Low-Power Implementation of Discrete Cosine Transform

Emad N. Farag and Mohamed I. Elmasry
VLSI Research Group
University of Waterloo,
Waterloo, Ontario, Canada N2L 3G1

Abstract

The demand for multimedia mobile terminals has created
a need for low power implementation of video compression
algorithms. In this paper we consider different implemen-
tations for the discrete cosine transform. The effect of pipel-
ing and parallelism on reducing the power dissipation is
considered for fast discrete cosine transform algorithms, as
well as ROM-based algorithms.

1. Introduction

The increase in demand for high throughput mobile
equipments whether portable computers or mobile termi-
nals, with limited improvement in battery technology, has
lead to increasing interest in low power systems. The chal-
lenge is to reduce the power without reducing the through-
put of the system.

Future mobile terminals are expected to handle video
as well as audio information. Due to the limited
bandwidth allocated to these terminals, video compres-
sion/decompression becomes an essential process that has
to be done in a mobile terminal.

Discrete cosine transform (DCT) is frequently used in
video compression [4]. In this paper, we consider three
different implementations for DCT. We also consider the
effect of pipelining and parallelism on reducing the power
dissipation of each implementation.

In CMOS systems, most of the power dissipation is dy-
namic switching power dissipation [1] which is proportional
to the square of the voltage. Hence, the most effective way
to reduce the power dissipation is by reducing the operating
voltage. However, reducing the operating voltage increases
the overall system delay and hence reduces the throughput.
To counteract this effect we use pipelining or parallelism.

2. Three Alternative Implementations

2.1. Multiplier implementation

While the direct implementation of an 8-point 1D-DCT
requires 64 multiplications and 56 additions, various algo-
rithms have been proposed that require a fewer number of
additions and multiplications. As an example, the algo-
rithm given in [3], [4] and shown in Fig. 1, requires only 29
ADD/SUB blocks and 13 multipliers.

2.2. Pure ROM implementation

The idea of this implementation is to replace the multipli-
cation operation with addition and a look up ROM table.
This process is known as distributed arithmetic [4], [5].

We can use the distributed arithmetic technique to imple-
ment the 8-point 1D-DCT. The block diagram for this
implementation is shown in Fig. 2. Eight 256-word 8-bit-per-word ROMs are required for this implementation.

3. Reducing power through pipelining and parallelism

3.1. Pipelining

As the voltage decreases (to decrease power dissipation) the overall delay of the data-path increases (an undesirable side effect). To counteract this increase in delay, we can use pipelining [2]. The rational of using pipelining here is that the increase in delay accompanying the decrease in voltage is balanced by dividing the data-path into smaller pipestages and keeping the maximum delay of any pipestage, at the lower voltage equal to the overall delay of the data-path without pipelining, at the higher voltage.

As a first order approximation, make the following assumptions:

- Neglect the effect of overhead caused by the pipeline registers, whether in terms of increased capacitance or increased delay.
- Assume that the pipeline can be perfectly balanced. All pipestages have the same delay.
- The delay of any stage is inversely proportional to the applied voltage.

To maintain the same maximum throughput when dividing the data-path into N pipestages, each pipestage can now operate at a voltage \( V/N \). The power dissipation in this case is given by:

\[
P_{pl} = \frac{P_s}{N^2}
\]

Where, \( P_s \) is the power dissipation before pipelining, and \( P_{pl} \) is the power dissipation after pipelining and reducing the voltage.

3.2. Parallelism

In the last section, we considered how to decrease the power through pipelining the data-path. In this section another way to decrease the power is considered, which is parallelism [2]. To compensate the increase in data-path delay as the voltage decreases, we replicate the data-path \( N \) times. The input samples are split among the \( N \) data-paths. The outputs of the \( N \) data-paths are then multiplexed onto a single output stream. This allows the system to maintain its throughput, while each data-path is operating at a lower rate (lower voltage).

Unlike pipelining, parallelism greatly increases the area. Making approximations similar to those made in the analysis of pipelining, we can show that the power dissipation of a
system consisting of N parallel data-paths and having the same throughput rate is given by,

\[ P_{PR} = \frac{P_s}{N^2} \]  

(2)

Where, \( P_s \) is the power dissipation for a single data-path system, and \( P_{PR} \) is the power dissipation for a system consisting of N parallel data-paths and having the same maximum throughput as the single data-path system.

4. Evaluating the performance of the multiplier implementation

Table 1 gives the overall delay and power dissipation of the multiplier implementation of Fig. 1 in 0.8\( \mu \)m BiCMOS technology, power dissipation is evaluated at a frequency 5 MHz. Notice that, the power dissipation is reduced as the voltage decreases but this is at the expense of the increased delay (decreased throughput). To maintain the same throughput rate at the lower voltage we use pipelining or parallelism.

<table>
<thead>
<tr>
<th>Voltage</th>
<th>Delay</th>
<th>Power</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>101.56 ns</td>
<td>13.49 mW</td>
</tr>
<tr>
<td>4</td>
<td>132.27 ns</td>
<td>7.64 mW</td>
</tr>
<tr>
<td>3.3</td>
<td>162.53 ns</td>
<td>4.99 mW</td>
</tr>
</tbody>
</table>

Table 1. Delay and power dissipation at 5 MHz for the multiplier implementation with no pipelining

Fig. 4 shows the theoretical relation between the power dissipation and the number of pipestages (equation 1). Also shown on the same curve are the results obtained from pipelining the multiplier implementation. Everything is normalized to the case of a single stage system operating at the same rate (throughput).

