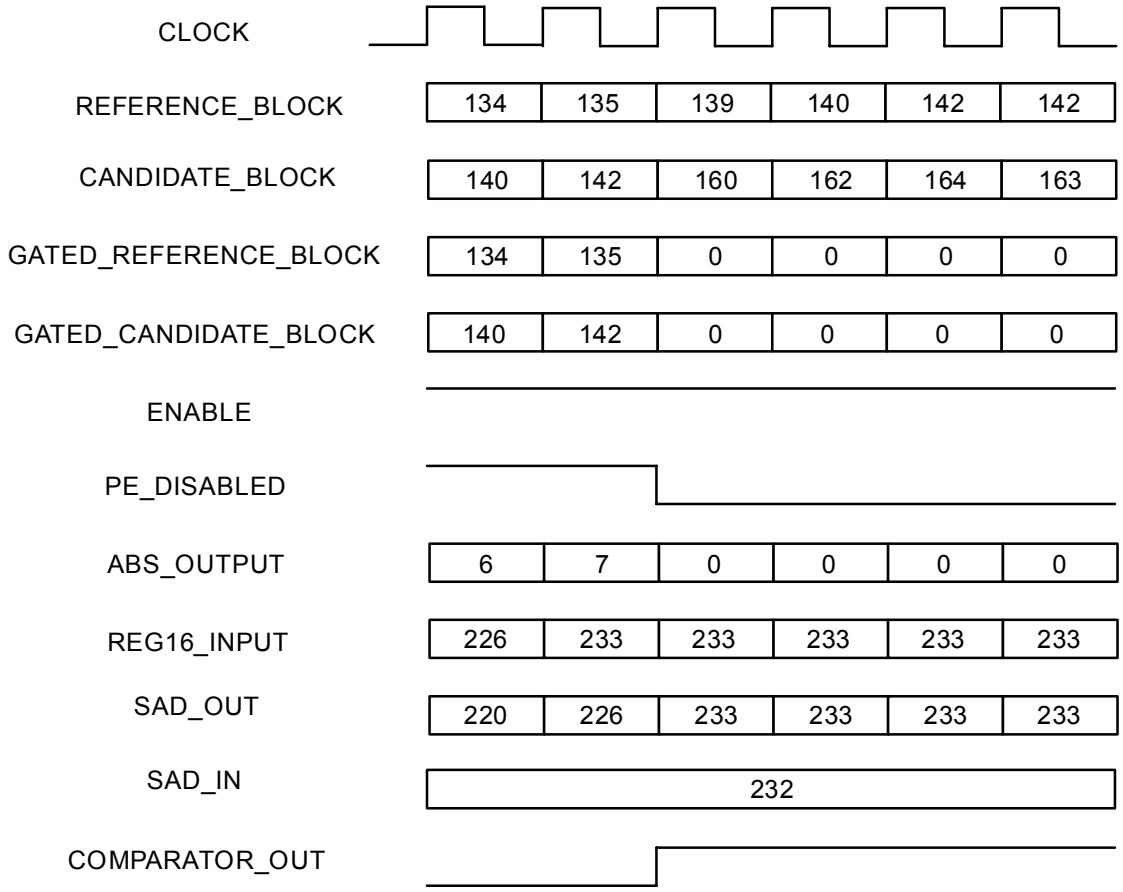

Figure 4.7 - Typical Timing Diagram for PE Comparator SAD Stop

4.3 Zero-biased Searches

The 4SS method consumes more power as more steps in the algorithm are utilized. This observation is verified by the discrepancy in power consumption from lower-motion video to larger-motion video seen in the baseline models. (Refer to Chapter 5.) The zero-bias method simply reduces the SAD rating of the center motion vector in the 1st through 3rd steps by a factor value to induce the algorithm to a faster ending, hopefully removing unnecessary steps in the algorithms while coming to an acceptable solution. Figure 4.8 shows the comparator logic with the zero-bias step added.

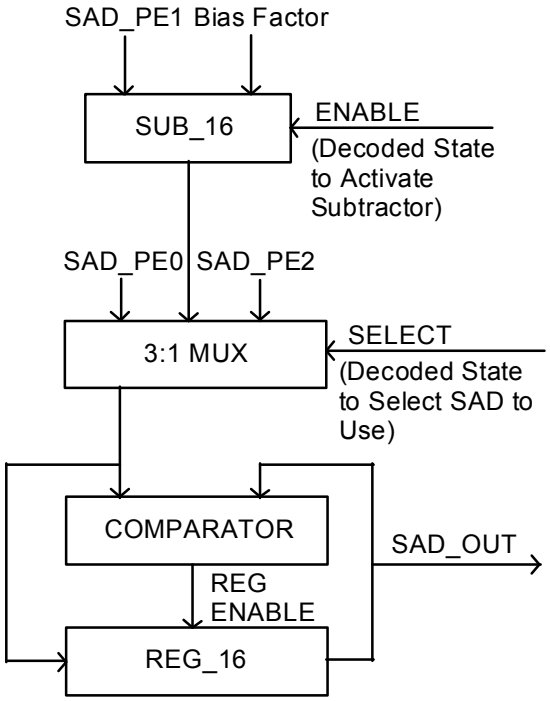

Figure 4.8 - Comparator Unit with SAD Biasing Logic

The extra subtractor in the comparator unit either subtracts the bias factor from the SAD_PE1 or enables the SAD_PE1 to pass without alteration. (In other words, it functions as a multiplexer and a subtractor.) The ENABLE line is decoded from STATE and STEP (search box row) information that determines if the center PE of the search box is active at that time according to the following equation: (STATE = 4 and STEP = 1)

$$ENABLE = STATE2 \cdot \overline{STATE1} \cdot \overline{STATE0} \cdot \overline{STEP1} \cdot STEP0 \quad (2)$$

A final consideration has to deal with not using the SAD of a PE that has been prematurely cut-off due to the comparator SAD stop method mentioned in the previous section. This can artificially reduce a SAD rating of a center PE to be much lower than the correct SAD rating minus the intended biasing factor. This is accomplished by simply gating a signal to the center PE1 during the middle row of the search box (STEP = 1) to disable the comparator SAD-stop logic. This prevents gross errors such as the one mentioned above from being introduced by the biasing logic.

4.4 Reducing Length of SADs

Revisiting the analysis of the test videos from Section 2.4 reveals that the SAD values from typical runs of the 4SS algorithm shows that for most data, the best SADs never exceed 12-13 bits and are often of much smaller bit-widths.

Using this information, it becomes apparent that perhaps not all bits in the SAD (16 in the baseline model) are necessary for computing accurate motion vectors given the types of video quality required. Large SADs also mean that the motion vector will, in most cases, not be used by the motion compensation unit anyway (i.e. the block will be encoded in INTRA mode). This information is exploited in the reduced precision model, where the length of the adder and the registers to hold the SAD in each PE (and the associated comparators) are reduced in size.

To handle cases where the SAD will "overflow" the register lengths an overflow detection scheme was devised. This scheme will detect an overflow condition by simply comparing the MSB of the previous sum and the newly computed sum. If the previous sum has an MSB of '1' and the current sum has an MSB of '0' then an overflow condition has occurred. For example, for a 14-bit SAD with the MSB of the accumulator being named REG(13) and the MSB of the adder being named REG(13), the overflow condition would then be described by the equation:

$$OVERFLOW = \overline{ADD(13)} \cdot REG(13) \tag{3}$$

A flip-flop is triggered to flag this overflow to both gate off any future inputs and signal the controller to completely disregard this SAD during output evaluation phase. The proposed PE with both comparator and reduced precision arithmetic logic is shown in Figure 4.9.

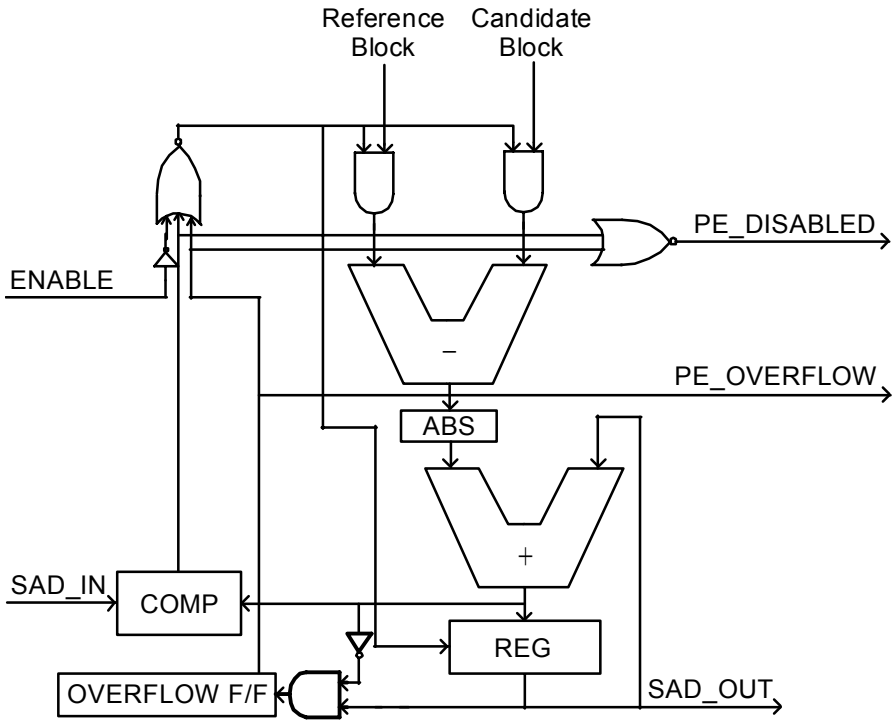

Figure 4.9 - PE with Reduced Bit-length Arithmetic and Overflow Detection Logic

Finally, the comparator unit must again be fitted with another enhancement such that the overflowed SADs are not used. The technique used is very similar to that in Figure 4.5 for not using SADs from disabled PEs. This involves gating the VALID signals with the PE_OVERFLOW signal. PE_OVERFLOW is positive logic, so the VALID signal should be deasserted when PE_OVERFLOW is asserted. The equation for the input line becomes:

$$PE\_MUX\_ENABLE_n = PE\_VALID_n \cdot PE\_DISABLE_n \cdot \overline{PE\_OVERFLOW_n} \quad (4)$$

The final logic for the comparator unit including the consideration for overflowed PEs is shown in Figure 4.10 below.

