LOW POWER LOOKUP TABLES FOR HUFFMAN DECODING

Jason McNeely and Magdy Bayoumi
{jbm8240, mab}@cacs.louisiana.edu
The Center for Advanced Computer Studies
University of Louisiana at Lafayette

ABSTRACT

Mobile video devices are energy constrained and therefore need to contain circuits that consume a minimum amount of energy. In this paper, different architectures of lookup tables for Huffman decoding are studied. The PLA type structure is common in these Huffman lookup tables because of their speed and simplicity advantage. However, we determine that the tree structure for table lookup can be a lower power alternative than a PLA structure in certain situations. These situations accounted for 56% of our total simulations runs, and of these runs, the average power savings of the tree in those situations was 78%. Another goal of this paper is to show the effects of varying the table size and varying the probability distributions of a table on power, area, and delay.

Index Terms— video, Huffman codes, variable length codes, table lookup, low power

1. INTRODUCTION

Two opposing restrictions in a mobile video device are having the processing capability to decode fast enough to meet the demands of the desired resolution and quality while still minimizing the energy consumption such that the battery life is prolonged. For a mobile device, some middle ground can be reached where we relax the high speed demands on the decoder, giving us the ability to employ techniques to reduce the energy consumption. The high speed demand is relaxed because most mobile devices cannot support such high resolutions as HDTV (High-Definition Television). For example, a mobile device such as the Apple iPod supports a resolution of 320x240 [1], whereas HDTV requires up to 1920x1088 [2]. In this paper, we compare different lookup table architectures in terms of power, area, and delay, which are used to decode variable length codes coming from the bitstream. Variable length codes, or Huffman codes [3], have been used in many video, imaging, and other compression applications, including in the newest video standard, H.264 [2].

A typical variable length code (VLC) decoder consists of a VLC detector, lookup table (LUT), and a controller. The controller may be as simple as an accumulator. This structure is shown in figure 1. The bitstream entering the

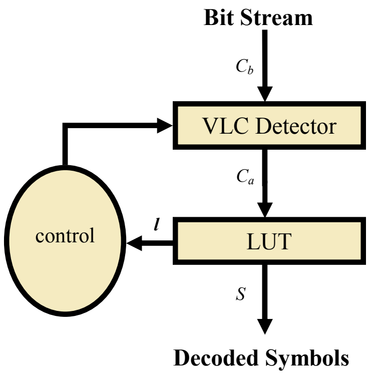

Figure 1: Typical VLC decoder

decoder is a series of codewords whose lengths are not known until they are decoded. A fixed number of bits from the bitstream, $\boldsymbol{C_b}$ (codewords from bitstream), are sent to the VLC detector. The VLC detector is comprised of either a shift register or a barrel shifter depending on if the architecture is serial or parallel. The output of the VLC detector is the aligned codeword, $\boldsymbol{C_a}$ (aligned codeword). By aligned, we mean that the first bit of the codeword starts at the most significant bit position of $\boldsymbol{C_a}$. The aligned codeword is then sent to the lookup table (LUT) for decoding. The LUT is the focus of this paper. The output of the LUT is the decoded symbol, $\boldsymbol{S}$, and the length of its codeword, $\boldsymbol{l}$. The controller can then send this length to the VLC detector to shift the bits $\boldsymbol{l}$ times to align the next codeword for decoding. The VLC detector is re-loaded with another set of $\boldsymbol{C_b}$ bits if it runs out of bits to shift. Other design conventions also exist such as a ROM or PLA (Programmable Logic Array) based finite state machine, as discussed in [4]. However, we only consider the typical architecture in this paper because it fits well with our goals of having an interchangeable LUT style for testing.

