A low power ASIC Design of a FSBM Motion Estimator
For H.264 AVC

Majdi ELHAJI, Abdelkrim ZITONI and Rached TOURKI

Laboratory of Electronic and Micro-Electronic (LAB-IT06)
Faculty of Sciences of Monastir, 5019, Tunisia
Majdielhaji@yahoo.fr
Rached.tourki@fsm.rnu.tn
Abdelkrim_zit@yahoo.fr

Abstract: In a video Codec such as H264 AVC, motion estimation comprises one of the most important compression methods for video communications. Since this task is computationally intensive to result in large power consumption, a low-power design is essential for portable or mobile systems. In this paper, a low power full-search block matching (FSBM) motion estimation design for the H.264 low bit rate video coding was proposed. It presents an ASIC design of a two versions (Synchronous and Gated clock) of a FSBM using 130 nm CMOS technology. ASIC synthesis and power consumption estimation flows are based on Synopsys and Cadence tools. Experimental results show that the gain in consumption obtained by the gated clock version compared to the synchronous version of the estimator is about 23.86% with a penalty in terms of area only about 3.76mm².

Key words: H.264, FSBM

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.

Since video compression is computationally intensive, low-power implementations are critical for widespread acceptance and use of video devices. The FSBM algorithm is the most straightforward motion estimation operation. The process of block-matching is to find in a search window of previous frames the macro blocks most similar to the macro blocks in the current frame. The accuracy of ME depends on the matching criteria, and one of the most popular criteria is the sum of absolute difference, or SAD, given by:

\[
SAD(k,l) = \sum_{i=1}^{16} \sum_{j=1}^{16} |P(i,j) - P_{-1}(i + k, j + l)|
\]

where \((k, l)\) is the location in the search window, \(P(i, j)\) is a pixel at \((i, j)\) in the current frame, and \(P_{-1}(i, j)\) is a pixel in the previous frame. When the value \(SAD(k,l)\) is minimum, \((k,l)\) is the motion vector of the macro block. 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 [Kom 89] Programmable devices are becoming more and more complex and popular. For these devices power consumption is a major concern, as a consequence, the market is up to now mainly dominated by advanced ASICs commonly named System on Chip.

The organization of this paper is reported as follows. Section 1 presents the related works. Section 2 describes the FSBM estimator architecture with the technique of using the gated clock. Section 3 shows the ASIC implementation and power consumption for the proposed motion estimator also the synchronous version. Section 4 points out the main differences between the proposed motions estimators with some other famous architectures presented in the literature. Section 5 concludes the paper.

1. Related works

A few attempts have been made in reducing the power consumption of FSBM implementations [Mos 99], [DO 98], [Sou 99], [La 99], [Geo 01] In the case of large throughput requirements; the motion estimator is responsible for nearly 60% of the power dissipated in certain FSBM video encoders [Mos 99]. Thus, any reduction in power in the motion estimator would be critical for inclusion in a portable or other power-conscious device. In [Mos 99], Moshnyaga introduced...
the idea of a SAD criterion for shutting down the PE after a certain point in the calculation of a motion vector. The proposed block alters the FSBM algorithm by monitoring picture variation. In [DO 98], [Sou 99] and [La 99] Do and Sousa presented an alternative strategy to shutting off the PE. A conservative estimate is calculated for each row of the macro-block. 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 [DO 98]. Hardware structures are also presented to accomplish the estimation and the comparison and stop operations [Mos 99] [Tak 00] Alternative implementations to this algorithm based upon the same concepts of a conservative SAD estimate calculation and comparison have been presented in [DO 98], and [Sou 99]. For the ASIC world many low power design techniques have been proposed to deal with the two power components at different levels are summarized in [Ama 05] and [Ben 98]

2. Motion estimator architecture

The performance characteristics of the motion estimation block that we have developed meets the following requirements.

- Perform motion estimation on 16x16 pixel macroblocks.
- Use a search area of [-7,+7].
- Perform motion estimation for 99 macro-blocks per frame (QCIF-size).

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. The current frame search block and the previous frame search area are stored in local SRAM.

![Figure 1. Architecture of the proposed ME](image)

The proposed ME circuit is constituted by various modules such as PE (processing element), DFF (Delayed flip-flop), multiplexers, a logic block, an address generation block and a gated clock module. The block diagram of the motion estimator is shown in figure 1, where a is the data from a macro block, and b0, b1 are the data in two sub-regions of search window. The SAD is calculated in each of 16 PE. Figure 2 shows the conventional architecture of a PE. Each data a from a macro block is send to the next PE by DFF. The data flow for a PE is shown in Table 1. The first SAD in each PE (from PE0 to PE15) is output sequentially from 256 clock cycles to 271 clock cycles. After that, the SAD in each PE (from PE0 to PE15) is output sequentially every 256 clock cycles. The minimum SAD in a 16x16-block region is selected in the comparator module and the motion vector is given.

