IMPLEMENTATION OF A FAST MPEG-2 COMPLIANT HUFFMAN DECODER

Mikael Karlsson Rudberg (mikaelr@isy.liu.se)
and Lars Wanhammar (larsw@isy.liu.se)
Department of Electrical Engineering, Linköping University, S-581 83 Linköping, Sweden
Tel: +46 13 284059; fax: +46 13 139282

ABSTRACT

In this paper a 100 Mbit/s Huffman decoder implementation is presented. A novel approach where a parallel decoding of data mixed with a serial input has been used. The critical path has been reduced and a significant increase in throughput is achieved. The decoder is aimed at the MPEG-2 Video decoding standard and has therefore been designed to meet the required performance.

1. INTRODUCTION

Huffman coding is a lossless compression technique often used in combination with other lossy compression methods, in for instance digital video and audio applications. The Huffman coding method uses codes with different lengths, where symbols with high probability are assigned shorter codes than symbols with lower probability. The problem is that since the coded symbols have unequal lengths it is impossible to know the boundaries of the symbols without first decoding them. Therefore it is difficult to parallelize the decoding process. When dealing with compressed video data this will become a problem since high data rates are necessary.

The architecture of the Huffman decoder presented in this paper is based on a novel hardware structure [1] that allows high speed decoding.

The decoder can handle all Huffman tables required for decoding MPEG-2 Video at the Main Stream, Main Level resolutions [2]. The design is completely MPEG-2 adapted with automatic handling of the MPEG-2 specific *escape* and *end of block* codes. In total our decoder supports 11 code tables with more than 600 different code words. Since the code books are static in the MPEG-2 standard the Huffman decoder has been optimized for these specific MPEG-2 codes. A decoding rate of 100 Mbit/s is required and also achieved in our implementation.

2. HUFFMAN DECODER

Huffman decoding can be performed in a numerous ways. One common principle is to decode the incoming bit stream in parallel [3, 4]. The simplified decoding process is described below:

1. Feed a symbol decoder and a length decoder with M bits, where M is the length of the longest code word.

2. The symbol decoder maps the input vector to the corresponding symbol. A length decoder will at the same time find the length of the input vector.

3. The information from the length decoder is used in the input buffer to fill up the buffer again (with between one and M bits, figure 1).

The problem with this solution is the long critical path through the length decoder to the buffer that shifts in new data (figure 1).

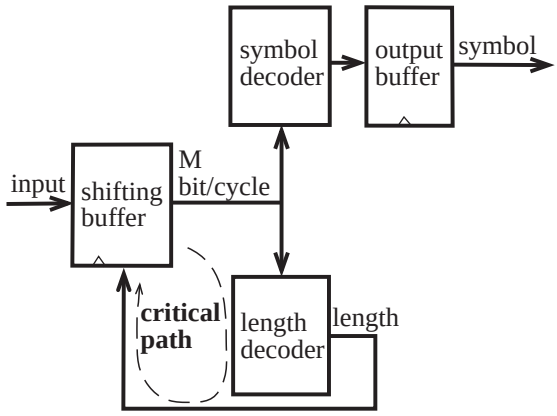

Figure 1. A constant output rate Huffman decoder.

In our decoder the shifting buffer is realized with a shift register that continuously shifts new data into the decoder (figure 2). The length decoder and symbol decoder are supplied from registers that are loaded every time a new code word is present at the input. The decoding process is described below:

1. Load the input registers of the length and symbol decoder.

2. If the coded data has a length of one go back to point 1.

3. If the coded data has a length of two go back to point 1.

and so on with codes of length three and four up to M.

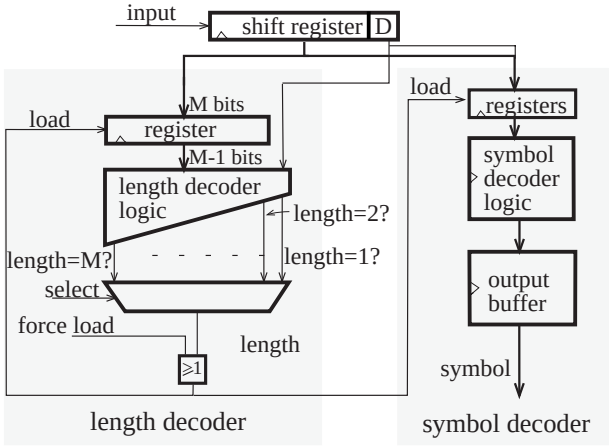

Figure 2. Huffman decoder with relaxed length evaluation time.

This structure allows longer evaluation times for longer code words. The delay in the critical path is reduced to the time it takes for evaluating the length of code words with a length of one or two bits. Codes with other lengths are allowed to be evaluated in several cycles, i.e. code words with lengths of three must be evaluated in two cycles and so on.

Comparing this algorithm with the previous one we note the following:

- The input rate of our new structure is constant while the original has a variable input rate.

- The new structure evaluates short code words in a few cycles but requires more cycles for longer words. The original structure has a constant evaluation time for all code words.

- The new structure allow higher clock rate since the critical path is reduced. But this also means that the symbol decoder must be faster since it in the worst case will receive new data every clock cycle.

- The new structure has a variable output rate while the original one has a constant output rate.

The new structure require higher clock rate to perform the same amount of work. But, if the average code length is short enough the new structure will have a higher speed due to the significantly higher clock rates that can be achieved. Normally the shorter code words will dominate in Huffman coded data and therefore the new decoder is faster during normal circumstances.

2.1 Handling of special markers

Special markers are placed in the data stream to indicate for example *end of block (eob)* at the end of a coded block of data. After this marker other types of data like uncompressed stream information will follow (figure 3). In the Huffman decoder it is essential to detect the presence of this marker to be able to stop the decoding process and let other units process the data that will follow. This decoder detects the *eob* marker in the length decoder and halts the decoding process until a new start signal is applied.

