Synthesis of Low Power High Performance VLSI Architecture for Video Codec Chip

Synopsis seminar report submitted in partial fulfillment of the requirements for the degree of

Master of Science

in

Information Technology

by

Ranjan Maity
[Roll No. 05IT7003]

Under the supervision of
Dr. Debasis Samanta

School of Information Technology
Indian Institute of Technology, Kharagpur-721302
West Bengal, India
August, 2007
1 Introduction

In recent years, video compression becomes an important activity due to the massive digital data representation of videos. Digitally coded video must be stored in some storage space or must be transferred through the communication channel. Due to the limitation of storage space and the channel bandwidth the essence of video compression has become an important issue for researchers. Compression of video can be achieved by exploiting data redundancies in any video sequence. In any video sequences two types of data redundancy can be observed, namely spatial and temporal data redundancy. Spatial data redundancy (also known as intra-frame data redundancy) exists due to highly correlated pixel values in a particular image frame of a video sequence. Temporal data redundancy (also known as inter frame data redundancy) exists due to the correlation of pixel values between successive frames.

Compression can be applied for removing both temporal and spatial data redundancy, which obviously reduces the number of bits to be transmitted over the communication channel or to be stored in a storage space. $DCT$ (Discrete Cosine Transformation), $DWT$ (Discrete Wavelet Transformation), $HWT$ (Haar Wavelet Transformation) are some popular methodologies used for removing spatial data redundancy. In addition to spatial data redundancy, motion estimation technique is known for removing temporal (Inter-frame) data redundancy. It is reported [1] that motion estimation takes approximately 80% of total encoding time and more than 50% of total encoding and decoding time for video coding. Motion estimation is also important so far the image quality and compression ratio are concerned. A popular way of estimating motion of an image is block matching algorithm adopted by several $ISO\, MPEG$ [2-4] series and $ITU/T\, H.26X$ [5-7] standards. Block matching algorithm (BMA) subdivides a current image frame with non-overlapping rectangular blocks. Then BMA estimates the motion of each block in the current frame by evaluating all or some predefined blocks (also known as search points) inside a rectangular zone (known as search window) in the reference frame. Evaluating all the blocks inside the reference frame will produce best reconstructed video but as the number of the search points increase the computational complexity increases. So there is requirement for a block matching algorithm having as much as less number of search points, while keeping the reconstructed video quality as acceptable.

Motion compensation is a supplementary task over the motion estimation in order to reconstruct a good quality video. Motion compensation generates the predicted frame by using the set of motion vectors evaluated by the motion estimation procedure. Following this, an error image
(also known as residual image) is evaluated by subtracting the predicted image from its original image [8]. Several intra-frame coding techniques like Discrete Cosine Transformation (DCT), quantization, variable length coding, arithmetic coding etc. are applied on top of the error image (generally block-wise, 8×8). These intra-frame coding techniques are followed to achieve compression for the error image. Motion compensation followed by error frame generation and intra-frame coding of error image are again computationally complex task. So there is a requirement of developing a computationally efficient technique without degrading the reconstructed video quality.

With the leaps and bounce advancement of VLSI technology requirement of low power, faster video coding chip became an important concern. The requirement of these types of video coding chip caters a suitable solution for a wide range of portable, handheld, battery driven video coding applications like mobile, PDA etc. Low power video coding chips also require for low bit rate video communications where the main constraint is the channel bandwidth. Further, the video coding chip is also suitable for storage space optimization. However design of a low power video codec chip can be achieved by developing a less computational complex video compression method and by adopting a set of low power techniques.

Development of a low power video codec architecture can be done in two ways: using an algorithm, which is computationally economic, and adopting some low power techniques in the architecture. Till date HEXBS block matching algorithm is known as best due to its less computational complexity. There are several strategies for developing low power architecture [9]. Among them frequency scaling and voltage scaling are important. But for designing an ASIC only few levels of voltage combinations are provided by the CAD tools. There are other strategies too; principle of parallel, pipelined and parallel-pipelined architectures can be exercised within the constraint of area, time and power dissipation. When power is concern in any design, it can be further deal with several others power minimization techniques like lesser amount of data transfer from one component to others, coding of data for minimum switching activities, one hot encoding and etc.

