As portable handheld communication devices proliferate, vendors are adding more features, such as digital video and audio, to meet consumer demand. The ISO MPEG-4 standard is emerging as the predominant video and audio format for these applications. Based on standard 16-bit telecom DSP's, these devices must perform 8-bit video and 16-bit telecommunication operations to support these applications. The low-power requirements of portable devices mean that these computations must be handled with maximum efficiency.

Dynamically reconfigurable DSPs achieve the high efficiency required via cycle-to-cycle reconfiguration while retaining flexibility and programmability. Portable handheld devices, such as video cell phones, need a simpler but still computationally intense implementation of the MPEG-4 standard. The MPEG-4 simple profile at level 1 is ideal for portable applications with a Quarter Common Intermediate Format (QCIF—176 x 144 pixels), 10-15 frames per second (fps) and 64 kbps data rate.

The MPEG-4 encoder is the most computationally costly part of the codec. Motion estimation consumes 66-94% of the cycles required for the encode operation, depending on the estimation method. Efficient motion estimation is essential for creating an efficient implementation. While custom hardware can be designed for estimation, it limits the flexibility to grow as motion estimation algorithms and digital video standards evolve.

Since an MPEG-4 encoder is not explicitly specified by the MPEG-4 standard, it need not conform to a specific profile and level, as is the case for a decoder. A simple profile at level 1 decoder must be able to support up to four video objects (VOPs); however, a bitstream encoded at this level may contain only one object. The system designer determines the complexity of the encoded bitstream and therefore the complexity of the encoder. The simple profile at level 0 or level 1 is currently the most suitable for portable handheld applications, with the only difference between the two being the maximum number of VOPs the encoder supports. To reduce the complexity of the encoder, the entire QCIF frame can be encoded as a single object, significantly reducing the computational requirements by eliminating the need for segmentation and shape coding, while the bitstream remains valid for both the simple profile at level 0 and at level 1. In addition, VOP selection and segmentation typically involves human input, which is not practical for portable devices such as cell phones.

Figure 1 shows a simple block diagram for a fully programmable MPEG-4 encoder and the
algorithms required for encoding a simple and single VOP MPEG-4 video stream. The MPEG-4 encoder contains a model of a decoder that is used in conjunction with the motion estimator. This model ensures that the motion estimation is carried out with the same reference frame (previous frame) that the decoder will use to achieve the best possible results.

Initial estimates suggest that the full-search block-matching motion-estimation algorithm requires as many as 800 Mcycles on a typical dual-execution-unit DSP. Ten percent of this figure would be unacceptable for a portable handheld application, so fast motion-estimation algorithms with reduced computational complexity are a necessity. There are a large number of fast motion-estimation algorithms available; however, the four-step search (4SS) algorithm is the best for small frame portable, programmable applications.

Finally, we use the Carmel DSP, a typical dual-execution-unit, 16-bit "telecom" DSP, to illustrate the algorithms employed and the impact of reconfigurability. This DSP is similar to many competitive DSPs in this segment, such as the Texas Instruments C55x and C62x DSP series, with a long instruction word format that enables up to six operations per cycle. All of these DSPs can be configured with tightly coupled co-processors that can implement special instructions on a cycle-t-cycle basis, illustrating the power of dynamically reconfiguring the available execution units and datapath.

4SS Motion Estimation
The four-step search was developed from the popular three-step search algorithm in order to improve efficiency, especially for small motion vectors. The popularity of the three-step search derives from its simplicity and effectiveness, and experimental results show that it provides near optimal performance. For video-telephony applications, the efficient performance for small-motion vectors of the four-step search is advantageous, since it is expected that the majority of encoded video streams will comprise a face with simple motion. The four-step search algorithm still performs very well for more complicated motion with small frame sizes such as QCIF, where the relatively small search area, (-7, +7 pixels) does not hinder performance. An additional bonus is that the small search area of the four-step search algorithm means that you can represent the resultant motion vectors using 8-bits, creating more easily handled data.

The 4SS algorithm uses a center-biased search pattern, with nine checking points on a 5x5 window in the first step. The center of the search window is then shifted to the point with minimum block-distortion measure. The search window of the next two steps depends on the location of the minimum BDM points. The 4SS is summarized as follows:

**Step 1:** A minimum BDM point is found from a nine-checking-points pattern on a 5 x 5 window located at the center of the 15 x 15 search area (Figure 2a).

**Decision:** If the minimum BDM point is found at the center of the search window, go to Step 4 otherwise go to Step 2.
Step 2: The search-window size is maintained at 5 x 5. However, the search pattern will depend on the position of the previous minimum BDM point.

a. If the previous minimum BDM point is located at the corner of the search window, five additional checking points (Figure 2b) are used.

b. If the previous minimum BDM point is located at the middle of the horizontal or vertical axis of the previous search window, three additional checking points (Figure 2c) are used.

Decision: If the minimum BDM point is found at the center of the search window, go to Step 4; otherwise go to Step 3.

Step 3: The searching pattern strategy is the same as Step 2, but will now proceed to Step 4.