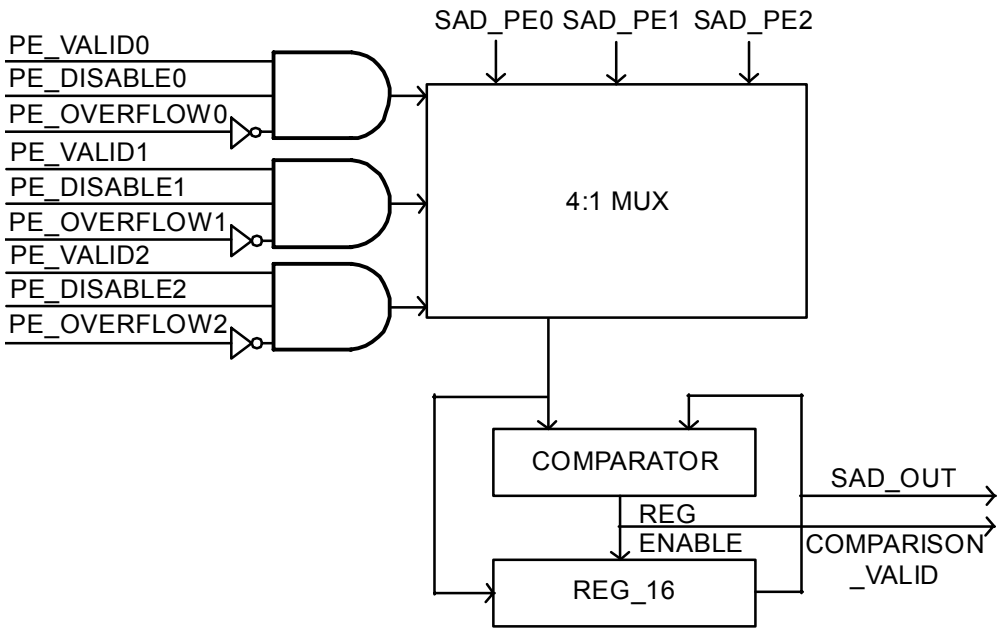

Figure 4.10 - Complete Comparator Unit Including Logic to Disregard Overflowed PEs

4.5 Miscellaneous Techniques

These power-savings techniques are based upon observations made during model development and attempt to reduce computations. They have been implemented with the "Combined" model that combines all of the power-savings techniques described in this chapter into a single model proposed in this thesis as most power-efficient.

4.5.1 Disabling VDUs and Memory

During model simulations the observation was made that all PE's were disabled quite often. During this condition, all PEs have effectively shut off pixel inputs and stopped calculations until evaluation and reset during the Decision State, or State 6. It makes sense then that the VDUs and the Memory units do not have to supply any more pixel values to the PEs, since they will be ignored anyway. A small amount of extra logic can disable memory and VDUs. Adding the simple gates increases power consumption slightly, but should result in an overall savings. This is because the gated signals disable larger blocks of logic which consume more power than the gates to generate the control signals. The net result should be a savings in power consumption for the overall circuit.

The memory access enable logic can disable the memories when needed. This occurs when all the PEs are disabled during SAD calculation. Since this condition does NOT stop the counters, any accesses to memory are unnecessary and should be disabled. To decide whether a PE is shut-off for the duration of a SAD calculation, both the PE_DISABLE from the redundant search logic (Figure 4.4) and the PE_DISABLED signal derived from the comparator SAD-stop and overflow logic (Figure 4.9) are needed. If either of those signals is deasserted (recall they are "negative" logic in relation to their names), then the PE can safely be assumed to be disabled. In relation to the memory enable signal, if any of the PEs are "alive", then the memory must be enabled. This is defined by the "OR" gate in Figure 4.11. Finally, a master signal MEMORY_ENABLE from the system enables memory when the circuit is enabled. This disables memory during idle cycles in the video encoding system.

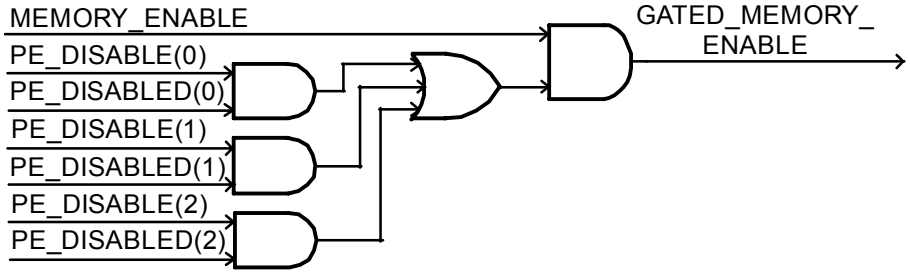

Figure 4.11 - Memory Access Disable Logic

Since there are two VDUs to be disabled, the VDU disabling logic differs slightly between them. (Refer to Figure 3.1 for better understanding.) Recall there are two sets of VDUs, one between PE0 and PE1, and another between PE1 and PE2. Recall also that pixel flow occurs from PE 0 to PE1 to PE2 through the VDUs. The condition, then, for disabling a VDU must be that all the PEs "downstream" from that VDU must be *disabled*. So to disable VDU0, both PE1 and PE2 must be disabled. To disable VDU1, only PE2 need be disabled. Figure 4.12 details the logic for each of these blocks. (Recall a $PE\_DISABLED_n$ signal is '1' when PE_n is enabled.)

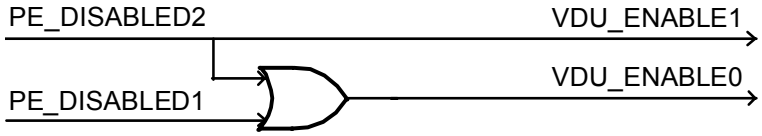

Figure 4.12 - VDU Disable Logic

4.5.2 Re-encoded Finite State Machine

Another possible power optimization includes optimizing the encoding for the finite state machine. The optimization should reduce the transitions among the state bits as much as possible to reduce dynamic power consumption. To optimize, the state transitions and their probability to occur during normal operation should be determined. Figure 4.13 shows a simple state diagram of the 4SS model. The percentages shown in the figure reflect the probability of that path being taken when the circuit *leaves that state*. Therefore, transitions back to 0 state, or reset, and transitions where the state is maintained, during Count state (1), are not considered in the diagram. The only states that have a choice as to which state will be next are Counter Reset state (2) and Decision state (6). The percentages from the Decision state rely on the number of steps taken in the 4SS algorithm. The statistic given in the figure for the 6-to-1 transition of 92% pertains to the case where the algorithm uses all four steps. In case of the algorithm using three steps, the 6-to-1 path would be taken 89% of the time and the 6-to-7 path 11%. The statistics reduce to 83% and 17%, respectively. In any case, the 6-to-1 path dominates.

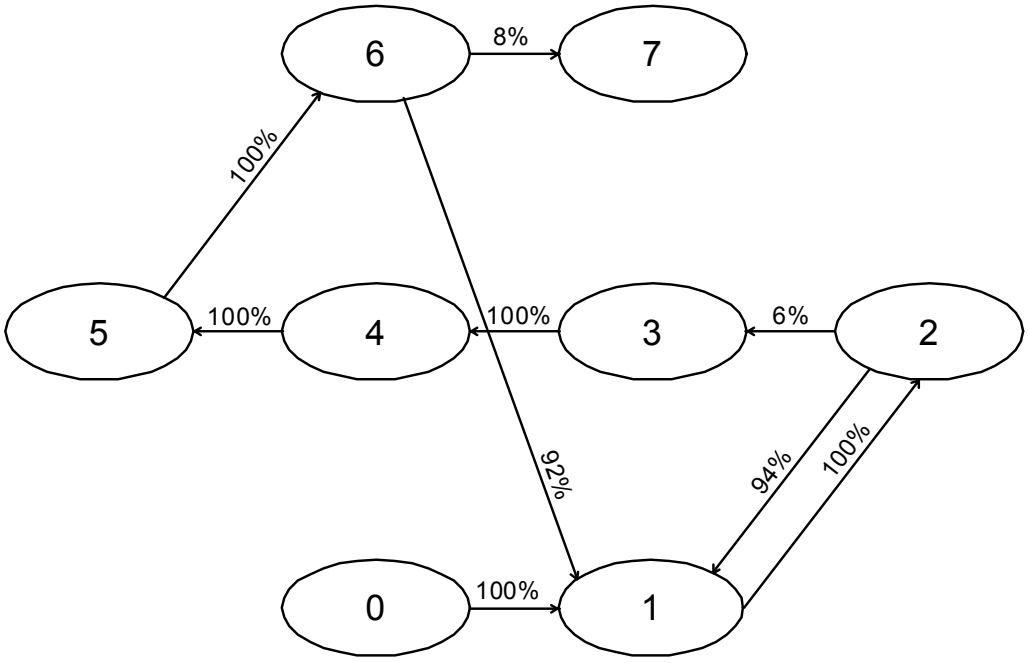

Figure 4.13 - State Machine Transitions

The goal of the optimization is to encode transitions with high probability with a smaller number of bit changes, possibly one bit change. It turns out that a Gray encoding for the states fills this need. Table 4.2 shows the new encoding.