### 2 Motivation and Objectives

According to several MPEG and H.26X standards, video coding consists of two major operations video coding (compression) and video decoding (decompression). Basic operations of video compression and decompression can be categorized as Intra-frame coding/decoding and Inter-frame coding/decoding. Inter-frames coding consumes a huge portion of computational time. For
inter-frame coding, motion estimation using block matching algorithm is known as best. A number of BMAs and their associated VLSI realizations have already been proposed. Among them FSBM (Full Search) which is an exhaustive search technique has the property of least lossy than other BMAs. But the one and only disadvantage of FSBM is the large number of search points and hence not suitable for low bit rate video applications. Among several BMAs [10-18] are important. After analyzing the above mentioned BMAs we observed that there is a clear trade off between the numbers of search points with the reconstructed video quality. Till date Hexagon based search (HEXBS) [18] is treated as the best due to its lesser number of search points compared to all existing block matching algorithms. HEXBS also maintains good reconstructed video quality. However, there is a further scope to reduce the number of search points without compromising video quality significantly.

Motion compensation and intra-frame coding on error images assure a good reconstructed video quality but it requires massive computational effort and storage/channel bandwidth for storing/transmitting the intra-frame coded residual image [8]. A possible solution may be to skip the total motion compensation and intra-frame coding of error images and accomplish frame encoding only by a set of motion vectors. But this may lead to a poor reconstructed video quality because the motion estimation procedure (BMA) fails to evaluate motion vectors accurately for non translational motion like zooming, occlusion, rotation and arrival of a new object [8]. These motivated us develop an economic motion estimation and motion compensation technique without compromising the reconstructed video quality as such.

Motion estimation and compensation followed by error image generation in video coding are computationally intensive tasks and consume more power for its operation. It was reported [19] that total 50% power of total video coding process is required for motion estimation unit. This means that the device with video processing should be supported with longer battery life. Since the battery technology has not been advanced as the VLSI technology, hence longer battery life implies heavy battery pack. The requirement of portability from user side places the restriction on size and weight of battery and heavy battery pack is not an acceptable solution. Over the last few years several VLSI [12 and 20-29] architectures for motion estimation unit has been proposed. Most of them are based on the FSBM due to its simplicity. But as FSBM is computationally intensive, fast throughput of those architectures came at the cost of huge number of functional units. Best of our knowledge motion compensation has also not considered by any one of these architectures. This has motivated us to develop a low power VLSI architecture based on a computationally economic motion estimation and compensation approach.
The specific objectives of our research work are listed below.

- Development of a faster BMA without compromising video quality.
- Development of an efficient motion compensation technique.
- Synthesis of a low power architecture based on the computationally efficient approach.
- Making it more low power by adopting several low power techniques.

3 Our Approach

In the following sections we discuss our research work. In Section 3.1 we discuss about our proposed block matching algorithm. Section 3.2 covers the efficient motion compensation technique developed by us. VLSI architecture of our motion estimation and motion compensation technique is discussed in Section 3.3.

3.1 HEXBS++ Block Matching Algorithm

After analyzing all block matching algorithms and in particular HEXBS block matching algorithm [18] we found that the search points associated with HEXBS can be reduced significantly without compromising video quality. We propose the concept of a threshold value to control searching for the best matched blocks. This threshold value is not fixed rather dynamically vary from one frame to another.

![HEXBS++ block matching algorithm](image)

**Figure 1**: HEXBS++ block matching algorithm

Two search patterns LHSP (large hexagonal search pattern) and SHSP (small hexagonal search pattern) of HEXBS++ block matching algorithm is shown in Figure 1(a). The threshold value in
HEXBS++ is denoted by $\delta$. Threshold value $\delta_i$ from an image frame at time $i$ is computed with the help of Equation 1.

