# A Parallel Processing Engine for Motion Estimation in MPEG-4 Multimedia Applications

RUI GAO, DONGLAI XU and JOHN P. BENTLEY
School of Science and Technology
University of Teesside
Borough Road, Middlesbrough, TS1 3BA
UNITED KINGDOM

*Abstract:* In this paper, a parallel VLSI processing architecture for MPEG-4 standard motion estimation implementation is proposed. It possesses the characteristics of low memory bandwidth and low clock rate requirements, thus primarily aiming at 3G mobile applications. Based on one dimensional tree architecture, the processing core employs dual-register/buffer technique to reduce the preload and alignment cycles. As an example, a full-search block matching algorithm has been mapped on this architecture using a 16-elements PE array, which has the ability to calculate motion vectors of QCIF video sequences in real time at 8.2 MHz clock rate and using 15.5 Mbytes/s memory bandwidth.

*Key-Words:* Motion estimation, VLSI, parallel architecture, MPEG, multimedia, video compression

## 1 Introduction

The third generation (3G) wireless system has recently been becoming popular in wireless telecommunications. It provides the high-speed mobile platform with Internet Protocol (IP) [1], which allows many types of IP-based internet applications to be implemented on the 3G platform, such as e-mail service, web page browsing and image/video transmission. Among these, real-time video applications are becoming an increasingly important part of mobile multimedia [2]. However, due to the inherently huge size of video data, video compression techniques are required to reduce the size of the video data. This is achieved by exploiting the redundant data of the video stream. There are two types of redundancy in the uncompressed video stream: temporal and spatial redundancy. Generally, spatial redundancy is reduced by discrete cosine transformation (DCT) and other transformation techniques, while the temporal redundancy is reduced by motion estimation (ME) techniques. ME operations can normally take up to 80% of the computational burden of the completely video compression procedure [3]. Therefore, it is the most important component in real-time video applications. Many VLSI architectures for ME have been proposed. However, most of them target MPEG video coding standards, which are mainly for conventional video applications, such as videophone, video conferencing, video broadcasting, etc. These architectures are not particularly suitable for mobile and low power applications [4]. In this paper, a low power consumption architecture that is based on the 1D tree architecture [5] is presented. It features the high data utilisation by using parallel pipelining and the low clock rate by introducing dual-register/buffer techniques that reduce idle clock cycles.

The rest of the paper is organised as follows. In section 2, two typical ME algorithms are briefly described. Section 3 presents the proposed VLSI architecture in detail. In section 4, the performance of the core architecture is analysed in terms of minimum clock rate and minimum memory bandwidth requirement. Finally, conclusions are drawn and future work on the core architecture is suggested in section 5.

## 2 Motion Estimation Algorithms

Recently, MPEG-4 video standard, which is widely used in conventional video transmission and storage, has been introduced to wireless multimedia applications [2]. It adopts block-matching algorithms with alpha binary plane to archive motion estimation [3][6]. Fig.1 illustrates the principle of block matching motion estimation algorithms. First, the video frames are segmented into *N*×*N* blocks. Every block within the current frames is matched to the corresponding blocks within the search range on the previous frame under given algorithms. A measurement of the difference between the current block and candidate blocks is calculated. Then, a motion vector for the position of the candidate block, which has the minimum measurement with the current block, is generated. It is used to replace the real motion of the objects in the compressed video stream [3]. Thus, temporal redundancy is reduced.