Table 4.2 - Re-encoded States for Power Optimization of 4SS Finite State Machine

State	Old Encoding	New Encoding
Reset (0)	000	000
Count (1)	001	001
Counter Reset (2)	010	011
Evaluate PE0 (3)	011	010
Evaluate PE1 (4)	100	110
Evaluate PE2 (5)	101	111
Decision (6)	110	101
Finished (7)	111	100

4.5.3 Combinational Logic Block Disabling

Logic should be designed carefully to reduce glitches or spurious inputs, as they dissipate unnecessary power. One example of a block of logic that violates this principle is the logic that translates the BASEX and BASEY values (detailing the center of the search box in unsigned absolute coordinates) to MVX and MVY values understandable by the outside world (signed 2's complement 4-bit numbers). BASEX and BASEY change often during the motion estimation operation; forcing changes to MVX and MVY. However, the MVX and MVY values are not deemed valid by the system until FINISHED is asserted at the end of the motion estimation calculation. This can lead to unneeded switching inside the combinational logic that derives the MVX and MVY outputs. Figure 4.14 shows a gating of the inputs to the motion vector logic based upon the STATE of the system.

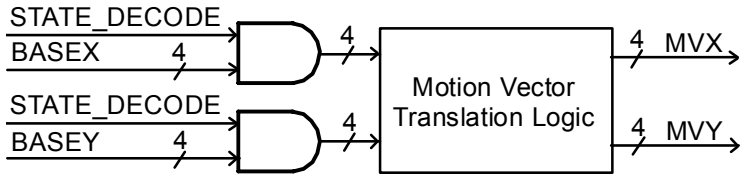

Figure 4.14 - Gated Motion Vector Translation Logic

4.6 Review of Low-Power Techniques

The major low-power techniques described in this chapter aim to reduce power consumption through reducing the computational load in the 4SS algorithm. The Redundant Search Removal and Comparator SAD Stop techniques removed calculations that are part of the regular hardware implementation of the regular 4SS algorithm, but unnecessary in reaching the final solution. The former method works best on removing these unnecessary calculations in large-motion video while the latter works best on removing those typically found in low-motion video motion estimation. The next two methods attempted to reduce the computational load by slightly altering the 4SS algorithm to reduce computational complexity without greatly disturbing the results. The Zero-bias Method aims to bias the algorithm to select the center vector in steps 1-3 to force a faster end to motion vector computation, reducing the overall number of points that must be checked. The other proposed idea reduces the number of bits that can be represented by a SAD. If a SAD calculation overflows the reduced bitlength, it is discarded and not used. Computations are thus eliminated on candidate vectors that are of too large a SAD.

Other methods in this chapter reduce transition activity in circuit elements whenever possible. Logic was presented to disable the local memories and VDUs. The state machine was reencoded so that all normal state transitions involve only a single bit. Finally, combinational logic blocks (e.g. PEs, scaling logic in the controller, motion vector decoders, etc.) are gated at the inputs to remove unnecessary signal transitions in those logic blocks when the outputs are not needed. This also reduces power due to glitches and spurious inputs.

Chapter 5
Experimental Results

In this chapter we present experimental results of the low-power design methods described in Chapter 4, both in terms of video quality and power characterization. Both power characterizations of the circuit logic estimated from gate-level simulations and the power consumed by memory accesses are considered in the results. The synthesized circuit area and measured performance are also presented.

To measure video quality, an H.263 software encoder/decoder system was developed. The motion estimation algorithms are implemented and measured for peak signal-to-noise ratio (PSNR) as a quantitative measure of video quality. Additionally the number of blocks transmitted in INTRA or INTER mode and length of encoded bitstreams for the test videos were measured to validate the proposed motion estimation design as a real-world encoder. Power dissipation was estimated for baseline models, a model incorporating one low-power design technique and the model incorporating all the low-power design techniques described in the previous chapter.

The rest of the chapter proceeds as follows. First, the general design flow of model generation, synthesis, and simulation are presented. A discussion of the power-characterization techniques is presented next. Then, the verification of the functionality of the designs is discussed. Experimental results on area and performance, video quality information, and power characterizations, and analysis on the results are presented finally.

5.1 Design Flow and Power Characterization Procedures

This section highlights the design flow used in the development of hardware models of the motion estimation designs described earlier. The flow is summarized in Figure 5.1. The basic design flow generates a synthesized model suitable for gate-level simulation derived from the Virginia Tech VLSI for Telecommunications (VTVT) 0.35um TSMC CMOS library. The library operates at a voltage of 3.3V and includes capacitance and power characterization information necessary to perform gate-level power characterization using Synopsys Power Compiler.

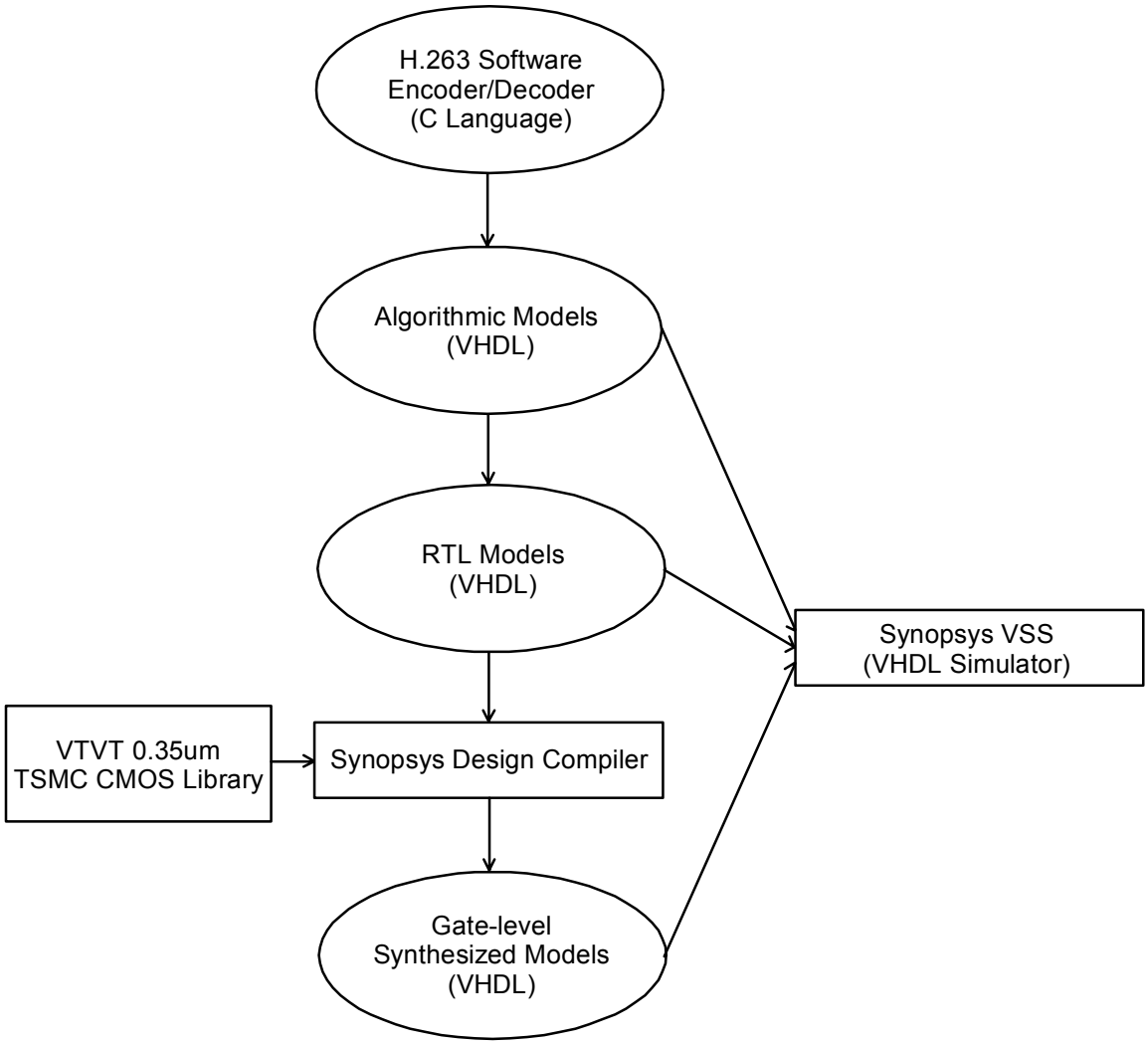

Figure 5.1 - Design Flow for Model Development

High-level development is a fully functional prototype of a complete H.263 encoder and decoder system written in C. There are a variety of uses for this model in developing the hardware.

- Validation of motion estimation algorithms on measured video quality and perceived video quality from actually viewing encoded video.
- Examining algorithmic enhancements for their impact on functional performance quickly.
- Establishing a baseline for verification of the hardware motion estimation models.

These advantages are realized by simply rewriting the motion estimation algorithm inside the prototype encoder. The video quality can be observed and the results collected for experiments.

The next models developed are algorithmic models in VHDL. They serve mostly to verify correctness of the later RTL and synthesized models. This procedure will be described more in detail in the next section. The RTL models are the heart of the design. VHDL code was developed for the intentions of being verified for correct functional performance and synthesis into a gate-level design. Synopsys Design Compiler performed the actual hardware synthesis using the aforementioned 0.35 um library. A synthesized model for gate-level simulation is produced in both VHDL files (for simulation) and Synopsys database format (for later power characterization).

The power characterization procedures follow the general flow presented in Figure 5.2. The RTL model is synthesized, and VHDL and Synopsys database format files are produced. The VHDL file can be analyzed and simulated using the same test benches developed for the RTL model verification. The VSS simulator, with the proper options set, will produce a toggle file, that lists the toggle frequency of signal lines within the gate-level simulation. The toggle file can be translated to a script file that is readable in Design Compiler. Then, Design Compiler reads in the Synopsys database file on the synthesized design, executes the toggle script, and reports the power consumed by the circuit.

