FPGA Implementation of H.264 Video Encoder

- BY KUSHAL KUNIGAL UNDER GUIDANCE OF

DR. K.R.RAO.

SPRING 2011,

ELECTRICAL ENGINEERING DEPARTMENT,
UNIVERSITY OF TEXAS AT ARLINGTON
Proposal

- This project is based on the implementation of H.264 Video encoder and the Algorithms for evaluating the Transform and quantization. Suitable for high speed implementation on FPGA/ASIC.
Modified Encoder Hardware Design:

Fig 1: Modified encoder hardware design [3].
Description of Test Data

The encoder will be used to encode a frame from a video sequence:

- 30 frames/second.
- Each frame is 176 x 144 pixels (QCIF Resolution, very typical for low bit-rate video contents in cell phones).
Description of Test Data

Reference Frame

Fig 2: Reference frame 1 [4].
Description of Test Data

Current Frame

Fig 3: Current frame 2 [4].
Fig 4: Residual of current frame [4].
Different Stages in designing H.264 Encoder

- Motion Estimation
- Motion Compensation
- Discrete Cosine Transform
- Quantization
Fig 5: The principle of block matching motion estimation algorithm is finding the best matching block in the searching area of a reference frame for each macroblock in the original frame.
Step 1: A macro block that is the “best match” is found from a nine-checking-points pattern on a 5x5 window located at the center of the 15 x 15 search area (Figure 5a).

Decision: If the “best matched” macro block is found at the center of the search window, go to Step 3 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 “best matched” macro block location.

a. If the previous “best matched” macro block is located at the corner of the search window, five additional checking points (Figure 5b) are used.
b. If the previous “best matched” macro block is located at the middle of the horizontal or vertical axis of the previous search window, three additional checking points (Figure 5c) are used.

- **Decision**: If the “best matched” macro block is found at the center of the search window, go to Step 3; otherwise go to Step 2.

- **Step 3,4**: The search window is reduced to 3 x 3 (Figure 5d) and the direction of the overall motion vector is considered as the “best matched” macro block is among these nine macro blocks.
4-Step Search

Check points for motion estimation

4x4 macroblock

A 20x20 pixel search range divided up by 4x4 pixel search

Fig 6a: A 20x20 pixel search range divided up by 4x4 pixel search [5].
4-step search

Step 2

Fig 6b: A 20x20 pixel search range with 5 additional checkpoints during step 2 of the four-step search algorithm [5].
4-step search

Step 3

Fig 6c: A 20x20 pixel search range with 3 additional checkpoints during step 3 of the four-step search algorithm [5].
4-step search

Step 4

Fig 6d: A 20x20 pixel search range with 5x5 search window, now reduced to 3x3 during step 4 of the four-step search algorithm [5].
Motion Vector Description

Fig 7. Motion vector description.
for $i =$ \# of MBs that can be fitted in one direction 
for $j =$ \# of MBs that can be fitted in another direction 
predicted_frame$(i, j) =$ RF$(i, j) +$ MV for that MB; 
end 
end [5]

Fig 8: Pseudo code for obtaining predicted frame [6].
Fig 9: Overall view of operations in encoder module [6].
Motion Compensation

Fig 10: Generating predicted frame using reference frame and motion vectors. [7].
Figure 11: The residual frame formed by subtracting the predicted frame from the actual current frame.
Discrete cosine transform

\[ Y = C_f X C_f^T \otimes E_f \]

Figure 12: Real DCT factor.

Where:
- \( X \) denotes the original macroblock
- \( a = 0.5 \)
- \( b = \sqrt{2/5} \)
- denotes array multiplication
Results

Platform:

ModelSim: Version XE III 6.0d.
Quartus II: Version 10.1 -sp1 Web Edition (64-Bit).

The hardware description language used was VHDL (Very high speed integrated circuit Hardware Description language). Figure 14 shows the list of code segments used.
Figure 13: List of code segments used to design the H.264 Video encoder.
List of tasks

Figure 14: List of tasks performed.
### Resource usage summary

<table>
<thead>
<tr>
<th>Resource</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Total logic elements</td>
</tr>
<tr>
<td>2</td>
<td>Combinational with no register</td>
</tr>
<tr>
<td>3</td>
<td>Register only</td>
</tr>
<tr>
<td>4</td>
<td>Combinational with a register</td>
</tr>
<tr>
<td>5</td>
<td>Logic element usage by number of LUT inputs</td>
</tr>
<tr>
<td>6</td>
<td>4 input functions</td>
</tr>
<tr>
<td>7</td>
<td>3 input functions</td>
</tr>
<tr>
<td>8</td>
<td>&lt;=2 input functions</td>
</tr>
<tr>
<td>9</td>
<td>Register only</td>
</tr>
<tr>
<td>10</td>
<td>Logic elements by mode</td>
</tr>
<tr>
<td>11</td>
<td>Normal mode</td>
</tr>
<tr>
<td>12</td>
<td>Arithmetic mode</td>
</tr>
<tr>
<td>13</td>
<td>Total registers*</td>
</tr>
</tbody>
</table>

Figure 15: Resource usage summary.
Figure 16: generated register transfer logic for the H.264 video encoder.
The FPGA implementation algorithms along with the complexities involved in the mapping of H.264 video encoder transforms onto the hardware were discussed.

A complete analysis of the motion estimation algorithm, motion compensation, DCT and quantization was made. VHDL code was written to implement the H.264 video encoder. The code was compiled and synthesized using ModelSim and Quartus ii softwares. The Register transfer schematic for the code was generated.


Websites and References


Websites and references


THANK YOU!!!!