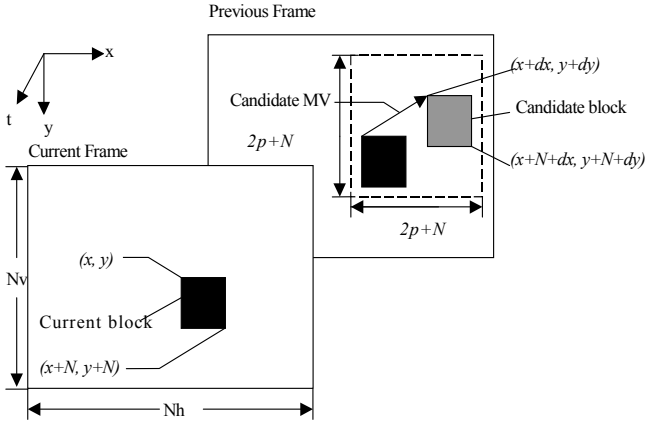


Fig.1 Principle of block-matching algorithms

### 2.1 Full-search block-matching algorithm

Because of its low distortion and regular data flow, the full-search algorithm is the most popular algorithm used in ME. In the full-search block-matching algorithm, the current block is matched to every candidate blocks within the *(2p+N)*× *(2p+N)* search window, where *(-p, p-1)* is pixel search range. For every candidate block, every pixel must be matched as well as the blocks, and then a sum of absolute difference (SAD) is calculated. It is given by

$$SAD(dx,dy)=\sum_{m=x}^{x+N-1}\sum_{n=y}^{y+N-1}\left|I_k(m,n)-I_{k-1}(m+dx,n+dy)\right|$$

where $I_k$, $I_{k-1}$ are the intensity values of the pixels in current and previous blocks, respectively, which are located at position *(x, y)*. When the SAD of the current candidate block is obtained, the SAD of the next candidate block is calculated. Then two SADs are compared to find smaller one to be stored as the minimum SAD. This process continues until all blocks are matched and a final minimum SAD, whose position is pointed by motion vector, is generated [3].

### 2.2 Object-oriented motion estimation

The recently finalised MPEG-4 standard emphasises object-oriented video coding, which is different from previous visions of video compression standards [6]. To support the motion estimation of arbitrary-shaped objects, an alpha binary plane has to be defined. The alpha plane contains information as to whether a pixel is inside the object or not [3]. Thus, the SAD for the object can be represented below:

$$SAD(dx,dy)=\sum_{m=x}^{x+N-1}\sum_{n=y}^{y+N-1}\left|I_k(m,n)-I_{k-1}(m+dx,n+dy)\right|\times Alpha(x,y)$$

where *Alpha (x, y)* is the binary value for the *(x, y)* pixel in the current block. The value is one when the pixel is inside the object; otherwise, it is zero, as shown in Fig.2.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Fig.2 Illustration of alpha binary plane

# 3 Proposed Architecture

In this section, we describe the core architecture of the proposed ME engine and its main components, in particular the PE array, since it is the most computationally intensive part of the whole architecture.

## 3.1 System overview

The Fig.3 shows the block diagram of the ME engine, which includes five components: memory unit, address generator, PE array, minimum unit and control CPU [3].

The memory unit is divided into two modules. One is to store current frame data and alpha plane data; the other is for previous frame data. The address generator computes the addresses, at which the candidate pixels for the block matching are stored. It also fetches pixel data from the memory unit and feeds them into the PE array. In the PE array, the absolute difference between previous and current frames is computed and sent to the minimum unit. Then, the SADs of all parallel-processed blocks are accumulated in the minimum unit, and are compared to find the minimum one to be stored in minimum SAD register. At the same time, a minimum flag signal is output to the control CPU, which, jointly with address generator, gives the location where the MV is found.

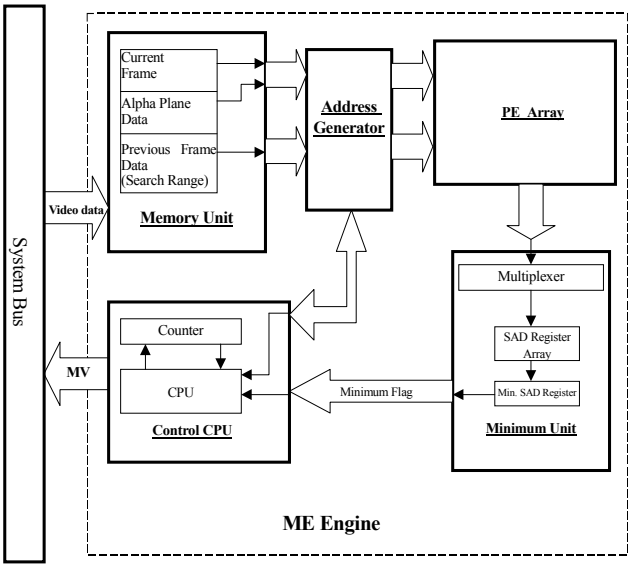


Fig.3 System overview

## 3.2 PE array

The PE array is the key component of the ME engine. It determines the performance of the system in terms of memory bandwidth and minimum clock rate for real time processing. Its architecture is based on a 1D tree architecture [5] with additional preload cycles. To increase the parallelism of the data flow, a group of parallel-pipelined processing elements have been used in the PE array, as shown in Fig.4.