![Figure 2. FSBM processing element](image)

### Table 1. A data flow diagram for a FSBM

<table>
<thead>
<tr>
<th>Clock cycle</th>
<th>Time segment</th>
<th>PE0</th>
<th>PE1</th>
<th>PE2</th>
<th>PE3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>a0</td>
<td>a0</td>
<td>a0</td>
<td>a0</td>
<td>a0</td>
</tr>
<tr>
<td>1</td>
<td>a0</td>
<td>a0</td>
<td>a0</td>
<td>a0</td>
<td>a0</td>
</tr>
<tr>
<td>2</td>
<td>a1</td>
<td>a1</td>
<td>a1</td>
<td>a1</td>
<td>a1</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>256</td>
<td>a0</td>
<td>a0</td>
<td>a0</td>
<td>a0</td>
<td>a0</td>
</tr>
</tbody>
</table>

Referring to Table 1, 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 PE to achieve the pixel flow in Table 1. 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 Adresse X counter. This counter counts as long as the ENABLE line is asserted. 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 Adresse Y counter. This counter is reset for every row of motion vectors to calculate. Adresse 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 Adresse X and Adresse Y counters roll over. Adresse Y and Adresse MVY are added to from the upper 5 bits of the left-hand memory address. According to Table 1, 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. However, the y-offset of the right-hand address is a delayed version of the y-offset of the left-hand address. 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. 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 illustrates the logic block.

### Table 2. Memory select truth table

<table>
<thead>
<tr>
<th>Compteur adder_x</th>
<th>Processing element memory bank select</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0 1 2 3 4 5 6 7 8 10 11 12 13 14 15</td>
</tr>
<tr>
<td>1</td>
<td>0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>2</td>
<td>0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>3</td>
<td>0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>4</td>
<td>0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>5</td>
<td>0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>6</td>
<td>0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>7</td>
<td>0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>8</td>
<td>0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>9</td>
<td>0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>10</td>
<td>0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>11</td>
<td>0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>12</td>
<td>0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>13</td>
<td>0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>14</td>
<td>0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>15</td>
<td>0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1</td>
</tr>
</tbody>
</table>

#### 2.2. Control logic block

The other major control logic is the line to each PE to select which side of the candidate memory to use. It turns out that this logic can be based entirely on the 4-bit Adresse X counter. It consists of a 16-bit shift-register. Referring back to Table 1, the pattern of which memory to use repeats every 16 clocks, coinciding with this counter. Table 2 gives the truth table of this logic for every PE.

#### 2.3. Comparator architecture

The comparator unit (Figure 4) is responsible for determining which SAD is the lowest value and saving both The RESET_PE lines signal when a SAD is ready in the respective PE. The encoder encodes this code for the 16:1 MUX. The inputs to the 16:1 MUX are the 16 bit SAD values from the PE. 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 and one of the SAD PE active (OR gate in the logic). Also, if the comparison is valid, the motion vector registers (MVX and MVY) reload a new motion vector.

#### 2.4. Gated clock block

Referring to Table 1, we notice that the PE does not operate simultaneously during the life time of the system. In fact, during the first 16 clock cycles only the PE0 begins its treatment at \( t = 0 \) to \( t = 15 \). The other PE begins their treatments after a number of clock cycles. The PE2 began its treatment at \( t+2 \), the
PE_0 at t+3, and so on. Also, we notice that the PE_0 finishes its treatment at the 255^{th} clock cycle. One clock cycle after it is the PE_1 that finishes its treatment, and so on.

The task of the gated clock module consists on shutting down each PE when it is not under running by gating its clock with a specific signal S_0. The clock of each PE will be activated at the moment when the latter begins its treatment. The finite state machine (FSM) of the gated clock module is shown in figure 5.

![Figure 5. FSM of the gated clock block](image)

### 3. ASIC power consumption

There are two main components, dynamic and static components as shown by the following equation:

$$P_d = \alpha C_{i} V_{dd}^2 F + I_{0} \times 10^{-\nu_{t}/S} V_{dd}$$

The first term denotes the dynamic power, \(\alpha\) is the activity of the circuit, \(C_i\) is the parasitic capacitance, \(V_{dd}\) the power supply and \(F\) the operating frequency. The second term denotes static component which is dominated so far by the sub- threshold component, \(V_{t}\) is the threshold voltage of the transistor and \(S\) the slope factor. The dynamic component is a quadratic function of \(V_{dd}\) and the sub- threshold component is an exponential function of \(V_{t}\) which makes it crucial for coming deep-submicron technologies. By reducing \(V_{dd}\) one can reduce drastically the dynamic component, but unfortunately at the expense of speed degradation.