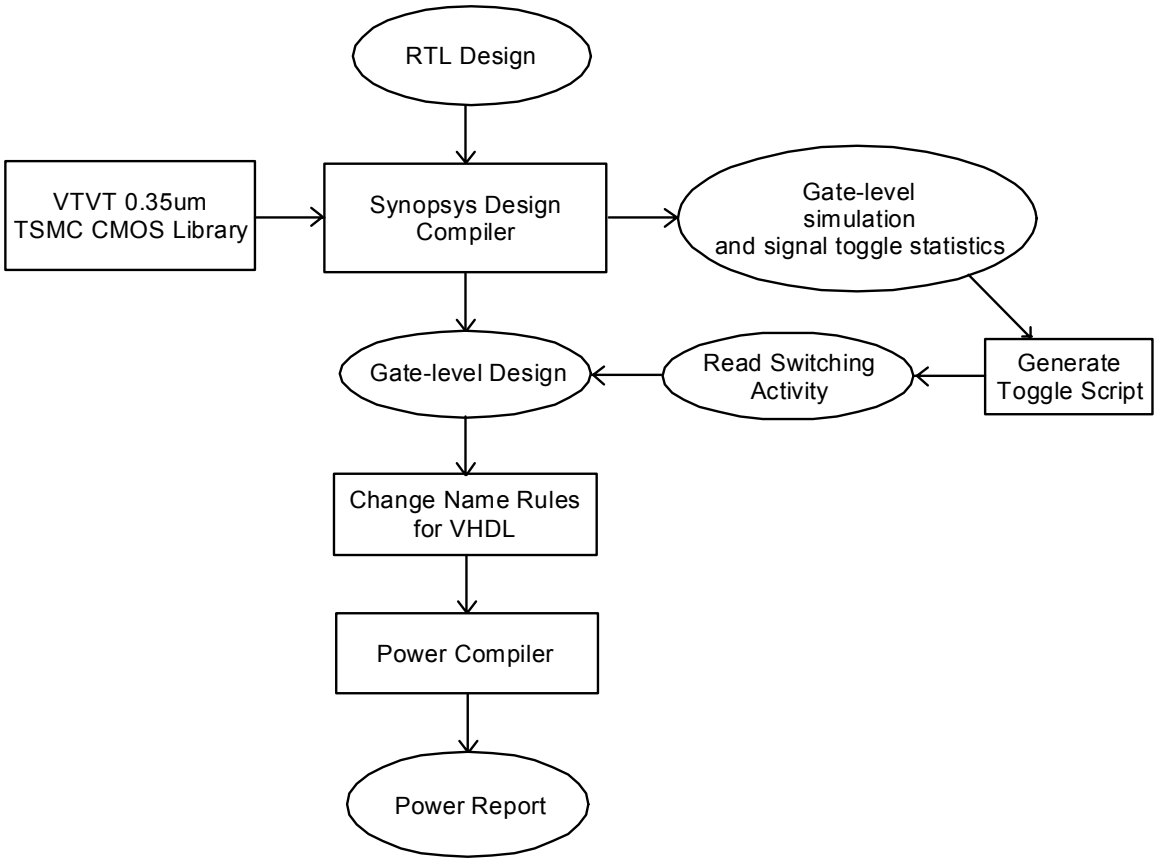

Figure 5.2 - Power Characterization Flow using Synopsys Tools

Memory power is characterized by calculating the number of accesses made by the model during operation. These access numbers can be obtained from the baseline models by studying operation and extracting the number of steps used by the implemented algorithm in each motion estimation operation. Scripts were developed to read this from the RTL simulation outputs. As part of the low-power enhancements, logic was added to detect periods when the memory could be disabled under certain conditions. Additional hooks, or extra VHDL behavioral code, were added to the RTL models to detect the number of clock periods these memory-disable conditions existed. Therefore, during RTL simulation, in addition to verification information, the number of memory accesses was extracted.

5.2 Design Verification

The RTL model was verified for functional correctness before synthesis and power characterization. Obviously, this is important to validate any power findings. The flow of verification is presented in Figure 5.3.

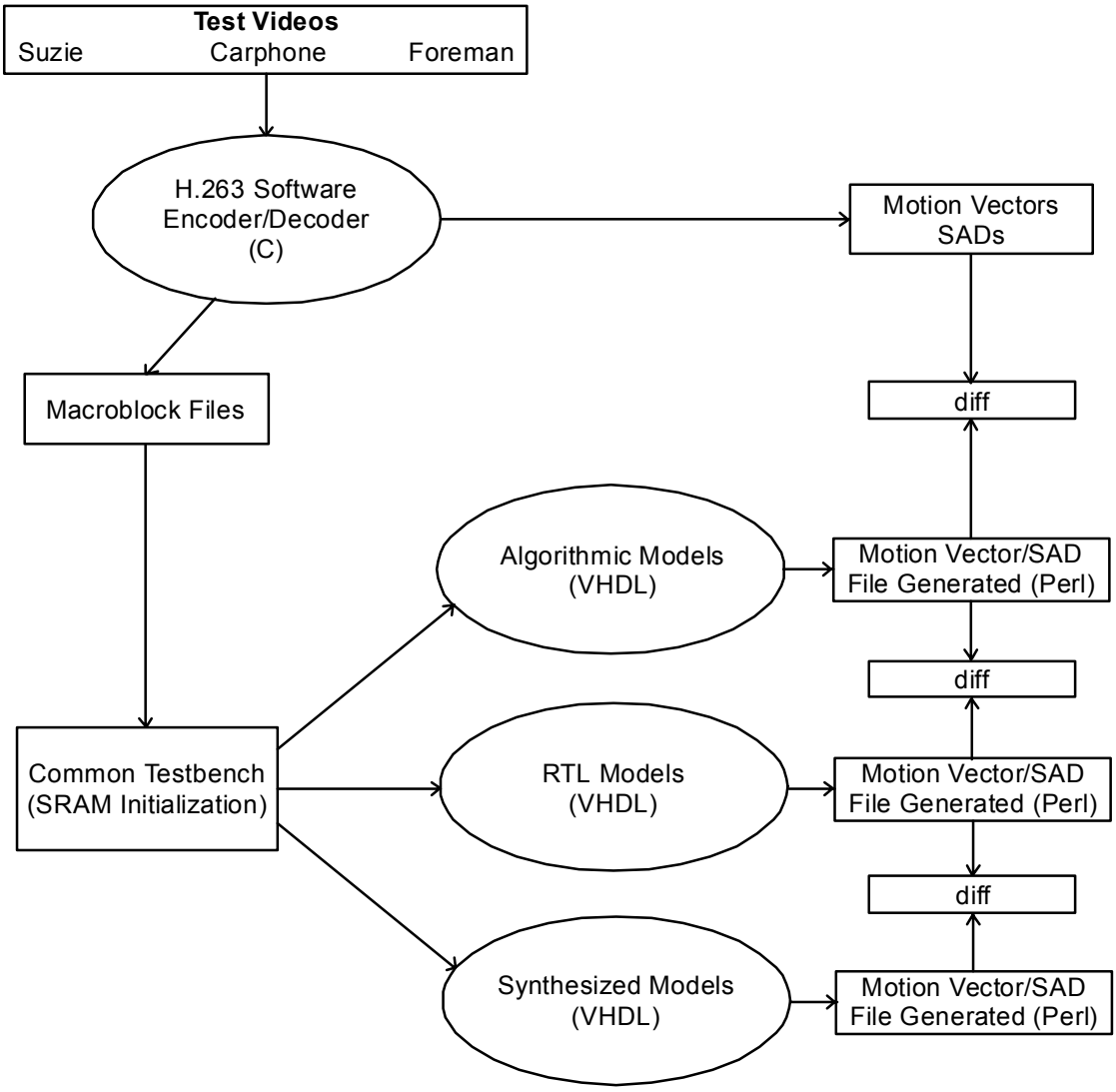

Figure 5.3 - Design Verification Flow

As mentioned previously, the prototype H.263 software encoder/decoder is vital to RTL model development and verification. The prototype is used as the starting point in the verification flow. Motion estimation algorithms were first written in C and plugged into the prototype. Extra subroutines extracted the pixel values from the reference macroblocks and the search areas. These subroutines also extracted the motion vectors and SAD ratings found with the software. The software itself was manually verified for correctness.

The pixel information and motion vector files were then used as the stimulus for the behavioral RAM model. This model is common to all the motion estimation models and model-types (algorithmic, RTL, and synthesized). The RAM model is responsible

for initializing the RAMs and delivering data to the motion estimation model depending on the addresses generated. The first model developed and verified was an algorithmic model using VHDL in behavioral context as a pure programming language. This model verified the VHDL algorithm, the behavioral RAM model and its initialization routine, and the associated support scripts. These scripts processed performance statistics and automatically verified functional correctness. Verification of the RTL and synthesized models simply became a matter of plugging in the correct models into the RAM initialization routine and test bench and checking ("diff"ing in Figure 5.3) the output file against that produced by the algorithmic model. This process was automated by using Perl scripts to automatically parse the macroblock input file, run the VSS simulator, collect motion vector data from the output, and prepare the output file. Using the same test benches for every model facilitated one set of scripts to perform design verification (and power characterization and memory access statistical collection) automatically. Over 13 frames worth of representative video were used to validate operation for each of the three test videos used in the research. That translates to over 1300 macroblocks used for model verification for each of the videos.

5.3 System Performance

This section details some performance parameters from the synthesized models. These include area and timing reports. Comparisons are also made between the models for area and speed tradeoffs.

5.3.1 Area

The relative area of each of the models was measured using Synopsys Design Compiler. Area measurements included are an approximation based upon the cells used from the 0.35um VTVT library, since the design exists only as a gate-level design. The Design Compiler tool reports the number of equivalent NAND2 cells used by the cells in a synthesized design. The major models report the following areas as shown in Figure 5.4.