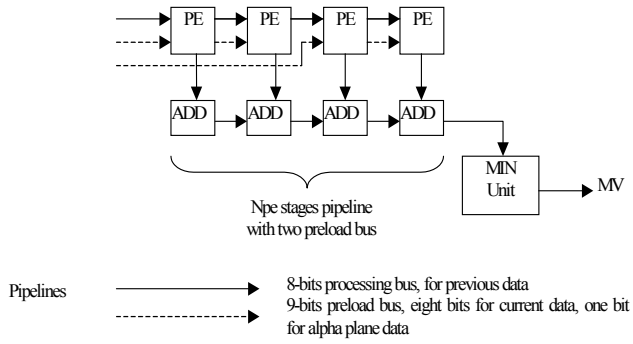


Fig.4 PE array architecture

In this architecture, motion estimation is carried out in two stages, preload and matching. In preload cycles, as shown in Fig.5, current block data and alpha plane data are preloaded into the PE array. They are stored locally in the appropriate PEs. Then, as illustrated in Fig.6, showing matching cycles, the previous block data are

loaded into the PEs by parallel pipelining. Before matching to the preloaded current data, the previous data must align with the current data. It takes $N_{PE}$ clock cycles to align the previous data with the current data, where $N_{PE}$ is the number of PEs. While the SAD calculation starts, the previous data shift from the left to right within the PE array until they match the corresponding current data already in the PE array. In every clock cycle, $N_{PE}$ absolute difference values for each of the parallel-processed blocks are calculated; these are summed by the adders physically below the PE array (Fig.4). The summed result is then sent to the minimum unit to calculate the SADs for each of the matching points and find the minimum SAD for the motion vector.

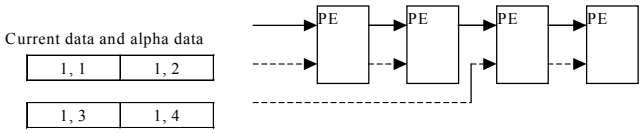


Fig.5 Preload cycles

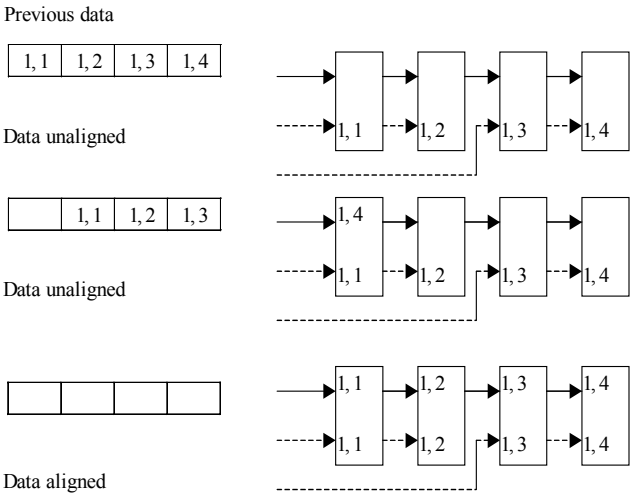


Fig.6 Matching cycles

Apparently, there should be $N_{PE}$ processing elements in the PE array. Here, we assume that $N_{PE}$ is 16, and as an example, full search BMA has been chosen to evaluate this architecture. In this case, $N_p$ blocks of the candidates can be processed simultaneously, and pipelining can be organised as in Fig.7.

