VLSI Designs for High-Speed Huffman Decoder

Shih-Fu Chang and David G. Messerschmitt

Dept. of Electrical Engineering and Computer Science
University of California, Berkeley, CA94720

Abstract

Many video compression systems require a high-speed implementation of the Huffman decoder. The recursive iteration of the decoding process limits the achievable decoding throughput with a given IC technology. In this paper, we designed two classes of VLSI architectures for high speed implementation, the tree-based architectures and the Programmable Logic Array (PLA) -based architectures. We also constructed a variable-length-code based on a popular video compression system and compared the pros and cons of each architecture. We laid out and simulated the major parts of a pipelined constant-input-rate PLA-based architecture using a high-level synthesis approach. We claim that the decoding throughput of 200 Mbps is achievable with CMOS 2.0 \( \mu \)m technology.

1 Introduction

Video compression systems such as high-definition televisions (HDTV) require a high-speed implementation of the Huffman decoding algorithm [1,2,3]. The recursive iteration of the decoding process limits the achievable throughput of the Huffman decoder in a given IC technology. Previous implementations can not achieve satisfactory decoding throughput [3,4]. In this project, we aim our goal at a high-level synthesis approach, a single chip implementation, and a decoding speed of 200Mbps, which can meet requirements of most video compression systems. We will achieve this goal by exploring two classes of parallel VLSI architectures, tree-based architectures and PLA-based architectures.

The procedure of using the Huffman tree to construct a Huffman code is shown in figure 1. Longer codewords are assigned to the more probable source symbols and shorter codewords to the less frequent source symbols. The encoding process of the Huffman code is just a table lookup with fixed-length inputs. For the decoder, the encoded bit stream has to be decoded sequentially bit by bit since it doesn’t contain the periodical boundary informations. Each input bit will route an imaginary token to traverse down the Huffman tree by one level until a terminal node is reached. The conventional Huffman decoder operates in this way and uses a Read Only Memory (ROM) to implement the Finite State Machine (FSM) in the decoding process. It decodes one bit per cycle synchronously. With the memory products available today, the throughput is still far behind the required speed.

To compare performance of different types of architectures, we construct a Huffman code based on the output statistics of a popular video compression algorithm, Discrete Cosine Transform (DCT) [2]. The resulting codeword length varies from 2 to 16 bits and the average codeword length (\( L_{avg} \)) turns out to be 2.85 bits per codeword.

2 Tree-Based Architectures

A natural way of mapping the tree-traversing decoding process to hardware is to model the branching function at each internal node as a 1-to-2 demultiplexer and use storage cells to store the source symbols associated with terminal nodes. As shown in figure 2(a), the resulting architecture is highly regular and dedicated.
Mukherjee et al. proposed a pipelined tree-based architecture for the Huffman encoder [5], but not the decoder, which is more complicated due to the sequential dependence property. To apply the pipeline technique here, we partition the whole decoder into pipeline stages, each of which includes one level of the Huffman tree. For example, in Figure 2(b), we use a single ROM to substitute all the branching functions and the symbol storages in each level. Thus, the whole decoder can be implemented by simply cascading several ROM's.

We can use the above pipelined architecture to decode multiple independent bit streams in parallel. In Figure 2(b), parallel bit streams are delayed by different amounts of time in a way analogous to that of skew pipelining. The rotation register is employed to schedule \( L_{\text{max}} \) (i.e. the maximal codeword length) input streams to \( L_{\text{max}} \) pipeline stages. When the codeword length is less than the maximal codeword length, the traversing process will be terminated in some intermediate stage where the decoded symbol will be output and subsequent stages will not be utilized by this stream. Also, the data fetch from the input buffer for this stream should be temporarily frozen. The freeze state will not be removed until the beginning cycle of the next codeword in this stream.

The bus used to transfer the decoded symbols from the storage unit in each pipeline stage to the output is also pipelined to ensure the correct timing relationship. Therefore, the output sequence on the bus is in a multiplexed form including the decoded codewords from the \( L_{\text{max}} \) independent encoded bit streams.

If each stream is of infinite length, the throughput is always as fast as one codeword per cycle. The most important advantage of this architecture is that the critical path is very short. It only includes the delay time of a small ROM or 1-to-2 demultiplexer, plus a single latch delay. Simulation results based on standard cell implementation of the demultiplexers gives a 10 ns estimation with 2 \( \mu \)m CMOS technology. The equivalent decoder throughput is 100 M codewords per second. Thus, this architecture is suitable for high-clock-rate, low-level optimizations and custom designs. Note that the above speedup is independent of the source alphabet size or the maximal codeword length. However, these two parameters will affect the circuit complexity.