$$\delta_i = \frac{\sum_{j=1}^{N} MAD^j_i}{N}$$

where $MAD^j_i$ denotes the MAD value between the $j$-th current block in the $i$-th frame and its best matched block in the reference frame, $N$ denotes the total number of blocks in an image frame. We propose to calculate the value of $\delta_i$ during the estimation of motion vectors for the blocks in the $i$-th frame (denoted by $F_i$) as shown in Figure 2. Initially, that is, at the starting of any frame sequence (see Figure 2) $\delta_i$ is trivially set to zero. For the estimation of motion vectors of each block in the $(i+1)$-th frame ($i \geq 1$) we apply LHSP and check one by one (as shown in Figure 1(b)) that if any block has MAD value less than the threshold value $\delta_i$. As soon as such a block is found, LHSP terminates, and for finding the finer motion we employ SHSP centered at the block found. Finally, we find the winner block, which is the block with minimum MAD value out of the five blocks corresponding to the SHSP. We sum up the MAD values obtained between a current block and its winner block towards the calculation of $\delta_{i+1}$. This early switching from LHSP to SHSP reduces the number of search points significantly. It may be noted that HEXBS considers at least 11 search points (7 (one LHSP) + 4 (one SHSP)), where as HEXBS++ require only 5 search points (which occurs when the center of the first hexagon has $MAD \leq \delta$, thereby 1 search point for LHSP and 4 search points for SHSP) for finding the motion of a block as shown in Case 1 of Figure 1(b). Figure 1(b) further illustrates two more cases of the HEXBS++ algorithm. For brevity, we number each search point as 0 is the centre point, 1 is the southmost point to the center and monotonically increasing in clockwise (see Figure 1(b)). Case 2 occurs when $MAD \leq \delta$ criteria matches in any one of the six neighbors of the first LHSP (excluding the center). Case 3 occurs when $MAD \leq \delta$ criteria matches after applying more than one LHSP.

### 3.2 HEXBS++ Motion Compensation

Here we are going to discuss about our motion compensation technique. We employ the threshold $\delta_{i+1}$ to decide whether motion compensations of the blocks in $F_i$ are to be performed or not. $\delta_{i+1}$ gives a measure whether a good correlation exists between a block $CB^j_i$ (the $j$-th block in the frame $F_i$) and its winner block in the reference frame. Let the MAD value of these two blocks be $MAD^j_i$. In our approach, we consider that good correlation exists if $MAD^j_i \leq \delta_{i+1}$. For a block say $CB^j_i$, if we find a winner block with good correlation then we skip the motion compensation for that block. In this case only the motion vector is used to encode the block $CB^j_i$. On the other
hand, if \( \text{MAD}_j > \delta_{i+1} \), then we perform the motion compensation which finds the residual block (between the current and winner block), followed by the intra-frame coding of the residual block. In this case the \( \text{CB}_j \) is encoded as the motion vector accompanied with the intra-frame coded residual block.

\[
\begin{align*}
\text{d}_{i-1} & \text{ is used as LHSP terminator and } \text{d}_i \text{ is evaluated while frame } F_{i+1} \text{ is processes}\end{align*}
\]

**Figure 2:** Illustration of threshold calculations with a sequence of frames

### 3.3 HEXBS++ Architecture

For VLSI realization we have considered HEXBS++ motion estimation and compensation as is less computationally complex, produce faster throughput with supported video quality. We termed our architecture as HEXBS++ architecture. The system level view of the HEXBS++ architecture is shown in **Figure 3**. Two off chip memory \( C \) and \( R \) are used to store the pixel information of the current and reference frame respectively. The working of HEXBS++ is discussed as follows. \( \text{CFLU} \) loads a block from \( C \) to \( M1 \). Similarly, \( \text{RFLU} \) loads a part of reference frame from \( R \) to \( M2 \). This part of the \( R \) is decided by the currently fetched block in \( M1 \) (related to search window) and has discussed in our Thesis. For each block loaded in \( M1 \) and its corresponding search space in \( M2 \), the \( \text{METCU} \) calculates \( \text{MAD} \) value as well as update threshold value dynamically. Based on the calculated \( \text{MAD} \) value motion vector is chosen. \( \text{AGU1} \) and \( \text{AGU2} \) generate memory references to \( M1 \) and \( M2 \), respectively for this purpose. \( \text{MCU} \) generates the error block for those current blocks, which don’t have the good correlational block in the reference frame. Experimental results discussed in **Section 4** shows that our HEXBS++ architecture achieves significant power reduction than that of the previously proposed version.
Different low power techniques used for developing our HEXBS++ architectures are listed below.

- On-chip memory with higher data reusability.
- Skipping unnecessary computations.
- Segmentation of on-chip memory with frequency scaling.
- Parallel-pipelined base approach using frequency scaling.
- Simple data scheduling between on-chip memory and METCU.
- Adopting a computationally efficient video compression approach.

**Figure 3**: Basic building blocks of HEXBS++ architecture
4 Contribution of Our Work

In this Section we discuss the significance of our work. In this Section we briefly mention the performance of HEXBS++ block matching algorithm followed by the performance of our motion compensation technique and finally the performance of the HEXBS++ architecture.