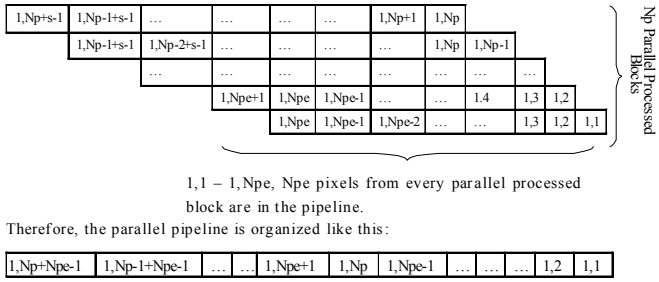


Fig.7 Pipeline organisation

### 3.3 Dual register/buffer

The architecture presented in section 3.2 has demonstrated that current data and alpha plane data need to be loaded only once when processing $N_p$ candidate blocks. Hence, as $N_p$ is increased, the bandwidth required for processing current data is sharply reduced. However, extra clock cycles are needed to preload current data and align the previous data with current data, thus the PEs are in idle status during the preloading and the aligning. For instance, $N_{PE}$ / 4 (with four preload bus) cycles are needed to preload current data and alpha plane data, and $N_{PE}$ cycles are needed to deal with data alignment. This can cause a high clock speed requirement. To solve this problem, a dual register/buffer structure has been adopted in the processing element. As illustrated in Fig.8, in each of the PEs, there are two 8-bit registers for the previous data and two 9-bit registers for the current and alpha plane data, respectively, to allow preloading and matching to be performed simultaneously. While the PEs are matching the data in register Group A, the following data are preloaded into register Group B. In addition, when the matching operations of the data in Group A are finished, the PE switches operational mode so that the PEs match the data in Group B, while the Group A register is during preloading cycles.

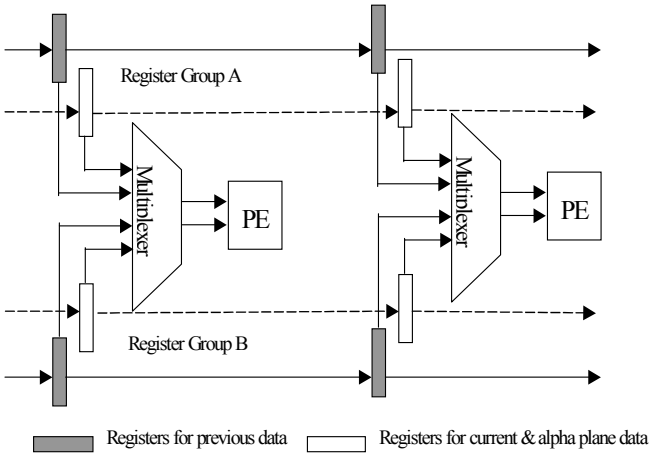


Fig.8 Dual register architecture

### 3.4 High Parallelism

High-paralleled processing is different from other architectures [3]-[5], it can be easily achieved on this architecture, since the architecture is capable of processing higher numbers of blocks in parallel (more than 16), such as 32, 64 or 128, etc. Fig.9 illustrates the data flow organisation of the architecture processing 32 blocks in parallel. In this illustration the pixel search range is *(-p, p-1)* $p = 8$, and the block size is $N \times N$, where $N$ is 16. When the data in the first row (from *(1, 1)* to *(1,16)*) of the current block is in the PE array, as shown in Fig.9 (a), all the previous data need to be matched in the search range, as shown in Fig.9 (b). The data required to match the first sixteen blocks is the first row of the search range, as shown in Fig.9 (c). To achieve high-paralleled processing, we simply load the data of blocks 17-32, the second row of the search range, after completing the matching of blocks 1-16 without changing the current data, as shown in Fig.9 (d). Hence, there is neither the need to access the memory for another group of current data, nor for preloading cycles. Furthermore, with a dual register/buffer structure, the data in the second row are loaded into register Group B at the same time of matching the first row data in register Group A. This allows the alignment cycles to be skipped for the previous data.