Step 4: The search window is reduced to 3 x 3 (Figure 2d) and the direction of the overall motion vector is considered as the minimum BDM point among these nine searching points.

Figure 2 illustrates the four possible search types utilized by the four-step-search motion-estimation algorithm. Figure 2a shows the initial nine checking point search on the 5 x 5 search window. Figure 2b shows a five-point corner-based search, where the dark pixels represent the five new search positions and the light pixels depict previously searched positions. Figure 2c shows a three-point side-based search scheme, and Figure 2d illustrates the final eight-point center-based search on a 3 x 3 window.

Block-matching motion-estimation algorithms select the appropriate motion vector by minimizing a cost function, the block distortion measure (BDM). The BDM is generally based on one of the many available distance criteria such as the sum of absolute difference (SAD) or mean absolute error (MAE). These distance criteria, which differ by a division operation, are the most widely used due to their low complexity and superior results. The SAD is a better BDM for implementation on a fixed-point DSP since it omits the division operation required for the MAE and is therefore more efficiently calculated.

The maximum number of search positions is 27; however, the intermediate steps of the algorithm may be skipped with a jump to the final step if at any time the minimum BDM point is located at the center of the search window. The minimum possible number of checking-points for the four-step search algorithm is 17, with experimental results indicating an average of 19.

Efficient 4SS Implementation
The four-step-search motion-estimation algorithm is best implemented in assembly language by breaking the entire algorithm into sub-functions, facilitating efficient program memory use. Table 1
shows these sub-functions along with a description of their tasks and the associated cycle counts. The best- and worst-case figures stem from the variation in the computational requirements of the Valid_Search() function. This function requires 16 cycles when the current search position is described by the (0,0) motion vector, as it is a valid search; however, non-trivial search positions require a more thorough checking procedure. The computational cost of the entire algorithm for a typical 16-bit, dual-execution-unit DSP can be derived from the results in Table 1. The best case represents the minimum 17 search positions with the best realizable results from Valid_Search(), while the worst case consists of the maximum possible 27 search positions and 59 cycles consumed for each search validation step. The algorithm must be executed for each luminance and chrominance macroblock in the frame, at 10 frames per second. **Equations 1** and **2** show the method for calculating the overall best- and worst-case cycle counts:

Best Case: Cycle cost = 9 x (11x (Initial + Centre + Resid_Calc + 20) + 3) x 1.5 x 10

Worst Case: Cycle cost = 9 x (11x (Initial + 2 x Corner + Centre + Resid_Calc + 42) + 3) x 1.5 x 10

Evaluating **Equations 1** and **2** for the overall cycle cost for a 10 fps session, the overall best-case result is 6.2 Mcycles, while the worst-case result is 10.5 Mcycles. These figures are much more practical for implementation as a software implementation for a portable device than the full-search algorithm. As a result, an overall programmable MPEG-4 encoder could now be implemented in 20-30 Mcycles of computation, which is a more realistic value for portable devices with multiple applications.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Description</th>
<th>Calculation</th>
<th>Best Case (cycles)</th>
<th>Worst Case (cycles)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Valid_Search</td>
<td>Checks each candidate search position and determines whether it is within the frame boundary and therefore valid</td>
<td></td>
<td>16</td>
<td>59</td>
</tr>
<tr>
<td>SAD_Calculation</td>
<td>Calculates the SAD value for the two macroblocks under consideration</td>
<td></td>
<td>158</td>
<td>158</td>
</tr>
<tr>
<td>Resid_Calculation</td>
<td>Calculates the prediction error for each pixel of each macroblock and stores back to memory.</td>
<td></td>
<td>294</td>
<td>294</td>
</tr>
</tbody>
</table>
Implementing the motion estimation on a 16-bit, dynamically reconfigurable DSP may reduce the cycle cost even further by enabling the calculations performed on 8-bit pixel values to be carried out in a highly parallel manner. Specifically, you can use tightly coupled co-processors that can be dynamically employed on a cycle-to-cycle basis to perform multiple, parallel 8-bit operations using 16-bit input words packed as 2 x 8-bit pixel values. These co-processors accelerate the algorithm by working on each 8-bit section of the input words and repacking the result into a 16-bit word.

### Reconfigurable DSPs and Efficient Motion Estimation

There are three main characteristics that make a DSP dynamically reconfigurable while retaining flexibility. Tightly coupled co-processors enhance and expand the available instruction set and datapath on a cycle-to-cycle basis by adding application-specific execution units to the DSP core. A flexible instruction set allows the size of instructions to be tailored to a specific application while minimizing program memory use, and flexible memory-access rules allow maximum use to be made of available memory bandwidth.