We have carried out exhaustive experiments on several video benchmarks and the experimental results of are discussed here. We have done our experiments on Caltrain (30 frames 512×400), Claire (100 frames 352×256), Garden (60 frames of 352×240), Salesman (100 frames of 352×240), Susie (50 frames 352×240), Tennis (100 frames 352×240), Football (50 frames size of 352×240), Trevor (100 frames 256×256), Miss America (100 frames 352×288) video sequences with 16×16 block size and search window size of ±8. We have implemented HEXBS++ motion estimation algorithm and also FS, TSS, NTSS, DS and HEXBS with C language in Linux operating system for all these experiments. Table 1 shows the average number of search points comparison for the above mentioned BMAs. In this context we may note that for FS and TSS the number of search points according to our search window (i.e. +7 to -8) is 256 and 25 respectively. For comparing our HEXBS++ with the best known HEXBS approach we have used \( \%SIR \) (Speed Improvement Rate) metric defined below.

\[
\%SIR_{HEXBS^{++}} = \frac{C_{HEXBS} - C_{HEXBS^{++}}}{C_{HEXBS^{++}}} \times 100
\]  

(2)

where \( C_{HEXBS} \) = number of search points per block for HEXBS algorithm and \( C_{HEXBS^{++}} \) = number of search points per block for HEXBS++ algorithm.

From Table 1 we may say that HEXBS++ achieves on an average 42 \( \% SIR \) over HEXBS. So far the reconstructed video quality Table 2 indicates that on the average 0.48\% \( MAD \) value increases in HEXBS++ compared to HEXBS. So the reconstructed video quality of HEXBS++ motion estimation is also acceptable.
We have implemented our motion compensation approach by using C language and tested on the same benchmarks used for testing HEXBS++ block matching algorithm. Our experimental results on motion compensation following the motion estimation are shown in Table 3. To judge the efficacy of our approach we consider a metric called %EBI, which implies the percentage of the block need to be motion compensated. %EBI is defined as follows.

$$\%EBI = \frac{M}{N} \times 100\%$$  \hspace{1cm} (3)
where \( N \) = total number of blocks in a current image frame,
\[ M = \text{total number of blocks in a current image frame whose corresponding winner block's } MAD > \delta. \]

From Table 3, we see that HEXBS++ requires on the average 50.44% blocks to be motion compensated. Further, with our motion compensation approach we are able to reconstruct blocks with on the average \( MAD \) value of 0.01. This means we are not losing video information significantly compared to that of the original image.

**Table 3 : Result of \%EBI and Average \( MAD \) value on different video benchmarks**

<table>
<thead>
<tr>
<th>Benchmarks</th>
<th>%EBI</th>
<th>( \text{Avg. } MAD )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Caltrain</td>
<td>40.02</td>
<td>0.007</td>
</tr>
<tr>
<td>Claire</td>
<td>74.58</td>
<td>0.002</td>
</tr>
<tr>
<td>Football</td>
<td>39.60</td>
<td>0.020</td>
</tr>
<tr>
<td>Garden</td>
<td>43.03</td>
<td>0.027</td>
</tr>
<tr>
<td>Miss America</td>
<td>49.24</td>
<td>0.005</td>
</tr>
<tr>
<td>Salesman</td>
<td>69.70</td>
<td>0.005</td>
</tr>
<tr>
<td>Susie</td>
<td>47.09</td>
<td>0.005</td>
</tr>
<tr>
<td>Tennis</td>
<td>36.87</td>
<td>0.013</td>
</tr>
<tr>
<td>Trevor</td>
<td>53.86</td>
<td>0.007</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td>50.44</td>
<td>0.010</td>
</tr>
</tbody>
</table>