For example, consider an alphabet with only 'A', 'B', 'C', and 'D' as possible symbols. Let their codewords be 1, 01, 000, and 001 respectively. Let the incoming bitstream be 10011 (which is 'A', 'D', 'A'). $\boldsymbol{C_b}$ would first be set to 100 and loaded into the VLC Detector. This is because the VLC detector must be able to handle the largest codeword, which in this case is 3 bits. Since the accumulator (controller) starts at 0, no shifting occurs, so C_a also becomes 100. The LUT decodes the 1 in the MSB of C_a as 'A', which is sent out through ***S.*** The LUT also sets ***l*** to one indicating to the accumulator/controller that the VLC detector needs to shift once. So, C_a then becomes 00X (all bits shifted left by one). The 'X' is actually a '1' in this case, since the barrel shifter in the VLC detector just rotated the first bit to the end, but we will not use it again. If no codeword is found, as in this case since 00 alone is not valid, the shifter must be reloaded with more bits through C_b.

As mentioned earlier, there are power and delay restrictions on the variable length decoder. Some authors have successfully shown methods to reduce the power in the decoder by either breaking the LUT into parts and/or grouping codewords in the table [5], [6], [7], or even modifying the barrel shifter in the VLC detector [8], [9].

In [5], a large table is broken into multiple parts. Later, [6] improves on this power reduction by considering successive codewords by making use of a cache type of lookup table. Yet another technique is breaking tables into groups using a mincode for each group [7].

In this paper, however, we are not as concerned about how to divide the table, but what underlying table structure can be used to provide low power decoding, regardless of how the table is broken up. The previously mentioned work focused mainly on how to divide or otherwise manipulate the tables. The actual architecture of table(s) used is usually either not mentioned or specified as PLA, static CMOS, or other. We endeavor here to determine which underlying lookup structure would be a good choice, even when applying one of the previously published power reduction techniques.

2. LUT ARCHITECTURES CONSIDERED

The lookup tables in the variable length decoder (VLD) typically use more power and area than the other components in the VLD [5]. For comparison purposes, we compared the PLA, minimized PLA, and tree architectures. The PLA is a well known structure, and the minimized PLA here is simply a minimized PLA using the Espresso minimization tool. The tree structure is discussed in [4] and an example is shown below in figure 2.

The tree in figure 2 is made up of demultiplexers, leaves which are storage cells (labeled A-D), and two or-gate banks. Each storage cell holds the symbol ID and codeword length for its codeword. Only the storage cell that is activated will output its data, all others output 0's to the or banks. So, from the previous example, if C_a = 100, the first demux would propagate the 1 (or token) to leaf 'A', which is activated and sends out the symbol A (which may be represented by 00 on line ***S***) and the length of 1 to the or banks. The other leaves did not get the token since it went only down one branch of the tree. All other leaves (storage cells), therefore, continue outputting zero to the or banks.

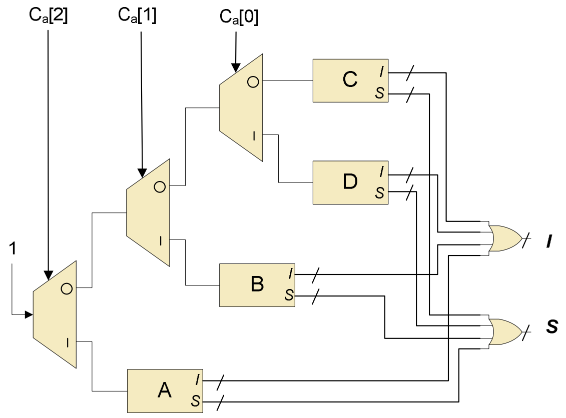

Figure 2: Tree Structure LUT

The tree has not received much attention in research literature lately, probably due to its lower speed, which contradicts the high speed goals of most designs. However, we will show it has power advantages and thus can be used as a low power lookup table in some cases.