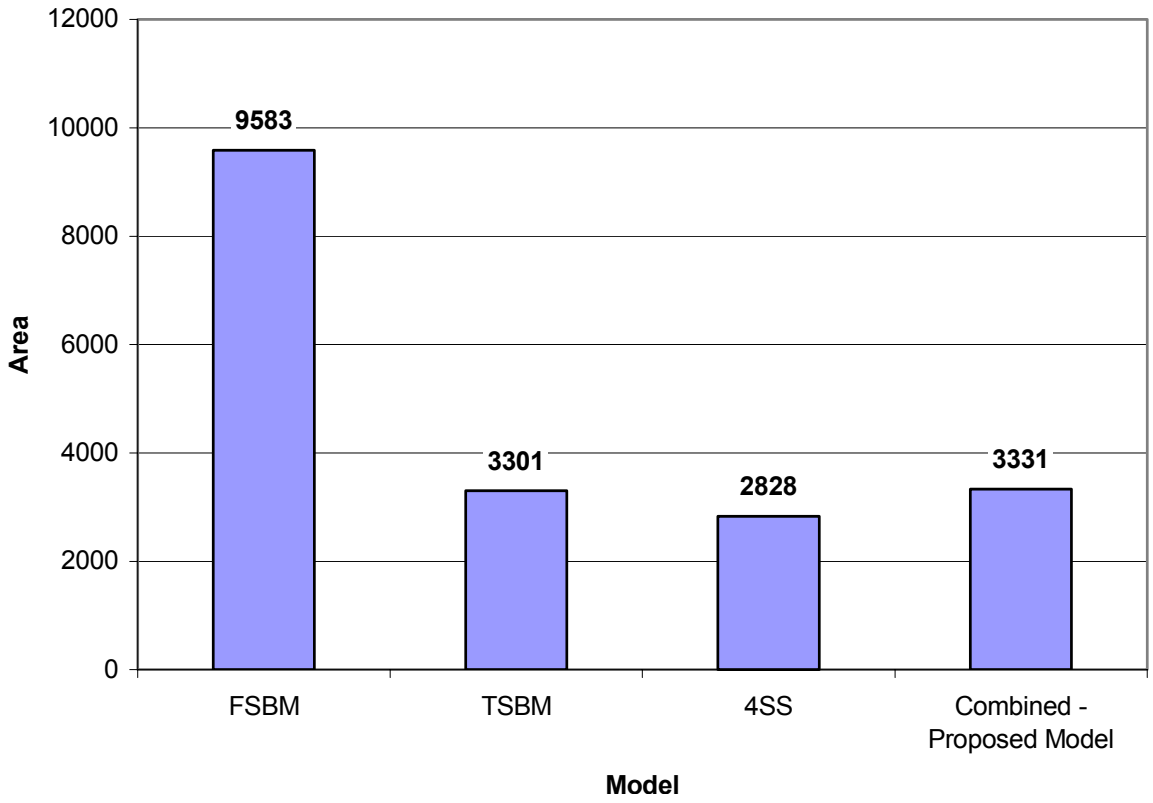

Figure 5.4 - Areas of the Baseline Models

As expected the FSBM model sacrifices area, almost 3 times that of the other models to achieve the superior throughput necessary to implement the FSBM algorithm. In short, trading accuracy for area occurs (and power as will be seen later). The other three models are closer in size. The 4SS requires less hardware than the TSS model, mostly due to the fact that the VDUs shrink in size. The Combined model requires extra logic for power savings features and is larger than the 4SS model upon which it is based. The following table further breaks down the area relationship according to major components in the TSS, 4SS and Proposed models.

Table 5.1 - Area of Major Components of Models (Equivalent NAND2 Gates)

	TSS	4SS	Combined
PEs	1335 (40%)	1335 (47%)	1644 (49%)
VDUs	676 (20%)	306 (11%)	340 (10%)
Controller	1199 (36%)	1148 (41%)	1293 (39%)
Total	3301	2828	3331

The area of the Combined model increases 18% compared to the 4SS model. Most of this increase is due to the increased size of the processing elements to nearly half of the entire model. The VDUs show a moderate increase to allow for the logic to disable them described in Chapter 4. The Controller unit also shows an increase of nearly 13% in size over the 4SS model. This increase is due to mostly the redundant search removal logic and the alterations to the comparator.

The models with a single power-savings method enabled showed slight increases in circuit area, ranging from 2921 to 3122. The reduced range arithmetic models reduced circuit area as expected down to 2668 for the 12-bit model. This is another advantage for including the reduced range arithmetic method in the Combined model-it can offset the size increase by including other power-savings circuitry to result in a circuit approximately the same size as the TSS model.

5.3.2 Performance

Synopsys Design Compiler supports generation of timing reports for the synthesized models. This information is used to predict maximum throughput performance based upon timing reports. When using this information a consideration must be made for the RAM used in the models. It is modeled behaviorally in the simulations and thus access time for the RAMs must be taken into account for a timing path. There are two figures in the table below for timing. One is the delay reported by Synopsys based upon information from the actual cell library. The other is the access time that adds to the delay seen on the path. These two statistics are summed to generate the total path delay for each model to predict maximum performance. For the FSBM model, the worst-case access comes from the 512-byte memory and is 17.5 ns [52]. The other models use the 1kB memory and its access time is taken as 21.8 ns [52].

Table 5.2 - Timing and Performance Results for Major Models

	Delay Path (ns)	Worst-case Memory Access (ns)	Total (ns)	Maximum Clock Frequency (MHz)	Maximum Throughput for QCIF (fps)
FSBM	21.3	17.5	38.8	25.8	68
TSS	10.7	21.8	32.5	30.8	98
4SS	10.8	21.8	32.6	30.7	77
Combined - Proposed Model	15.1	21.8	36.9	27.1	68

From the results in the table, the minimum clocking requirements for 15 frames per second video at QCIF are met easily by the timing for the models. In fact, there is room to quadruple the frame rate if so desired in the 4SS and combined models. Notice also, that memory access time dominates the performance of all the models. Finally, the extra logic present in the combined models does reduce possible performance, but not by an unsatisfactory margin (12%). The minimum clock speed of 5.9 MHz presented in Chapter 3 satisfies timing requirements for the Combined model.

5.4 Video Results

While saving power in the hardware implementation of a motion estimator is the primary focus of this research, care must also be taken to ensure that the video quality is suitable for inclusion in a real-world design. This is where the prototype is extremely valuable. Secondly, an important system consideration must be looked at for any motion estimation block used in an H.263 system. That is, how many blocks can the algorithms find matches for that are deemed suitable for motion compensation. This consideration and the numerical measure of picture quality are looked at in this section.

5.4.1 Macroblock Mode Selections

Recall from Chapter 2 that every block can be encoded either INTER or INTRA. INTRA implies that the SAD rating is too high and the macroblock is transformed, quantized and sent across the channel without motion estimation. INTER mode implies that a suitable match was found and the motion compensation unit is used and the block difference (from the previous best-match block) and the motion vector are sent across the

channel. The following table (Table 5.3) summarizes the results of INTER or INTRA mode selection for all models. Note that "factor" for the zero-bias model indicates the SAD biasing factor used for the center vector in the first step of the 4SS algorithm.

Table 5.3 - Mode Selection for Macroblocks in Test Videos

	Suzie			**Carphone**			**Foreman**		
	INTER	*INTRA*	*%*	*INTER*	*INTRA*	*%*	*INTER*	*INTRA*	*%*
Full-Search	14713	38	99.7	37261	458	98.8	36691	434	98.8
TSS	14713	38	99.7	37200	519	98.6	36609	516	98.6
4SS	14713	38	99.7	37156	563	98.5	36505	620	98.3
Zero-bias (Factor of 100)	14709	42	99.7	37171	548	98.5	36509	616	98.3
14-bit Arithmetic	14713	38	99.7	37156	563	98.5	36505	620	98.3
13-bit Arithmetic	14708	43	99.7	37170	549	98.5	36462	663	98.2
12-bit Arithmetic	14707	44	99.7	37447	272	99.3	36645	480	98.7
Combined	14710	41	99.7	37444	275	99.3	36626	499	98.7

The results from this table reflect positively on the algorithmic changes made to the models in attempts to reduce power consumption. For the Suzie video, the algorithmic enhancements make virtually no change in the number of blocks sent in INTER mode. In Carphone and Foreman, the number of INTER blocks degrades somewhat from the FS algorithm due to the larger motion in the videos. However, the number of blocks encoded INTER does not change significantly from the 4SS to the single-method power savings models. Some significant "improvements" seem to be made with the 12-bit Arithmetic and Combined models. The improvement, however, results from the algorithm using the largest 12-bit number possible (4091) as the SAD when a lower SAD cannot be found in the first two steps.

These results confirm that the compression techniques of H.263 are still usable and the motion estimator still successful with the power-savings enhancements. A more complete picture of the answer can be gained by including the information from the next table (Table 5.4). This table gives the actual number of bits used by the encoded bitstream of the entire video under the different motion estimation models.

Table 5.4 - Encoded Bit-length of Test Videos Under Motion Estimation Models

	Suzie		Carphone		Foreman	
	Number of Bits	*% Increase from FS*	*Number of Bits*	*% Increase from FS*	*Number of Bits*	*% Increase from FS*
Full-Search	391,584	0.0	889,968	0.0	945,088	0.0
TSS	388,480	-0.8	888,616	-0.2	945,056	0.0
4SS	389,816	-0.5	890,568	0.1	955,576	1.1
Zero-bias (Factor of 100)	395,168	0.9	892,728	0.3	956,704	1.2
14-bit Arithmetic	387,264	-1.1	890,984	0.1	955,776	1.1
13-bit Arithmetic	391,376	0.6	889,472	-0.6	960,200	1.6
12-bit Arithmetic	394,088	0.6	893,504	0.4	1,015,496	7.4
Combined	399,120	1.9	895,544	0.6	1,010,928	7.0

Again, for the Suzie and Carphone video sequences, the compression algorithm in the encoder works well with the motion estimation block. The increase in the number of bits transmitted is small, on the order of a 1-2% increase. Foreman, however, shows similar results until the 12-bit Arithmetic model and the Combined models are considered. The increase in bits transmitted is more significant to 7%. This is a primary concern since the transmitter now must consume more power and the video encoder unit must transmit more data. The largest increase in the bits transmitted occurs during the large camera pan in the Foreman video, where motion increases drastically throughout the entire picture. Despite the fact that more blocks are encoded INTER than in a typical Full-Search model, those matches are not ideal and the motion compensation unit ends up using more transmitted bits than if a better match or an INTRA block were transmitted. This illustrates a trade-off made by targeting the system to reduce power consumption for low-motion, low-SAD video.