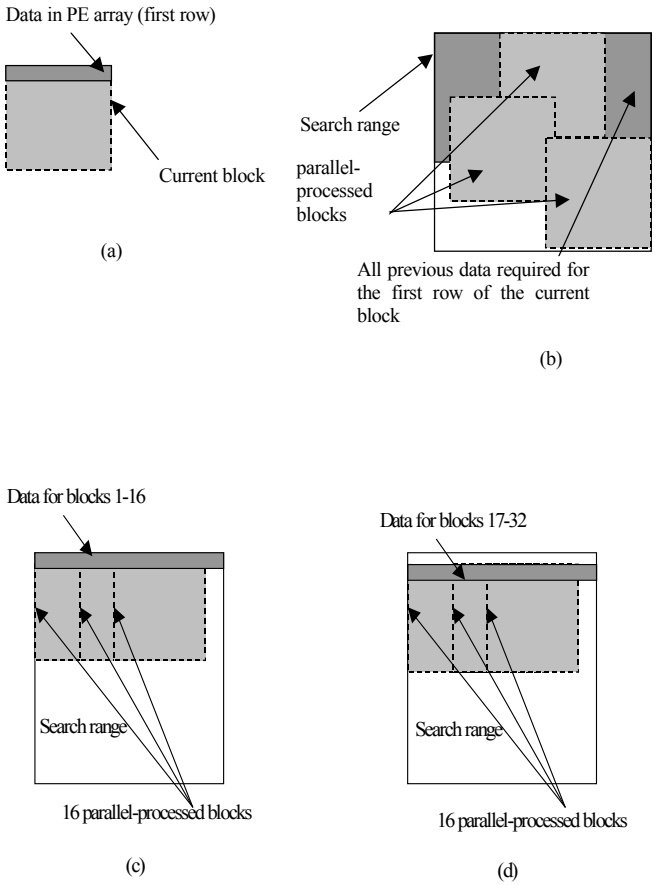


Fig.9 Data flow for previous block (search range)

## 4 Performance Analysis

This section analyses the minimum clock rate and the minimum bandwidth requirements, which reflect the performance of the proposed architecture and the main considerations for its VLSI design and implementation.

### 4.1 System requirement

This core architecture is proposed for the 3G mobile platform, which currently has 64kbits bandwidth to upload and transfer data. As a result, a minimum compression rate of 70 is required to achieve real time video applications with acceptable visual quality [4]. If we adopt QCIF as typical video format in mobile applications, the quantity of uncompressed video data per frame in kilobits should be

$176 \times 144 \times 8 / 1024 = 198 kbits$,

and the quantity of compressed data should be

*198 / 70 =2.829kbits.*

Therefore, the video transmission rate over the 3G platform should be

*64 / 2.829 = 22.628 frames/second.*

Taking into consideration the bandwidth requirements of audio and protocol, the maximum frame rate is going to be 20 frames per second, which determines the minimum clock rate for real time processing.

### 4.2 Minimum clock rate

To meet the real-time processing condition, $\frac{N_h \times N_v}{N^2} \times fps$ current blocks have to be matched per second, where $N_h \times N_v$ is the frame size. It is *176×144* for the example specified above. *N×N* is the block size, which equals *16×16* in our case. For every current block, there are $N_{can} = (2p) \times (2p)$ candidate blocks in the search range, where $p$ specifies the search range *(-p, p-1)*. Therefore, there are $\frac{N_h \times N_v}{N^2} \times fps \times (2p)^2$ candidate blocks to be matched per second. The current blocks are divided into a group of $N_{PE}$ - pixel sub-blocks, in which the pixels can be matched simultaneously within the PE array. The number of clock cycles to match a sub-block with $N_p$ candidate blocks, which are processed simultaneously, is defined as $C_{sub}$. In addition, the number of clock cycles needed to preload current data and alpha plane data is defined as $C_{pre}$, and the number of the cycles for matching and aligning are defined as $C_{match}$ and $C_{align}$, respectively.