If the length of each bit stream is finite and constant, like those generated by source segmentation [6], then the pipeline speedup will be decreased to some extent. Some blocks may run out of data while other blocks still have some to be decoded. This will cause some empty slots on the output bus and thus reduce the throughput. However, we can easily prove that the average throughput is still between \( L_{\text{min}} \) and \( L_{\text{max}} \) bits per cycle.

3 PLA-Based Architectures

We have designed 3 different concurrent architectures using the PLA technology to implement the FSM required in the Huffman decoding process. The constant-input-rate architecture, as shown in Figure 3, always decodes \( K \) bits per cycle. It produces variable number of codewords in every cycle. The input buffer sends \( K \) bits per cycle to the decoder PLA, where the table lookup process is undertaken. The PLA will decode all the codewords contained in the current block of input bits.
The constant-output-rate architecture, however, produces a fixed-rate output, one codeword per cycle, as shown in figure 4 with K=1. A similar architecture has been proposed by Sun et al. [3]. The operation is as followed. Initially, the input buffer transfers two words into two separate latches, one for the lower order and another one for the higher order. They form a 32-bit window where the PLA takes 16 bits as its input to decode one codeword. Since 16 bits is the maximal codeword length, it is guaranteed that at least one codeword can be decoded. The output of the PLA also includes the actual length of the decoded codeword. The length is accumulated and used to memorize the new starting position within the 32-bit window in order to get the next 16 bits for decoding. The barrel shifter controlled by the accumulator is used to select the 16 bits from the 32-bit window. In addition, the carry bit of the accumulator is used to issue a data fetch command and get the next word from the input buffer.

Usually, there are more than one codeword contained in one block of 16 bits. In the variable-I/O-rate architecture, as shown in figure 4 with K>1, we extend the above constant-output-rate architecture to decode K codewords per cycle when possible. The output field of the PLA is also expanded to afford the width of K decoded symbols and one additional field indicating the actual number of symbols decoded.

### 3.1 High-level PLA Optimization Techniques

The input width, output width, and the number of product terms all together determine the complexity of a PLA. Table 1 shows the complexity of 3 different types of architectures mentioned above. Redundant terms can be removed by using the logic minimization technique. The resulting minimized complexity is shown in the 4th column.

Another optimization technique is called PLA decomposition [7], which splits a single PLA truth table into several smaller ones of similar size and then optimize each of them locally. The decomposed PLA's operate in parallel and the overall function correctness is accomplished by adding a stage of OR gates after the PLA's. We found this technique to be very effective in reducing the clock cycle time.

To enhance the performance, the pipeline technique can also be applied to all PLA-based architectures described above. Instead of decoding all the PLA outputs at once, we can separate the PLA outputs to several stages. For example, we can decode the next state output in the first stage and decode the codewords contained in the current block in the second stage. Figure 5 shows an example using the constant-input-rate design with 8 bits

---

**Table 1. Complexity and throughput of PLA-based architectures**

<table>
<thead>
<tr>
<th>constant input rate</th>
<th>input #</th>
<th>output #</th>
<th>product #</th>
<th>minimized product #</th>
<th>throughput bits/cycle</th>
</tr>
</thead>
<tbody>
<tr>
<td>K=1</td>
<td>9</td>
<td>17</td>
<td>512</td>
<td>235</td>
<td>1</td>
</tr>
<tr>
<td>K=2</td>
<td>10</td>
<td>17</td>
<td>1024</td>
<td>406</td>
<td>2</td>
</tr>
<tr>
<td>K=3</td>
<td>11</td>
<td>26</td>
<td>2048</td>
<td>612</td>
<td>3</td>
</tr>
<tr>
<td>K=4</td>
<td>12</td>
<td>26</td>
<td>4096</td>
<td>901</td>
<td>4</td>
</tr>
<tr>
<td>K=5</td>
<td>13</td>
<td>34</td>
<td>8192</td>
<td>1239</td>
<td>5</td>
</tr>
<tr>
<td>K=6</td>
<td>14</td>
<td>34</td>
<td>16384</td>
<td>2048</td>
<td>6</td>
</tr>
<tr>
<td>K=7</td>
<td>15</td>
<td>43</td>
<td>32768</td>
<td>3130</td>
<td>7</td>
</tr>
<tr>
<td>K=8</td>
<td>16</td>
<td>43</td>
<td>65536</td>
<td>4665</td>
<td>8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>constant output rate</th>
<th>input #</th>
<th>output #</th>
<th>product #</th>
<th>minimized product #</th>
<th>throughput codeword /cycle</th>
</tr>
</thead>
<tbody>
<tr>
<td>variable I/O rate</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>K=2</td>
<td>16</td>
<td>21</td>
<td>1708</td>
<td>580</td>
<td>&lt;2</td>
</tr>
<tr>
<td>K=3</td>
<td>16</td>
<td>30</td>
<td>4789</td>
<td>1479</td>
<td>&lt;3</td>
</tr>
<tr>
<td>K=4</td>
<td>16</td>
<td>38</td>
<td>12197</td>
<td>479</td>
<td>&lt;4</td>
</tr>
<tr>
<td>K=5</td>
<td>16</td>
<td>47</td>
<td>24709</td>
<td>137</td>
<td>&lt;5</td>
</tr>
</tbody>
</table>