![Figure 4. The effect of pipelining on reducing the power dissipation](image)

It is also possible to maintain the same throughput while reducing the voltage by parallelism. Figure 5 shows the theoretical relation between the power dissipation and the number of data-paths (equation 2). Also shown on the same curve are the results obtained from increasing the parallelism in the multiplier implementation. Everything is normalized to the case of a single path system operating at the same rate (throughput).

![Figure 5. The effect of parallelism on reducing the power dissipation](image)

5. Evaluating the performance of the ROM implementations

Two different types of ROM implementations were considered. The pure ROM implementation, which requires eight 256-word 8-bit-per-word ROMs. The overall delay and power dissipation for this implementation is given in table 2. All eight ROMs have the same address, hence use the same ROM decoder.

<table>
<thead>
<tr>
<th>Voltage</th>
<th>Delay</th>
<th>Power</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>86.5 ns</td>
<td>30.4 mW</td>
</tr>
<tr>
<td>4</td>
<td>111.4 ns</td>
<td>18.3 mW</td>
</tr>
<tr>
<td>3.3</td>
<td>137.1 ns</td>
<td>10.5 mW</td>
</tr>
</tbody>
</table>

Table 2. Total delay and power dissipation at 5 MHz for the pure ROM implementation

The second type of ROM implementation considered is
the mixed ROM implementation. This implementation requires eight 16-word 8-bit-per-word ROMs. But it requires eight extra adder/subtractor units. The overall delay and power dissipation for this implementation is given in Table 3. The eight ROMs can be divided into two groups, the ROMs of each group have the same address, hence two ROM decoders are required.

Three architectural alternatives, in addition to the single stage alternative, were considered for each of the pure and mixed ROM implementations:

- Two stage pipeline
- Two stage pipeline with two parallel adders per path, Fig.6.
- Two stage pipeline with three parallel adders per path, Fig.7.

![Figure 6. Using two parallel adders](image1)

Table 4. Reducing power dissipation by pipelining and parallelism in the pure ROM implementation

<table>
<thead>
<tr>
<th>Delay(ns)</th>
<th>Implementation</th>
<th>Power (mW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>85 – 90</td>
<td>Single stage</td>
<td>30.4 (@ 5V)</td>
</tr>
<tr>
<td></td>
<td>2 stage 1 add/path</td>
<td>21.9 (@ 4V)</td>
</tr>
<tr>
<td>65 – 70</td>
<td>2 stage 1 add/path</td>
<td>36.1 (@ 5V)</td>
</tr>
<tr>
<td></td>
<td>2 stage 2 add/path</td>
<td>14.5 (@ 3.3V)</td>
</tr>
<tr>
<td>52 – 57</td>
<td>2 stage 2 add/path</td>
<td>24.6 (@ 4V)</td>
</tr>
<tr>
<td></td>
<td>2 stage 3 add/path</td>
<td>14.5 (@ 3.3V)</td>
</tr>
<tr>
<td>42 – 46</td>
<td>2 stage 2 add/path</td>
<td>40.6 (@ 5V)</td>
</tr>
<tr>
<td></td>
<td>2 stage 3 add/path</td>
<td>24.9 (@ 4V)</td>
</tr>
</tbody>
</table>

![Figure 7. Using three parallel adders](image2)

Table 5. Reducing power dissipation by pipelining and parallelism in the mixed ROM implementation

<table>
<thead>
<tr>
<th>Delay(ns)</th>
<th>Implementation</th>
<th>Power (mW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>65 – 70</td>
<td>2 stage 1 add/path</td>
<td>35.7 (@ 5V)</td>
</tr>
<tr>
<td></td>
<td>2 stage 2 add/path</td>
<td>15.0 (@ 3.3V)</td>
</tr>
<tr>
<td>50 – 55</td>
<td>2 stage 2 add/path</td>
<td>24.4 (@ 4V)</td>
</tr>
<tr>
<td></td>
<td>2 stage 3 add/path</td>
<td>16.0 (@ 3.3V)</td>
</tr>
<tr>
<td>41 – 43</td>
<td>2 stage 2 add/path</td>
<td>40.2 (@ 5V)</td>
</tr>
<tr>
<td></td>
<td>2 stage 3 add/path</td>
<td>25.8 (@ 4V)</td>
</tr>
</tbody>
</table>

Pipelining and parallelism were used to lower the power dissipation in the three implementations by lowering the voltage and keeping the throughput constant. Pipelining and parallelism can be combined together to overcome the unbalanced pipelastages delay problem. This technique was used in the ROM implementations to balance the pipelastages delays.

6. Conclusions

Three different implementations for the DCT were considered. In terms of power dissipation, the multiplier implementation has the lowest power dissipation, while the pure ROM and the mixed ROM implementations have higher power dissipations. In terms of speed, the multiplier implementation is the slowest and the mixed ROM implementation is the fastest.

References