Thus, we have

$$C_{sub} = C_{pre} + (C_{align} + C_{match}) \times N_p / N$$

$$N_{sub} = N^2 / N_{PE}$$

$$C_{pre} = N_{PE} / N_{pre}$$

$$C_{align} = N_{PE} - 1$$

$$C_{match} = 2p,$$

where $N_{PE}$ is the number of processing elements; $N_{pre}$ is the number of preload bus; $N_{sub}$ is the number of sub-blocks within a block in the current frame. Moreover, $C_{sub}/N_p$ is the number of clock cycles required to match a sub-block to one of the parallel processed blocks.

Therefore, the minimum clock rate required for real-time processing is given by:

$$C_{clk} = C_{sub} / N_p \times N_{sub} \times N_{can} \times fps \times N_h \times N_v / N^2$$

$$C_{clk} = \frac{[N_{PE} / N_{pre} + (N_{PE} - 1 + 2p) \times N_p / N] \times N^2 \times (2p)^2 \times fps \times N_h \times N_v}{N_p \times N^2 \times N_{PE}}$$

For the PE array with the dual register/buffer structure, there are only $N_{PE} - 1$ preload cycles in the beginning of motion estimation process. Hence, the minimum clock rate can be calculated as:

$$C_{clk\_double} = C_{sub\_double} / N_p \times N_{sub} \times N_{can} \times fps \times N_h \times N_v / N^2 + N_{PE} - 1$$

$$C_{clk\_double} = 2p \times \frac{N_p}{N} \times \frac{1}{N_p} \times (2p)^2 \times \frac{N^2}{N_{PE}} \times \frac{N_h \times N_v \times fps}{N^2} + N_{PE} - 1$$

The pictorial representation of these formulas is given in Fig.10, which clearly shows the relationship between the minimum clock speed and $N_p$ (where $N_{pre}$ is 4). It can be easily seen from the figure that the minimum clock speed required reduces while $N_p$ increases for single register structures; also the dual register structure requires much lower clock speed than the single register structure.

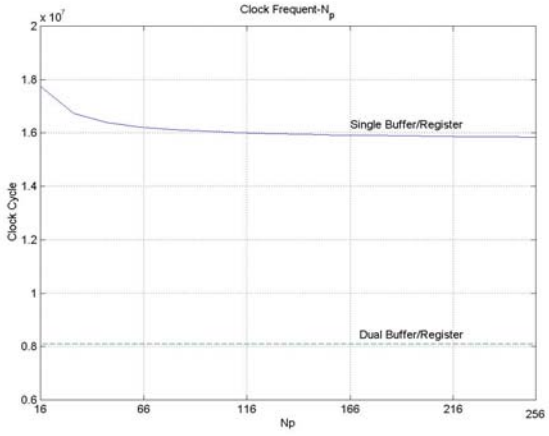


Fig.10 Relationship between clock speed and $N_p$

### 4.3 Minimum memory bandwidth

Power consumption is another important consideration for the intended mobile applications. For ME algorithms, memory access operations are the dominant factor that contribute to the power

consumption, rather than clock rate [3]. For parallel-pipelined architecture, as illustrated in Fig.7; if $M_{BW}$ represents the total amount of data fed to the PE array per second, $M_{BW}$ is equal to the quantity of memory access for every candidate block multiplied by quantity of candidate blocks. Therefore,

$$M_{bw} = (Q_{current\&alpha} + Q_{Previous}) / N_p \times N_{sub} \times N_{can} \times fps \times N_h \times N_v / N^2$$

$$Q_{current\&alpha} = N_{PE} \times 9$$

$$Q_{previous} = \frac{N_p}{N}(N + 2p - 1) \times 8,$$

where $Q_{current\ \&\ alpha}$ is the quantity of current and alpha data memory access for every sub-block; and $Q_{previous}$ is the quantity of previous memory access for $N_p$ candidate sub-blocks.

Then,

$$M_{bw} = \frac{[N_{PE} \times 9 + (N + 2p - 1) \times 8] \times N^2 \times (2p)^2 \times fps \times N_h \times N_v}{N_p \times N^2 \times N_{PE}}$$

If the preload bus is a 9-bit bus, 8 bits are needed for the current block and 1 bit for the alpha plane data. The matching data bus is an 8-bit bus for the previous data. The above formula can also be represented pictorially, as shown in Fig.11.