5.4.2 Video Quality Measurements

Recall that the motion estimator and motion compensation units are not the only units to be considered for video compression in an H.263 system. There is still the transform block and more importantly the quantization factor. The quantization factor is

the factor by which transformed coefficients of a macroblock are divided to implement a uniform compression (truncation) of least-significant bits. It is manipulated by the rate control of the system to increase or decrease quantization if the bit-rate target is not being maintained. A large increase in the number of bits may not be seen for the targeted low-motion video. The H.263 algorithm quantizes the bitstream by a very large factor to reduce the size of the bitstream and effectively trades off picture quality. To ensure a *complete* answer to the question of the effectiveness of the new motion estimation block, picture quality was measured for all videos and all models.

To review, the PSNR (picture-signal to noise ratio) measurement was used to quantify picture quality. The equation was given in Section 2.4. Table 5.5 shows the experimental results.

Table 5.5 - PSNR of Video Under Motion Estimation Models (dB)

	Suzie	*Carphone*	*Foreman*
Full-Search	32.575	28.780	27.722
TSS	32.476	28.618	27.199
4SS	32.458	28.744	27.346
Zero-bias (Factor of 100)	32.522	28.913	27.371
14-bit Arithmetic	32.458	28.744	27.346
13-bit Arithmetic	32.480	28.758	27.360
12-bit Arithmetic	32.420	28.662	27.128
Combined	*32.451*	*28.866*	*27.213*

These results again are favorable for the low-motion videos, Suzie and Carphone. There is only a slight drop in PSNR for the Combined model in Suzie compared to Full-Search and hardly no drop whatsoever from 4SS. Carphone actually shows a very slight *increase* in PSNR. Foreman shows a 0.5 dB drop in PSNR from the Full-Search model. Again, this illustrates the trade-off over using the motion estimator for video with larger motions. The power consumption will increase for the transmitter and there will be a drop in picture quality. However, a good point for the Combined model should be made that there is little picture quality loss from the 4SS model and identical performance to the TSS model.

5.5 Power Dissipation

5.5.1 Gate-level Power Consumption

As mentioned above, the synthesized models were simulated for power characterization using Synopsys Design Compiler and Power Compiler. The results are presented in this section.

The first section looks at the baseline models. The results for the power consumption of these models are shown in Table 5.6. The table includes the power consumed by some of the major blocks of the circuit for later comparison. Notice that the numbers do not completely add up. There is other miscellaneous logic in the models that is not broken down in the table completely. The three major power-consuming portions of the circuits were included to facilitate valid comparisons between the power consumption of the models.

Table 5.6 - Circuit Power Consumption (mW) for Baseline Models

Baseline Model		*Suzie*	*Carphone*	*Foreman*
FSBM	PEs	23.25	22.69	21.15
	Controller	1.45	1.44	1.44
	Total	**25.93**	**25.33**	**23.88**
TSS	PEs	3.445	3.252	3.009
	VDUs	0.987	0.967	0.966
	Controller	0.785	0.786	0.785
	Total	**6.633**	**6.428**	**6.202**
4SS	PEs	2.335	2.231	3.394
	VDUs	0.396	0.387	0.656
	Controller	0.506	0.510	0.801
	Total	**3.858**	**3.758**	**5.853**

As expected the TSS and the 4SS models dissipate much less power than the FSBM model. The table illustrates that power dissipation for the 4SS model depends greatly on the video processed. The 4SS models have the advantage of dissipating over 40% less power than the TSS model for low-to-medium-motion video. Large motion video dissipates more power in the PEs than the TSS model, but again the smaller VDUs account for an overall slight decrease in power consumption.

The power consumption was also characterized for models with a single enhancement added as well as the final model that combines all presented methods. Those results are shown in Table 5.7.

Table 5.7 - Power Consumption of Enhanced Models

Model		*Suzie*	*Carphone*	*Foreman*
Redundant Search Removal	PEs	1.856	1.862	2.321
	VDUs	0.339	0.343	0.544
	Controller	0.550	0.556	0.863
	Total	**3.467**	**3.491**	**4.875**
SAD Half-Stop	PEs	1.947	1.656	3.114
	VDUs	0.375	0.367	0.624
	Controller	0.504	0.508	0.798
	Total	**3.520**	**3.232**	**5.638**
Zero-bias	PEs	2.137	2.065	3.165
	VDUs	0.354	0.350	0.604
	Controller	0.483	0.493	0.787
	Total	**3.523**	**3.472**	**5.466**
14-bit Arithmetic	PEs	2.225	2.120	3.226
	VDUs	0.375	0.367	0.624
	Controller	0.506	0.510	0.801
	Total	**3.765**	**3.664**	**5.702**
13-bit Arithmetic	PEs	2.269	2.161	3.232
	VDUs	0.396	0.387	0.644
	Controller	0.497	0.501	0.775
	Total	**3.744**	**3.639**	**5.579**
12-bit Arithmetic	PEs	2.091	1.854	2.471
	VDUs	0.372	0.360	0.523
	Controller	0.485	0.485	0.671
	Total	**3.565**	**3.320**	**4.525**
Combined - Proposed Model	PEs	1.399	1.213	1.690
	VDUs	0.310	0.324	0.429
	Controller	0.479	0.524	0.670
	Total	**2.786**	**2.714**	**3.634**

As expected, the redundant search removal achieves the largest power saving for the larger-motion video, while the SAD Half-stop method worked best on the video with less motion. The zero-bias method resulted in an average power savings. The reduced arithmetic SAD model did not reduce power significantly until the length of the SAD was reduced to 12 bits. Then, a large amount of power savings was seen in the Foreman video.

The final results of the circuit power characterizations are summarized in Table 5.8. Included are the power savings compared with the baseline FSBM, TSS and 4SS models.

Table 5.8 - Logic Power Dissipation and Power Savings for Major Models

	Suzie	*Carphone*	*Foreman*
FSBM	25.93	25.33	23.88
TSS	6.633	6.428	6.202
4SS	3.858	3.758	5.853
Combined - Proposed Model	*2.786*	*2.575*	*3.634*
Savings (vs. FSBM)	*89.2%*	*89.8%*	*84.8%*
Savings (vs. TSS)	*58.0%*	*59.9%*	*41.4%*
Savings (vs. 4SS)	*27.8%*	*31.5%*	*37.9%*

The logic power savings are nearly 90% over a FSBM implementation. The savings is 60% for small-motion video and 40% for larger-motion video over the TSS. The power-savings enhancements help greatly reduce the power consumption of the proposed model for large-motion video. Recall that the baseline 4SS model had power consumption close to that of the baseline TSS model.

5.5.2 Memory Power Consumption

SRAMs are used to store the reference macroblock and candidate area pixel values for access by the motion estimation unit. The SRAM sizes used are 256x8, 512x8, and 1024x8. Each model uses the 256x8 RAM for its reference block pixels. The candidate areas are partitioned differently depending upon the model. The FSBM model uses two 512x8 SRAMs while the TSS and FSBM model, including the power-savings models, uses a single 1024x8 SRAM. Since these memories can consume significant power, the results of memory consumed due to accesses from the motion estimation models are included in this section. A typical low-power layout of these memories has the characteristics for delay and dynamic power consumption shown in Table 5.9 [52].

Table 5.9 - SRAM Statistics for Memories Used in Motion Estimation Models [52]

Memory	Dynamic Power (mW)	Access Time (ns)
256 B	25.86	14.7
512 B	30.00	17.5
1 kB	41.54	21.8

The first models presented are the baseline models. Calculating power due to memory accesses for the FSBM and TSS models is relatively easy, since the number of accesses is fixed for each motion estimation operation. The 4SS baseline model must be

simulated to obtain the average number of steps used in the algorithm. The results of simulation on the 4SS baseline model are presented in Table 5.10.

Table 5.10 - Average Number of Steps for 4SS Baseline Model

Video	2 Steps	3 Steps	4 Steps	Average Number of Steps
Suzie	57.7 %	33.2 %	9.0 %	2.51
Carphone	60.8 %	23.7 %	15.5 %	2.55
Foreman	2.2 %	25.7 %	72.1 %	3.70

As expected, the Foreman video with the large motion uses an average of more than an extra step of the 4SS algorithm to compute. This leads to larger memory consumption as seen in the next table, Table 5.11. The memory access statistics are either calculated or obtained from simulation are used to complete dynamic memory power statistics for the baseline models. The FSBM algorithm has the worst power consumption of all the baseline models. 4SS improves upon TSS by nearly 1 mW for low-motion videos; however, a large-motion video such as the camera pan in Foreman can drastically increase the memory power consumption, by over 2 mW in this case compared to the TSS model.

Table 5.11 - Baseline Model Memory Power Consumption (mW)

Model	**FS**	**TSS**	**4SS**		
Video	*All*	*All*	*Suzie*	*Carphone*	*Foreman*
Candidate Accesses	3840	2304	1928	1956	2838
Candidate Dynamic Power (mw)	2.91	1.75	1.46	1.49	2.16
Reference Accesses	7680	2976	2314	2349	3451
Reference Dynamic Power (mW)	6.08	4.90	4.19	4.25	6.25
Total Dynamic Memory Power Consumed (mW)	*8.99*	*6.65*	*5.65*	*5.74*	*8.41*

The low-power enhancements can help reduce dynamic memory power in two specific ways. The first one is by disabling memory during calculation when possible. Three of the power-saving methods work by shutting off the PEs whenever possible. Memory can be disabled when all 3 PEs of the 4SS model are disabled at one time. RTL simulations measured the number of skipped accesses for each power-savings enhancements. These results are given in Table 5.12.