Next we discuss the performance of HEXBS++ architecture. For synthesis of our architecture we have used the *Synopsis Design Vision* tool. We have used *Verilog* for the system level design of our HEXBS++ architecture. For realization of our architecture, we have applied 1.8\( V \) as supply voltage and 20\( MHz \) as the global operating frequency. Our proposed architecture requires only 14.79 \( mW \) power for its operation and only 4\( K \) gate counts excluding the on-chip memory. The result of our synthesized architecture is compared with the reported work [12], [27-29]. We compare our architecture with others with reference to the gate count, size of the on-chip
memory, delay and power. Since reported work use different process technologies and synthesis platforms, it is not possible to directly refer to the values of delay and power as reported in our comparisons. We consider a relative measurement for the delay and normalized measurement for the power. Suppose $T$ represents the time required for comparing one pixel value during the motion estimation and compensation. We can express the delay of an architecture considering the data paths and expressing the number of operations on the longest data path multiplied by $T$, we critically analyze all the data paths in an architecture and then find the number of comparisons involved. In order to express the power in normalized form, we consider $1.8 \, V$ as the base value for supply voltage and of $20 \, MHz$ as the base value for operating frequency. With reference to these base values, we express the power of an architecture following Equation 4, where $P_{\text{norm}}$, $V_i$ and $f_i$ are the normalized power, supply voltage and operating frequency, respectively. It can be noted that $\frac{C_L'}{C_L} > 1$ if we consider $C_L$ is the process parameter (for capacitance value) in a next generation of process technology compared to that of $C_L'$. For brevity we assume $\frac{C_L'}{C_L} = 1$. This normalized power expression is considered just to draw a comparison between any two architectures only.

$$P_{\text{norm}} = \frac{C_L'}{C_L} \left( \frac{V_i}{1.8} \right)^2 \frac{f_i}{20} \tag{4}$$

Experimental results furnished in Table 4 shows that proposed HEXBS++ architecture is area efficient compared to all architecture except Sarma et al. [12]. However, it can be noted that [12] requires more on-chip memory than the HEXBS++ architecture. So far the speed is concerned, architecture proposed by Lin et al. [27] is comparable to our architecture HEXBS++. This is due to the high degree of parallelism, nevertheless is at the cost of large number of processing elements. Power requirement in our architecture is significantly comparable to all existing architectures as shown in Table 4. We observe that HEXBS++ requires least power ($P_{\text{norm}} = 1$) compared to the architectures proposed in Lin et al. [27] ($P_{\text{norm}} = 3.47$ with degree of parallelism 4), Lin et al. [27] ($P_{\text{norm}} = 1.63$ with degree of parallelism 16), and He at al. [12] ($P_{\text{norm}} = 2.42$).
Table 4: Comparisons of HEXBS++ architecture with others

<table>
<thead>
<tr>
<th>Work</th>
<th>Gate Count</th>
<th>On-chip memory(Kb)</th>
<th>Delay</th>
<th>Power (normalized)</th>
<th>BMA used</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lin et al. [27]</td>
<td>18K</td>
<td>20.48</td>
<td>1024T</td>
<td>3.47</td>
<td>FS</td>
</tr>
<tr>
<td>parallel: 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lin et al. [27]</td>
<td>68K</td>
<td>20.48</td>
<td>256T</td>
<td>1.63</td>
<td>FS</td>
</tr>
<tr>
<td>parallel:16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>He et al. [28]</td>
<td>13K</td>
<td>-</td>
<td>4096T</td>
<td>2.42</td>
<td>FS</td>
</tr>
<tr>
<td>Sarma et al. [12]</td>
<td>3K</td>
<td>2016</td>
<td>2816T</td>
<td>-</td>
<td>MRTSS</td>
</tr>
<tr>
<td>He et al [29]</td>
<td>32.9K</td>
<td>-</td>
<td>2070T</td>
<td>-</td>
<td>NTSS</td>
</tr>
<tr>
<td>HEXBS++</td>
<td>4K</td>
<td>64</td>
<td>692T</td>
<td>1</td>
<td>HEXBS++</td>
</tr>
</tbody>
</table>

5 Organization of the Thesis

In this Section we have discussed about the organization of the thesis. Our thesis is subdivided into 7 chapters to present the complete research work carried out by us as and is stated below.

Chapter 1: In this chapter a brief introduction to the state-of-the-art, scope and need, issues to be addressed etc. are discussed.

Chapter 2: In this chapter we have discussed a brief overview on the different motion estimation and motion compensation approaches. We have also reviewed some important VLSI architectures for the motion estimation block.

Chapter 3: Our proposed approach of block matching algorithm is discussed in this chapter.

Chapter 4: Our approach of motion compensation is discussed in this chapter.

Chapter 5: Based on our approach of block matching algorithm and motion compensation technique and a set of low power properties we have developed a low power VLSI architecture, which is presented in this chapter.

Chapter 6: Experimental results of our approaches are furnished in this chapter.

Chapter 7: In this chapter, we have concluded our work and discussed future scope of our work.
References


Publications