3. TEST METHODOLOGY

We compared these three lookup table architectures considering power, area, and delay to make our conclusions. In addition, we also wish to see the effects on the area, power, and delay given different probability distributions of each table size. So, for each architecture (PLA, minPLA, and tree), we varied:

- the size of the table - **ts**
- the probability distribution of the codewords in that table - **X**

For each of the three lookup table architectures, we generated hardware for 11 different table sizes, **ts**, of 5, 19, 33, 47, 61, 75, 89, 103, 117,131, and 145 entries and varied the probability distribution with five different values of **X**: 0.0, 0.5, 0.7, 1.0, and 2.0, for each table size. Thus, we have a total of 55 tables. Once a table's symbols and probabilities for a given **X** and **ts** are created, we create Huffman codewords for each symbol using the Huffman algorithm. Then hardware is generated and simulated. There are 165 total simulation runs, three for each table, since there are three architectures under comparison.

The **X** we used is what we call the "X-factor". An X-factor of zero means all the symbols in the table have the same probability, which means the Huffman coding gives them all approximately the same codeword length. Higher X-factors skew the probabilities more. When **X** is 2, the first symbol has a high probability, and the subsequent symbols probabilities drop off considerably, giving the first symbol's codeword a very short length, and other codewords much longer lengths. Assigning the probability of each codeword in descending order using the X-factor was done with the following formula:

$$P(i,x,s) = \frac{1}{i^X \cdot s} \qquad ts \geq i > 0 \qquad (1)$$

where **i** is the index of the symbol (starting at 1) in the table, **s** is the scaling factor to make sure the probabilities of the table add up to 1 (100%), and **X** is the previously discussed X-factor. The constant **s** for a table is defined as:

$$s(ts,x) = \sum_{i=1}^{ts} \frac{1}{i^x} \qquad (2)$$

where **ts** is the table size. So, for example, for a table size of 33, and X-factor of 1, the scaling factor s would be 4.09. This would give the first symbol (i=1), a probability of 24%, and symbol i=33 a probability of 0.74%.

In order to create the simulation based power analysis, we used a verilog test bench (functioning as the VLC detector) that supplies the aligned codeword to the LUT (tree, PLA, or minPLA) under test and received the symbol and codeword length that the LUT generated. This is to ensure that the LUT was functionally correct and decoded the proper codeword. During the simulation, measurements of power were taken of the LUT. The LUT was at the layout stage in Cadence Encounter. The power measurements were collected by using simulation based power analysis with 0.18 µm technology libraries. Also, it was important that the input C_a be statistically correct. Our testbench had a randomly generated test bitstream which was created based on the statistics of the particular table used to generate the LUT under test. For example, if the first symbol had a probability of 33%, then 33% of the codewords fed to the LUT would have been the codeword of that first symbol. This is more realistic than just randomly selecting a codeword to decode at each time interval, since this would not be the case in a real environment and may cause less reliable power results.

4. RESULTS

Figures 3-7 show the power as a function of table size for each of the X-factor values. The PLA is only shown in figure 3 because its power increases significantly for increasing X-factors, so we wish to focus on the minPLA as compared to the tree.

In nearly all cases, for very small table sizes, the minimized PLA has the lowest power. However, as the table size is increased, and X-factor increased, the tree consumes less power than the minPLA. For example, for **X**=0.7, the tree uses less power for sizes greater than 47. For **X**=2, a table with more than 19 entries would use less power than a minimized PLA. The tree, when **X** is large, will be deep, but not full. Most of the switching activity will be near the top or root of the tree, since the token will not very often