Table 5.12 - Memory Accesses Skipped by Disabling All PEs

	Suzie	*Carphone*	*Foreman*
Redundant Searches	123	168	214
SAD Half-Stop	201	450	145
14-bit Arithmetic	0	0	0
13-bit Arithmetic	0	0	8
12-bit Arithmetic	28	102	134
Combined - Proposed Model	81	86	647

The second method by which dynamic memory power can be reduced is seen in the methods that change the 4SS algorithm, zero-biasing and reduced bit-length SADs. Namely, these algorithms can both reduce the number of steps that are required for the 4SS algorithm. This directly affects not only logic power consumption but can reduce, on average, the number of accesses to memory required per motion estimation operation. As mentioned previously, RTL simulations were used to obtain the number of steps required by the 4SS algorithm using the models with these enhancements. The results are summarized in Table 5.12.

Table 5.13 - Average Number of Steps of 4SS Algorithm Used with Power-Savings Enhancements

	Suzie	*Carphone*	*Foreman*
4SS Baseline	*2.51*	*2.55*	*3.70*
Zero-bias	2.26	2.28	3.23
14-bit Arithmetic	2.51	2.55	3.70
13-bit Arithmetic	2.50	2.55	3.68
12-bit Arithmetic	2.50	2.53	3.39
Combined - Proposed Model	2.26	2.28	3.24

Zero-biasing has the largest effect on reducing memory accesses by reducing the average number of steps needed by the 4SS algorithm. While reducing the available bits for a SAD value has the ability to change the algorithm, it is, in fact, the zero-biasing that has the largest effect, since the Combined models statistics nearly identically correlate with the single-method model using the zero-bias method.

These results were tabulated to give the following table for dynamic memory power consumption for all the major models.

Table 5.14 - Dynamic Memory Power Consumption (mW)

	Suzie	*Carphone*	*Foreman*
FS	8.99	8.99	8.99
TSS	6.65	6.65	6.65
4SS	5.65	5.74	8.41
Combined - Proposed Model	4.86	4.91	5.69
Savings (vs. FSBM)	*45.9%*	*45.4%*	*36.7%*
Savings (vs. TSS)	*26.9%*	*26.1%*	*14.4%*
Savings (vs. 4SS)	*14.0%*	*14.5%*	*32.3%*

Some amount of power reduction is seen with the low-motion videos-approximately 14% compared to 4SS baseline and 26% better than the TSS baseline model. For the large-motion Foreman video, the power reduction is much more significant-over 32%. The large reduction in memory power is even more significant with this video due to the fact that the Combined model could increase the number of bits to encode the video. A larger power savings here is needed to offset an increase in transmitter power for the entire system.

5.5.3 Total Power Consumption

A complete picture of the power consumption of these models can be gained by considering the power consumed by the local memories with the power consumed by the circuit elements. Table 5.15 shows the total power consumption statistics for each baseline model and the Combined model.

Table 5.15 - Total Power Consumption (mW)

	Suzie	*Carphone*	*Foreman*
FS	34.92	34.32	32.87
TSS	13.28	13.08	12.85
4SS	9.51	9.50	14.26
Combined - Proposed Model	7.65	7.49	9.32
Savings (vs. FSBM)	*78.1%*	*78.2%*	*71.6%*
Savings (vs. TSS)	*42.4%*	*42.7%*	*27.5%*
Savings (vs. 4SS)	*19.6%*	*21.2%*	*34.6%*

The results show that considering memory power can lower expected power savings due to the fact that local memory consumes a larger portion of power than the circuit elements. The overall power savings are still significant with low-motion video

showing a 20% decrease in overall power. 35% reductions are seen for larger-motion video. The power characterizations, which are in the 7-9 mW range are feasible for use in a mobile system.

Chapter 6
Conclusion

Compressing the information required to transmit video is the principle goal of any H.263 video encoding system. The process of video prediction compresses video by estimating motion from the current frame based upon a previous frame and compensating for that motion by transmitting only the difference in the blocks brought about by the detected motion. Motion estimation, or the process of calculating motion, becomes the most critical and computationally intensive part of the process of prediction. In this research, we investigated algorithms and implementations to design a low-power hardware block suitable to perform motion estimation for a low bit-rate, low-power, and portable video system.

We described the three most widely known the motion estimation algorithms: the Full-Search Block Matching, Three-Step Search, and Four-Step Search algorithms. Also we reported some hybrid algorithms based upon the concepts presented within these major algorithms. Implementations and low-power enhancements for the major algorithms were also presented as the basis for introducing our proposed model and low-power enhancements.

The motion estimation design targeted for low-motion video can be applicable to videophones, mobile patrols, and videoconferencing. These videos were studied for characteristics that could be exploited in a low-power model. Based upon this research and our findings with baseline models, the 4SS algorithm was selected as the basic algorithm for our proposed model. Other models were developed based upon the FSBM and TSS algorithms for comparison with the proposed model. We proposed low-power enhancements to the basic 4SS model to further enhance power savings.

Validation of results involved both video quality and power reduction. An entire H.263 video codec was developed in software. The prototype allowed the viewing of real-world video using the proposed motion estimation algorithms. Additionally, measuring the compression efficiency (bitstream length) and video quality (PSNR) of each proposed algorithm was facilitated by the prototype. Finally, the prototype was used as a baseline for functional verification of the hardware models.

We developed hardware models to enable gate-level simulation and power-characterization using a design flow used in industry. The models were primarily developed as VHDL RTL code and verified for functionality. Then, using a 0.35 um CMOS library the RTL models were synthesized using Design Compiler. The resulting gate-level VHDL code was simulated and characterized for power dissipation with an input of data from test video.

The proposed model shows favorable characteristics for use in a real-world system. The PSNR measurements are better than or equal to that achieved using a 4SS motion estimation block. For low-motion to medium-motion video, the number of bits to transmit video increases by only 1-2%. The number of equivalent NAND2 gates of the circuit is 3331, comparable to the basic TSS model. The circuit performance allows usage of a low-speed clock, as low as 5.9 MHz to achieve 15 frames per second. The logic-unit power consumption of the circuit logic is 2.786, 2.575, and 3.634 mW for three test videos of increasing amounts of motion. These figures represent a savings of 30-40% over a baseline 4SS model, 40-60% over the TSS model, and nearly 90% over the FSBM model. The number of memory accesses required during operation are also reduced. Our proposed model reduces these memory accesses by 14% over the 4SS model for low-motion video and up to 30% for large-motion video, reducing memory power by a similar margin. The total power consumption for this block is estimated to be from 7.5-9 mW depending upon the type of video being motion estimated. This represents a savings of 20-35% over baseline 4SS models and savings of 28-43% over baseline TSS models.

From the results, it becomes evident that there is a good deal of timing slack in the circuit. This means that at the low clock rates mentioned above, for most of any given clock cycle, the circuit logic is holding state and not computing anything. It is conceivable that some parts of the logic can be reduced in drive strength to reduce power consumption. This will probably require analysis of the circuit at a circuit-level instead of the gate-level analysis presented in this thesis to more accurately optimize the power consumption. However, the reduction in power could be significant. It is left open for future research.

Bibliography

1. ITU-T Recommendation H.263. Video Coding for low bitrate communication, 01/98.
2. Vasudev Bhaskaran and Konstantinos Konstantinides. *Image and Video Compression Standards: Algorithms and Architectures*. Kluwer Academic Press, Boston, 1997.
3. J. R. Jain, A. K. Jain, “Displacement measurement and its application in interframe image coding,” in *IEEE Transactions on Communications*, vol. COM-29, No.12, pp.1799-808, Dec. 1981.
4. R. Srinivasan, K. R. Rao, “Predictive coding based on efficient motion estimationm” in *IEEE Transactions on Communications*, vol. COM-33, No. 8, pp. 888-896, Aug. 1985.
5. H. Yeo, Y. H Hu, “A novel matching criterion and low power architecture for real-time block based motion estimation.” in *International Conference on Application-Specific Systems, Architectures and Processors*, pp.122-130, 1996.
6. T. Komarek, P. Pirsch, "VLSI architectures for block matching algorithms," in *ICASSP-89: 1989 International Conference on Acoustics, Speech and Signal Processing*, vol. 4, pp.2457-60, 1989.
7. Y. S. Jehng, L. G. Chen, T. D. Chiueh, "An efficient and simple VLSI tree architecture for motion estimation algorithms," in *IEEE Transactions on Signal Processing*, vol. 41, No. 2, pp. 889-900, Feb. 1993.
8. R. Plompen, Y. Hatori, W. Geuen, J. Guichard, M. Guglielmo, H. Brusewitz, "Motion video coding in CCITT SG XV-the video source coding," in *IEEE Global Telecommunications Conference and Exhibition - Communications for the Information Age,* vol. 2, pp.997-1004, 1988.
9. R. Li, B. Zeng, M. L. Liou, "A new three-step search algorithm for block motion estimation," in *IEEE Transactions on Circuits & Systems for Video Technology*, vol. 4, No. 4, pp. 438-442, Aug. 1994.

10. Y. S. Jehng, L. G. Chen, T. D. Chiueh, "A motion estimator for low bit-rate video codec," in *IEEE Transactions on Consumer Electronics*, vol. 38, No. 2, pp.60-69, May 1992.
11. L. M. Po, W. C. Ma, "A novel four-step search algorithm for fast block motion estimation," in *IEEE Transactions on Circuits & Systems for Video Technology*, vol. 6, No. 3, pp. 313-317, June 1996.
12. P. Lakamsani, "An architecture for enhanced three step search generalized for hierarchical motion estimation algorithms," in *IEEE Transactions on Consumer Electronics*, vol. 43, No. 2, pp. 221-227, May 1997.
13. J. Y. Tham, S. Ranganath, M. Ranganath, A. A. Kassim, "A novel unrestricted center-biased diamond search algorithm for block motion estimation," in *IEEE Transactions on Circuits & Systems for Video Technology*, vol. 8, No. 4, pp. 369-377, Aug. 1998.
14. S. C. Shin, H. Baik, M. S. Park, D. S. Ha, "A Center-Biased Hybrid Search (CBHS) Method for Block Motion Estimation", in *Proceedings of the Fifth International Conference on Computer Science and Informatics*, March 2000.
15. H. Yeo, Y. H. Hu, "A novel modular systolic array architecture for full-search block matching motion estimation," in *1995 International Conference on Acoustics, Speech, and Signal Processing,* vol.5, pp. 3303-3306, 1995.
16. C. M. Wu, D. K. Yeh, "A VLSI motion estimator for video image compression," in *IEEE Transactions on Consumer Electronics*, vol. 39, No. 4, pp. 837-846, Nov. 1993.
17. H. Yeo, Y. H. Hu, "A novel modular systolic array architecture for full-search block matching motion estimation," in *IEEE Transactions on Circuits & Systems for Video Technology*, vol. 5, No. 5, pp. 407-416, Oct. 1995.
18. H. D. Lin, A. Anesko, B. Petryna, "A 14-Gops programmable motion estimator for H.26X video coding," in *IEEE Journal of Solid-State Circuits*, vol. 31, No. 11, pp. 1742-1750, Nov. 1996.
19. C. H. Hsieh, T. P. Lin, "VLSI architecture for block-matching motion estimation algorithm," in *IEEE Transactions on Circuits & Systems for Video Technology*, vol. 2, No. 2, pp. 169-175, June 1992.