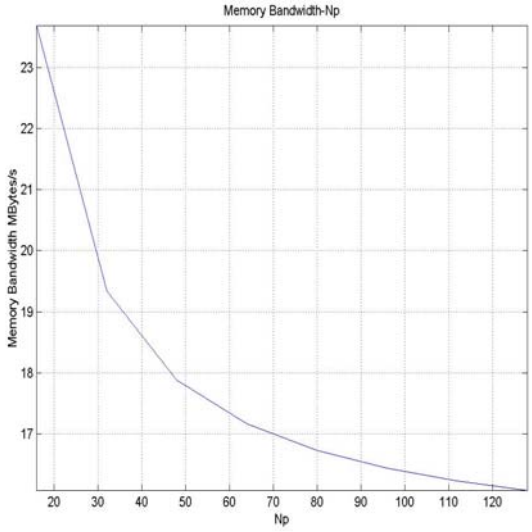


Fig.11 Relationship between Minimum Memory Bandwidth and $N_p$

From this figure, it is clear that memory bandwidth is sharply reduced while $N_p$ increases, especially when the $N_p$ is between 16 and 32. In addition, architectures with single and dual register/buffer structures have the same quantity of memory access per second. Therefore, they have the same minimum memory bandwidth requirement. Table 1 presents detailed results for clock speed and memory bandwidth analysis; $N_{cycle}$ being the numbers of clock cycles needed to match one current block.

Table 1. Clock speed and memory bandwidth

| Np | CLK Single Register | CLK Double Register | Memory Bandwidth (bytes/s) | N cycle with Single Register | N cycle with Double Register | Bytes /MB |
|---|---|---|---|---|---|---|
| 16 | 17,740,800 | 8,110,080 | 24,837,120 | 8,960 | 4,096 | 12,544 |
| 32 | 16,727,040 | 8,110,096 | 20,275,200 | 8,448 | 4,096 | 10,240 |
| 64 | 16,220,160 | 8,110,096 | 17,994,240 | 8,192 | 4,096 | 9,088 |
| 128 | 15,966,720 | 8,110,096 | 16,853,760 | 8,064 | 4,096 | 8,512 |
| 256 | 15,840,000 | 8,110,096 | 16,283,520 | 8,000 | 4,096 | 8,224 |

## 5 Conclusions and future work

This paper presents the core architecture of a highly parallel-processing motion estimation engine, aiming at 3G mobile applications. Initial analysis shows that the architecture requires relatively low memory bandwidth and clock rate, and is therefore suitable for low power consumption and low cost VLSI design/implementation. Moreover, due to the adoption of the dual-register structure, the architecture significantly speeds up data processing and therefore provides high throughput. These make the architecture ideal for the mobile video applications. Future work following on from this proposed architecture will include detailed performance evaluation and the comparisons with a wide range of existing architectures, as well as architectural refinement, design and optimisation for the targeted applications.

*References:*

[1] Z. E. Lee-Hamlin, Evolution in the Technological Revolution: Preparing for 3G Wireless Technology, *National Urban League Technology Policy Alert*, January, 2001.

[2] S. N. Fabri, S. Worral, A. Sadka, and A. Kondoz, Real-time Video Communications over GPRS, *University of Surrey*, Unite Kingdom.

[3] P. Kuhn, *Algorithms, Complexity Analysis And VLSI Architecture for MPEG-4 Motion Estimation,*

KLUWER ACDEMIC PUBLISHERS, 2001.
[4] A. A. J. Roach and A. Moini, VLSI Architecture for Motion Estimation on a Single-chip Video Camera, *In Visual Communications and Image Processing 2000, Proceedings of SPIE*, Vol.4067, 2000.
[5] Y. S. Jehng, L. G. Chen and T. D. Chiueh, An Efficient and Simple VLSI Tree Architecture for Motion Estimation Algorithms, *IEEE TRANSACTIONS ON SIGNAL PROCESSING*, Vol.41, No.2, February 1993.
[6] T. Ebrahimi and C. Horne, MPEG-4 Natural Video Coding - An Overview, *Signal Processing: Image communication*, Vol.15, 2000, pp.365-385.