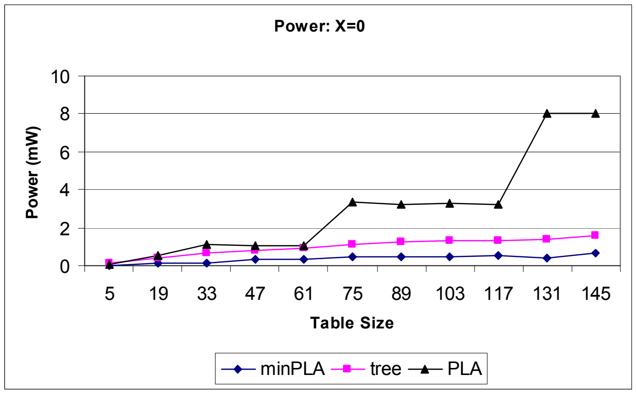

Figure 3: Power consumption of various table types with X=0

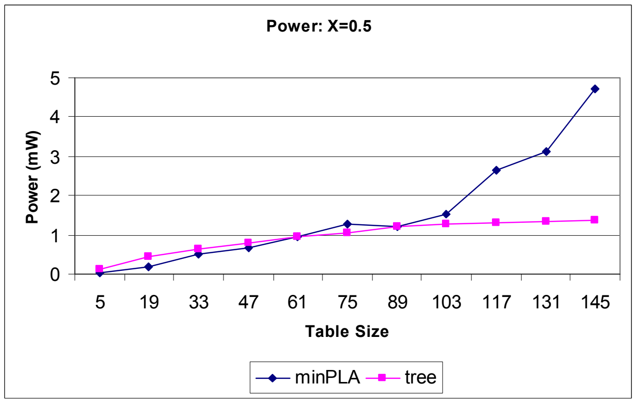

Figure 4: Power consumption with X=0.5

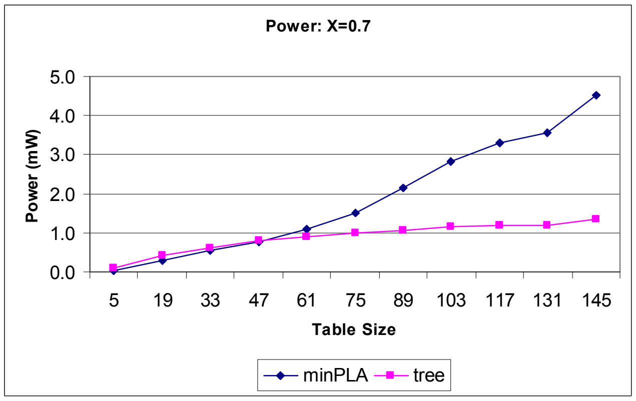

Figure 5: Power consumption with X=0.7

propagate past the top of the tree for large X-factors. If a single bit changes on the minPLA input, multiple inputs could be switched, since each input bit is normally fed to all gates in the and plane, whereas it may only go to a single demux in the tree. An input will go to many demuxes in a tree only for small X- factors, such as 0, where the tree will be shorter and fuller. Also, the tree is fairly stable when it comes to power as compared to the minPLA. The minimized PLAs power generally increases more for increasing table

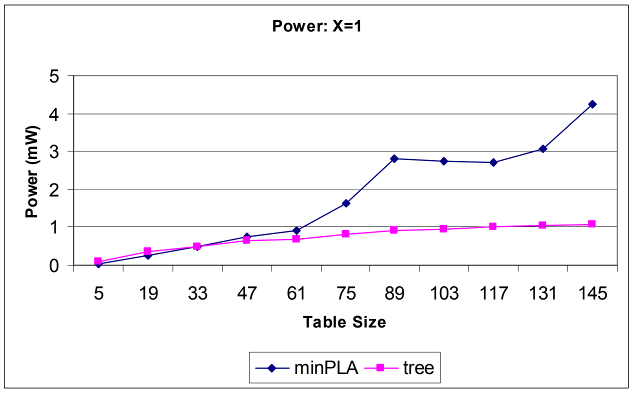

Figure 6: Power consumption with X=1

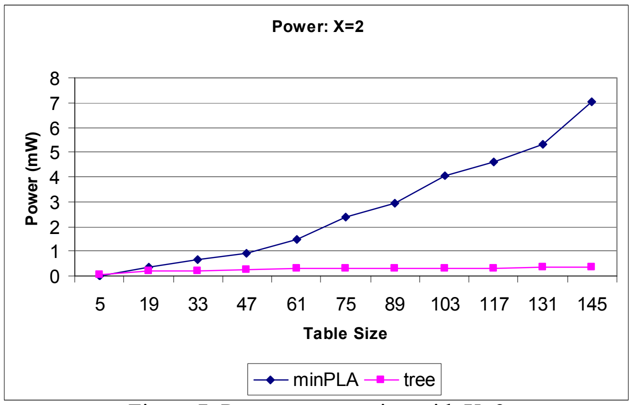

Figure 7: Power consumption with X=2

sizes than the tree's power does. From the total of 55 runs for each style, 31 runs show the tree consuming less power.