---

**Figure 3.** The constant-input-rate PLA-based architecture for the Huffman decoder.

**Figure 4.** The variable-I/O-rate PLA-based architecture for the Huffman decoder. It can decode up to K codewords per cycle. The constant-output-rate architecture is a special case with K=1.
decoded per cycle. The complexity of the PLA's in each stage is greatly reduced and since there is no feedback left in the second stage, the PLA in the second stage can be further pipelined.

![Diagram of a pipelined constant-input-rate PLA-based architecture for the Huffman decoder. The throughput is 8 bits per cycle.](image)

**Figure 5.** A pipelined constant-input-rate PLA-based architecture for the Huffman decoder. The throughput is 8 bits per cycle.

### 3.2 Performance Analyses and Implementations

The net information rate that the decoder can decode equals the number of bits or codewords decoded per cycle multiplied by the clock rate. There is a trade-off relationship between these two terms. The more bits or codewords we try to decode in one cycle, the more complicated the PLA will become and thus slow down the clock rate. For the constant-input-rate architectures, the throughput is K bits per cycle. For the constant-output-rate architectures and variable-I/O-rate architectures, it's about K codewords per cycle. These two throughput formats can be compared by noting that the average codeword length is 2.85 bits per codeword in our video compression experiments. However, the exact throughput (bits/sec or codewords/sec) still depends on the actual clock rate. As we described before, we can use some optimization techniques to greatly reduce the PLA complexity and the clock cycle time. With advanced manufacturing technologies, like 1.2 μm CMOS technology, and a high-speed clock rate in the order of 100 MHz, the decoder throughput of 1 Gbps is possible. But with CMOS 2.0 μm technology and a high-level synthesis approach, we assume the highest achievable clock rate is about 30 MHz. Therefore, in order to achieve a decoding rate of 200 Mbps, we should choose the constant-input-rate design with 8 bits per cycle or the variable-I/O-rate design with 3 codewords per cycle.

Another performance factor we need to consider is the chip size. Based on our simulation results, we claim that both the above two choices can be implemented on a single chip with 2.0 μm CMOS technology. We choose the pipelined constant-input-rate architecture with 8 bits per cycle for implementation. The major parts have been laid out but not fabricated. The original complexity of the single PLA is very high. Using the output separation and pipeline method, we separate the single large PLA to two pipeline stages. The 1st-stage PLA is decomposed to 64 small PLA's. For the 2nd-stage PLA, since there is no feedback left, the PLA can be decomposed and pipelined as many levels as necessary. The whole architecture is shown in figure 5. The critical path of the first stage is about 40 ns (25 MHz clock rate), with 2 μm CMOS technology. Therefore, a throughput of 200 Mbps is achieved.

### 4 Conclusions

We have designed two different classes of concurrent VLSI architectures for high-speed Huffman decoders: the tree-based architectures and the PLA-based architectures. Their performance, throughput, circuit complexity, and pipeline efficiency are analyzed. We have proved that the goal of a single-chip implementation, a decoding speed of 200 Mbps, and a high-level synthesis approach, is achievable. Moreover, the architectures we presented can be applied to a quite flexible speed range (up to 1 Gbps). Given a Huffman codebook, we can determine the best arrangement of manufacturing technology, optimization techniques, and the degree of parallelism to obtain the best cost-performance ratio.

### References