For logical and physical ASIC design of the motion estimators we have used the Synopsys Design Vision and the Cadence Encounter (CMOS 130 nm) environments respectively. In the gated motion estimator, the use of gated clock technique reduces the parameter \(\alpha\) which reduces the dynamic power. The maximum operating frequency obtained for the two estimators is about 264 MHz. The layout of for the gated and synchronous estimator is shown in figure 6. The design results in terms of logic gate numbers, silicon area and total power consumed by the two versions of estimator are presented in Table 3. The comparison of the silicon area and the power conception of the two versions of the estimator are presented in Table 4. This study shows that the gated clock motion estimator occupies a silicon area which is slightly over than the silicon area occupied by the synchronous estimator (3.76\% over). In contrast, the power consumed by the gated clock motion estimator is about 23.86\% times over than the power consumed by the synchronous estimator.

![Table 3. Design of the two versions of the estimator.](image)

<table>
<thead>
<tr>
<th>Estimator version</th>
<th>Logic gate number</th>
<th>Silicon area (mm^2)</th>
<th>power (mW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synchronous</td>
<td>13408</td>
<td>0.186</td>
<td>44.21</td>
</tr>
<tr>
<td>Gated clock</td>
<td>14472</td>
<td>0.193</td>
<td>33.66</td>
</tr>
</tbody>
</table>

![Table 4. Comparison in terms of silicon area and dissipated power](image)

<table>
<thead>
<tr>
<th>area</th>
<th>(100 - Gated clock area / Synchronous area) (\times 100%)</th>
<th>-3.76%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power</td>
<td>(100 - Gated clock power / Synchronous power) (\times 100%)</td>
<td>+23.86%</td>
</tr>
</tbody>
</table>

### 4. Comparison of the proposed gated clock architecture with other architecture

In this section we compare our gated clock motion estimator with other architecture. This comparison is presented in Table 6. An exact comparison is complicated by the fact that these architectures have been implemented with different technologies and exhibit variations in their specifications and capabilities. Nevertheless, it will be noted that the design presented exhibits the highest level of flexibility in terms of block sizes catered for. These results show also that our motion estimator outperform all other architectures in terms of gate count. In term of clock frequency only the Yap'04 architecture is higher than the frequency of our architecture.
Figure 6. Layout result of the gated clock motion estimator (a) and synchronous version (b)

Table 6. Results and comparison of motion estimators on ASIC devices

<table>
<thead>
<tr>
<th></th>
<th>[Vos 95]</th>
<th>[Huang 04]</th>
<th>[Shen 01]</th>
<th>[Yap 04]</th>
<th>[Kim 05]</th>
<th>This work</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PE. number</strong></td>
<td>16x16</td>
<td>Based GEA</td>
<td>64</td>
<td>16</td>
<td>16x16</td>
<td>16</td>
</tr>
<tr>
<td><strong>Search range</strong></td>
<td>16x16</td>
<td>16x16</td>
<td>16x16, 32x32</td>
<td>16x16, 32x32</td>
<td>32x32</td>
<td>30x30</td>
</tr>
<tr>
<td><strong>Process</strong></td>
<td>0.6um</td>
<td>0.35um</td>
<td>0.6um</td>
<td>0.13um</td>
<td>0.18um</td>
<td>0.13um</td>
</tr>
<tr>
<td><strong>Frequency</strong></td>
<td>72MHz</td>
<td>27.8Mhz</td>
<td>60MHz</td>
<td>294Mhz</td>
<td>100Mhz</td>
<td>264Mhz</td>
</tr>
<tr>
<td><strong>Gate count</strong></td>
<td>263k</td>
<td>33.2K</td>
<td>67k</td>
<td>61K</td>
<td>154k</td>
<td>14.4K</td>
</tr>
<tr>
<td><strong>Power</strong></td>
<td>--</td>
<td>189 mW</td>
<td>423 mW</td>
<td>--</td>
<td>--</td>
<td>33.66 mW</td>
</tr>
</tbody>
</table>

5. Conclusion

In order to reduce power dissipation in a video Codec, we have presented in this paper a hardware design of a gated clock FSBM motion estimator. The gated clock technique allows to shutting down each PE when it is not under running. The advantage of the proposed technique relatively to the existing techniques, like low bit rate, is that the power reduction is performed without the deterioration of the image quality. The synthesis study shows that the proposed architecture reduces 23.86% in terms of dissipated power relatively to purely synchronous motion estimation.

References