20. S. H. Nam, M. K. Lee, "Flexible VLSI architecture of motion estimator for video image compression," in *IEEE Transactions on Circuits & Systems II-Analog & Digital Signal Processing*, vol. 43, No. 6, pp. 467-470, June 1996.
21. Y. K. Lai, Y. L. Lai, Y. C. Liu, P. C. Wu, L. G. Chen, "VLSI implementation of the motion estimator with two-dimensional data-reuse," in *IEEE Transactions on Consumer Electronics*, vol. 44, No. 3, pp. 623-629, Aug. 1998.
22. L. Fanucci, R. Saletti, L. Bertini, P. Moio, S. Saponara, "High-throughput, low complexity, parametrizable VLSI architecture for full search block matching algorithm for advanced multimedia applications," in *Proceedings of ICECS '99. 6th IEEE International Conference on Electronics, Circuits and Systems*, vol. 3, pp. 1479-1482, 1999.
23. J. Bonk, A. Stone, E. S. Manolakos, "Synthesis of array architectures for block matching motion estimation: design exploration using the tool DG2VHDL," in *1999 IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 4, pp. 1925-1928, 1999.
24. E. G. Tzeng, C. Y. Lee, "An efficient memory architecture for motion estimation processor design," in *1995 IEEE Symposium on Circuits and Systems*, vol.1, pp. 712-715, 1995.
25. S. C. Cheng, H. M. Hang, "A comparison of block-matching algorithms mapped to systolic-array implementation," in *IEEE Transactions on Circuits & Systems for Video Technology*, vol. 7, No. 5, pp. 741-757, Oct. 1997.
26. S. B. Pan, S. S. Chae, R. H. Park, "VLSI architectures for block matching algorithms using systolic arrays," in *IEEE Transactions on Circuits & Systems for Video Technology*, vol. 6, No. 1, pp. 67-73, Feb. 1996.
27. T. Komarek, P. Pirsch, "VLSI architectures for hierarchical block matching algorithms," in *1990 IEEE International Symposium on Circuits and Systems*, vol. 1, pp. 45-48, 1990.
28. H. Yeo, Y. H. Hu, "A high-throughput modular architecture for three-step search block matching motion estimation," in *1996 IEEE International Conference on Acoustics, Speech, and Signal Processing Conference*, vol. 4, pp. 2303-2306, 1996.

29. Z. He, M. L. Liou, "Design of fast motion estimation algorithm based on hardware consideration," in *IEEE Transactions on Circuits & Systems for Video Technology*, vol. 7, No. 5, pp. 819-823, Oct. 1997.
30. Y. S. Jehng, L. G. Chen, T. D. Chiueh, "An efficient and simple VLSI tree architecture for motion estimation algorithms," in *IEEE Transactions on Signal Processing*, vol. 41, No. 2, pp. 889-900, Feb. 1993.
31. A. Costa, A. De Gloria, P. Faraboschi, F. Passaggio, "A VLSI architecture for hierarchical motion estimation," in *IEEE Transactions on Consumer Electronics*, vol. 41, No. 2, pp. 248-257, May 1995.
32. H. M. Jong, L. G. Chen, T. D. Chiueh, "Parallel architectures for 3-step hierarchical search block-matching algorithm," in *IEEE Transactions on Circuits & Systems for Video Technology*, vol. 4, No. 4, pp. 407-416, Aug. 1994.
33. T. H. Chen, "A cost-effective three-step hierarchical search block-matching chip for motion estimation," in *IEEE Journal of Solid-State Circuits*, vol. 33, No. 8, pp. 1253-1258, Aug. 1998.
34. S. Kim, Y. Kim, K. Yim, H. Chung, K. Choi, Y. Kim, G. Jung, "A fast motion estimator for real-time system," *IEEE Transactions on Consumer Electronics*, vol. 43, No. 1, pp. 24-33, Feb. 1997.
35. Y. K. Lai, L. G. Chen, J. F. Shen, "An efficient array architecture with data-rings for 3-step hierarchical search block matching algorithm," in *Proceedings of 1997 IEEE International Symposium on Circuits and Systems. Circuits and Systems in the Information Age*, vol. 2, pp. 1361-1364, 1997.
36. Z. He, M. L. Lieu, P. C. H. Chan, R. Li, "An efficient VLSI architecture for new three-step search algorithm," in *38th Midwest Symposium on Circuits and Systems. Proceedings*, vol. 2, pp. 1228-1231, 1996.
37. H. M. Jong, L. G. Chen, T. D. Chiueh, "Modifications and performance improvements of 3-step search block-matching algorithm for video coding," in *1994 International Symposium on Speech, Image Processing and Neural Networks*, vol. 1, pp. 256-259, 1994.

38. J. B. Xu, L. M. Po, C. K. Cheung, “Adaptive motion tracking block matching algorithms for video coding,” in *IEEE Transactions on Circuits & Systems for Video Technology*, vol. 9, No. 7, pp. 1025-1029, Oct. 1999.
39. J. Lu, M. L. Liou, “A simple and efficient search algorithm for block-matching motion estimation,” in *IEEE Transactions on Circuits & Systems for Video Technology*, vol. 7, No. 2, pp. 429-433, April 1997.
40. J. N. Kim, T. S. Choi, “A fast three-step search algorithm with minimum checking points using unimodal error surface assumption,” in *IEEE Transactions on Consumer Electronics*, vol. 44, No. 3, pp. 638-648, Aug. 1998.
41. J. N. Kim, T. S. Choi, "Fast motion estimation using UESA, threshold-half-stop and adaptive partial sum scan from gradient magnitude," in *Proceedings of the 1999 IEEE International Symposium on Circuits and Systems VLSI*, vol. 6, pp. 496-500, 1999.
42. A. Wu, M. F. So, “An efficient VLSI implementation of four-step search algorithm,” in *1998 IEEE International Conference on Electronics, Circuits and Systems. Surfing the Waves of Science and Technology,* vol.3, pp. 503-506, 1998.
43. D. H. Lee, "Modified four-step block-matching algorithm efficient for hardware implementation," in *Electronics Letters*, vol. 35, No. 19, pp. 1622-1623, Sept. 1999.
44. V. G. Moshnyaga, “A new architecture for computationally adaptive full-search block-matching motion estimation,” in *ISCAS'99. Proceedings of the 1999 IEEE International Symposium on Circuits and Systems VLSI*, vol. 4, pp. 219-222, 1999.
45. V. Do, K. Yun, "A Low-Power VLSI Architecture for Full-Search Block-Matching Motion Estimation," in *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 8, pp. 393-398, Aug. 1998.
46. L. Sousa, N. Roma, "Low-power array architectures for motion estimation." in *1999 IEEE Third Workshop on Multimedia Signal Processing*, pp. 679-684, 1999.
47. L. A. Sousa, "Applying conditional processing to design low-power array processors for motion estimation," in *Proceedings 1999 International Conference on Image Processing*, vol. 2, pp. 769-773, 1999.

48. V. G. Moshnyaga, “An MSB truncation scheme for low-power video processors,” *Proceedings of the 1999 IEEE International Symposium on Circuits and Systems VLSI*, vol. 4, pp. 291-294, 1999.
49. Z. He, M. L. Liou, “Reducing hardware complexity of motion estimation algorithms using truncated pixels,” in *Proceedings of 1997 IEEE International Symposium on Circuits and Systems. Circuits and Systems in the Information Age*, vol. 4, pp. 2809-2812, 1997.
50. M. M. Mizuki, U. Y. Desai, I. Masaki, A. Chandrakasan, “A binary block matching architecture with reduced power consumption and silicon area requirement,” in *1996 IEEE International Conference on Acoustics, Speech, and Signal Processing Conferenc,* vol. 6, pp. 3248-3251, 1996.
51. B. Natarajan, V. Bhaskaran, K. Konstantinides, “Low-complexity block-based motion estimation via one-bit transforms,” *IEEE Transactions on Circuits & Systems for Video Technology*, vol. 7, No. 4, pp. 702-706, Aug. 1997.
52. M. Jagasivamani, “Development of a low-power SRAM compiler.” M.S. Thesis, Virginia Polytechnic Institute and State University, Blacksburg, VA, USA, Aug. 2000.

Vita

Richard Steven Richmond II was born in Point Pleasant, West Virginia on August 21, 1975. He completed the Bachelor of Science in Electrical Engineering degree with Special Distinction from the University of Oklahoma in Norman, Oklahoma in December 1997. He worked as a Hardware Design Verification Engineer at Hewlett-Packard's High Performance Systems Division in Richardson, Texas from January 1998 until August 1999.

He entered the Bradley Department of Electrical and Computer Engineering at Virginia Tech in August 1999 named a Bradley Fellow. He worked under Dr. Dong Ha at the Virginia Tech for VLSI Telecommunications (VTVT) Laboratory concentrating his studies on low-power VLSI design. After receiving his M.S.E.E. degree, he will work for the Hewlett-Packard Company at the Fort Collins Microprocessor Laboratory in Fort Collins, Colorado.