The calculation of the SAD for each comparison consumes approximately 70% of the best-case cycles and 60-65% of the worst-case cycles, and consists predominantly of accumulated absolute-distance operations on 8-bit numbers. You can significantly improve the calculation of this 8-bit, non-MAC function with a dynamically reconfigurable processor. The following code example shows the kernel of the SAD calculation function implemented without dynamic reconfigurability on the Carmel 10xx. This kernel consists of two nested loops that process the 256 pixels in a MPEG-4 macro-block two at a time for each macro-block in the video frame. The configurable long instruction word (CLIW) command is the Carmel DSP's means of executing very long instruction word (VLIW) commands.

```c
rep (N) block {
  rep (N/2) single // 256 pixels per MacroBlk processed 2 at a time {
    cliw _sad(r0++, r0++, r4++, r4++)
  }
  &
  a0l = *ma1 - *ma3 || a2 += abs(a0l) || // |diff| stored from
  a1l = *ma2 - *ma4 || a3 += abs(a1l); // last cycle
```
Using a video-accelerating, tightly coupled co-processor with a custom quad-accumulated absolute-distance instruction and memory-packing feature, four pixel values can be processed in a single cycle. This approach results in a significant increase in performance for the entire algorithm. The next coded example shows the kernel of the SAD calculation function implemented by dynamically reconfiguring the Carmel DSP as a highly parallel 8-bit processor, where the "plug" statement represents a call to the co-processor's specialized instruction. Table 3 shows the sub-functions making up the motion estimation algorithm and the associated cycle cost when implemented with a dynamically reconfigurable DSP and compared to the results for a non-reconfigurable processor. The reduction in cycle cost of SAD_Calculation() is reflected in the reduced cost of the search sub-functions. Evaluating Equations 1 and 2 for this implementation of the four-step-search motion-estimation algorithms, the overall best-case cycle cost is 4.3 Mcycles, while the worst-case cost is 7.2 Mcycles. The overall reduction achieved using the tightly coupled co-processors to dynamically reconfigure the data path of the DSP is 31% for both the best- and worst-case scenarios, where the best-case result has been reduced to 4.3 Mcycles from 6.2 Mcycles.

```plaintext
rep (N) block {
    rep (N/4) single               // 256 pixels per MacroBlk processed 4 at a time
    {
        cliw _20xx_sad(r0++, r0++, r4++, r4++)
        {
            plug1 pa0 += abs(*ma1h-*ma3h)+abs(*ma1l-*ma3l)+abs(*ma-
h-*ma4h)+abs(*ma2l-*ma4l);
            // 4x acc abs dist
        }
    }
    r0 == r0+rn0 || r4 == r4+rn4; // Finished one line - index to next
}
```

### Table 3: Cycle costs of 4SS motion estimation sub-functions on a dynamically reconfigurable processor and a non-reconfigurable processor

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Calculation</th>
<th>Carmel 20xx</th>
<th></th>
<th>Carmel 10xx</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Rest Case (cycles)</td>
<td>Worst Case (cycles)</td>
<td>Rest Case (cycles)</td>
<td>Worst Case (cycles)</td>
</tr>
<tr>
<td>Valid_Search</td>
<td><code>*</code></td>
<td>16</td>
<td>59</td>
<td>16</td>
<td>59</td>
</tr>
<tr>
<td>SAD_Calculation</td>
<td><code>-</code></td>
<td>92</td>
<td>92</td>
<td>158</td>
<td>158</td>
</tr>
<tr>
<td>Resid_Calculation</td>
<td><code>-</code></td>
<td>150</td>
<td>150</td>
<td>294</td>
<td>294</td>
</tr>
<tr>
<td>Initial</td>
<td><code>9x (Valid_search + SAD_Calc) + 245</code></td>
<td>1412</td>
<td>1604</td>
<td>2022</td>
<td>2198</td>
</tr>
<tr>
<td>Corner</td>
<td><code>5x (Valid_search + SAD_Calc) + 141</code></td>
<td>763</td>
<td>896</td>
<td>1136</td>
<td>1269</td>
</tr>
<tr>
<td>Side</td>
<td><code>3x (Valid_search + SAD_Calc) + 120</code></td>
<td>474</td>
<td>573</td>
<td>676</td>
<td>775</td>
</tr>
<tr>
<td>Center</td>
<td><code>6x (Valid_search + SAD_Calc) + 229</code></td>
<td>1304</td>
<td>1437</td>
<td>1655</td>
<td>1993</td>
</tr>
</tbody>
</table>

A final factor to consider when implementing the encoder is memory requirements. Three QCIF frames must be buffered for the motion-estimation algorithm: the reference frame, the current frame, and the frame of residuals. If each pixel value occupies a 16-bit memory word on the non-reconfigurable processor, 114k words are required. Using the memory-packing feature of the video accelerator, this value can be reduced 33% to 76k words as the 8-bit pixel values of the reference and current frames may be packed into 16-bit locations. The residuals, however, must be represented by 9-bit words, since they result from the subtraction of 8-bit values and therefore cannot be similarly packed.

**Conclusions**

Motion estimation comprises the single most memory and computationally costly algorithm in an MPEG-4 video codec. The 4SS efficient motion estimation algorithm is seen to have significantly less computational cost than a full-search method, making it suitable for fully programmable, portable video applications. Using a dynamically reconfigurable DSP, you can realize cycle savings of 31% and memory savings of 33% over a non-reconfigurable processor. These savings are important in implementing the encoder and decoder within an allocated cycle budget and are a direct result of reconfiguring the DSP as a highly parallel 8-bit processor.