Considering the delay of the three architectures, data was also collected. The minPLA is faster than the tree structure in nearly all cases. The average delay across all runs of the tree was 1.86ns and 1.18ns for the minimized PLA. So, for some savings in power, the extra delay of the tree compared to the minPLA may be acceptable.

In terms of area, minPLA is also smaller than that of the tree structure in all cases. We noted that the tree does not vary in size very much for changing values of X. Actually, it is the minPLA whose area varies more widely for a changing X-factor. For example, for the largest table size of 145, the area of the tree for any X-factor stays within 70000-80000 μm^2. However, the minimized PLA varies from about 6000 to 59000 μm^2. Larger X-factors cause an increase in area of the minPLA for a given table size. The tree, on the other hand, basically just re-arranges its leaves and demuxes in response to larger X-factors by making longer branches. So, the area of the tree is more stable and does not vary widely with changing X-factors as the minPLA will. In summary, the area of the tree is larger than the minPLA and usually slower than the minPLA, but has some advantages in terms of power for large and highly skewed tables.

5. CONCLUSIONS

In this paper, we have compared the power consumption of three types of lookup tables. The tree lookup structure consumes the lowest power for cases when the table size is large and the probabilities of symbols are such that the first codewords are highly likely and the subsequent codeword probabilities drop off dramatically (meaning large X-Factor). However, the minimized PLA uses the least power for smaller tables with smaller X-Factors. The choice, therefore, of a table lookup style, depends on the size of the lookup table as well as the probability distribution of that table.

6. ACKNOWLEDGEMENTS

The authors acknowledge the support of the U.S. Department of Energy (DoE), EETAPP program DE97ER12220, the Governor's Information Technology Initiative, and the support of NSF, INF 6-001-006.

7. REFERENCES

[1] Apple, "Apple – iPod – technical specifications," December 2006, http://www.apple.com/ipod/specs.html

[2] Advanced video coding for generic audiovisual services, ITU-T recommendation H.264, May, 2003.

[3] D.A. Huffman, "A method for the construction of minimum-redundancy codes," Proceedings of the IRE, 1952, pp. 1098.

[4] S.-F. Chang and D.G. Messerschmitt, "Designing high-throughput VLC decoder. I. Concurrent VLSI architectures," *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 2, pp. 187-196, 1992.

[5] Seong Hwan Cho, T. Xanthopoulos and A.P. Chandrakasan, "A low power variable length decoder for MPEG-2 based on nonuniform fine-grain table partitioning,", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems,* vol. 7, pp. 249-257, 1999

[6] Sung-Won Lee and In-Cheol Park, "A low-power variable length decoder for MPEG-2 based on successive decoding of short codewords," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing,* vol. 50, pp. 73-82, 2003.

[7] Cheng-Hung Liu, Bai-Jue Shieh and Chen-Yi Lee, "A low-power group-based VLD design," Proceedings of the 2004 International Symposium on Circuits and Systems, 2004, pp. II-337-40 Vol.2.

[8] Chia-Hsing Lin and Chein-Wei Jen, "Low power parallel Huffman decoding," *Electronics Letters,* vol. 34, pp. 240-241, 1998.

[9] P.A. Beerel, Sangyun Kim, Pei-Chuan Yeh and Kyeounsoo Kim, "Statistically optimized asynchronous barrel shifters for variable length codecs," International Symposium on Low Power Electronics and Design, 1999, pp. 261-263.