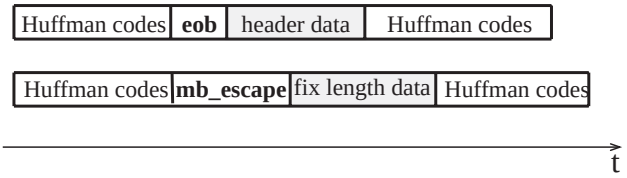

Figure 3. Markers in the MPEG-2 stream requiring special decoding.

The *mb_escape* marker is also important. After this symbol the following data is of fix length. Also this marker is detected in the length decoder and results in that the following data is passed through the symbol decoder unchanged (figure 3).

3. IMPLEMENTATION

The MPEG-2 standard requires that the input data must be decoded at a rate of about 100 Mbit/s. During the implementation special care had to be taken during the partitioning of the symbol decoder and a few critical paths had to be optimized manually. A few modifications of the new decoding algorithm had to be made to make it possible to achieve the targeted performance.

3.1 Improvements of the length decoder

The length decoder turned out to be to slow when evaluating codes with lengths of one or two bits ('*length* = 1*?*' and '*length* = 2*?*' in figure 2). These paths had to be broken up. How this was done is shown in figure 4 below. The evaluation of the '*length* = 1*?*' signal is done by taking data one step

earlier (i.e. from position $i-1$ instead of i) from the shift register and add a flip flop after the evaluation. For the '*length = 2?*' signal the register was moved to after the evaluation logic.

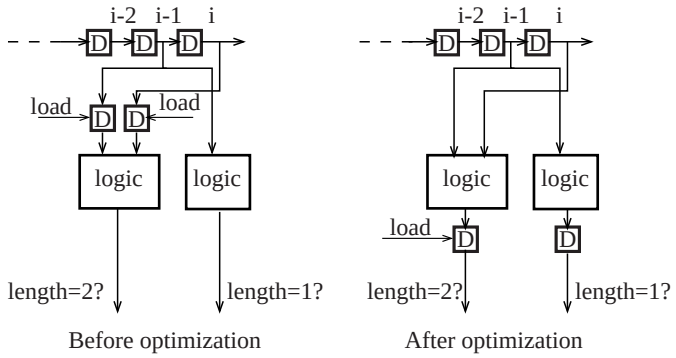

Figure 4. Optimization of critical paths in the length decoder.

Note that this way of breaking up the critical loops can be generalized to removing all critical loops in this structure, see [1].

3.2 Symbol Decoder

The symbol decoding task is more complicated than the length decoding. The symbol decoder could not be designed to receive data in 100 Mbit/s. Code words with a length of one bit are rare in MPEG coded data. The most frequent used code tables do only contain codes with more than two bits. Therefore the symbol decoder is fed with data no more often than every second clock cycle. The input shift register is halted one clock period every time a one bit code is found, and hence, the symbol decoder only need to process 50 Mbit/s without a significance loss in performance. However, this modification causes the input rate to vary.

The symbol decoder were split into five separate units that takes care of their own part of the code tables (figure 5). Every unit consists of an input register that holds the data and a combinatorial block that maps the input vector to the symbol. One of the five units' output are chosen and passed to the output.

Data is in two's complement after the *mb_escape* marker while other data is decoded to signed magnitude format. A post processing stage converts the two's complement data to signed magnitude representation.

3.3 Interface

The interface of the Huffman decoder consists of an eight bit, parallel input port for coded data. A signal indicates when a new input vector can be applied. The decoded data is delivered with a maximum of 50 Msymbol/s. The '*symbol present*' signal (figure 5) indicates when data is valid at the output.

The shift register at the input of the decoder (figure 2) can be read and controlled externally. This is necessary since the Huffman coded data is interleaved with other information.

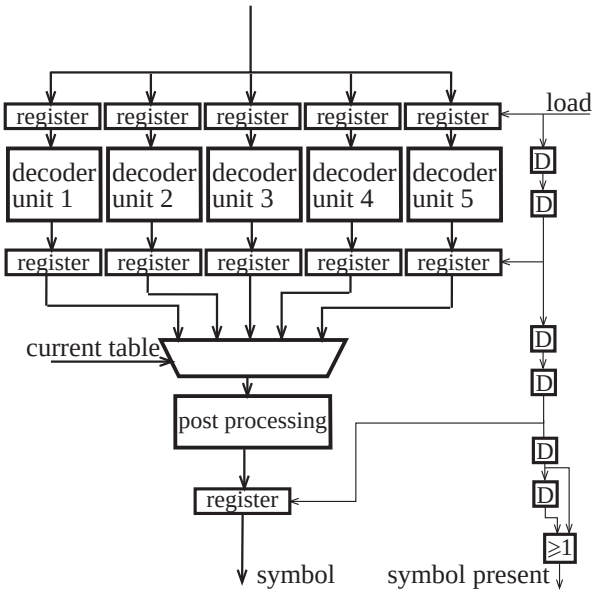

Figure 5. Realization of symbol decoder.

3.4 Synthesis

The decoder has been described in VHDL and then transformed to a circuit using synthesis tools mapping to an 0.8 µm CMOS standard cell library. Some post processing had to be done after the synthesis step to achieve the necessary performance. The main problem was to get the symbol decoder to work fast enough. Therefore the symbol decoding has been split into five separate units. The core area is about 8.4 mm^2 the total area is 14.5 mm^2 (3.9 x 3.75 mm). About two third of the area is occupied by the symbol decoder (figure 6). The power supply is 5V and the transistor count is 26900.

3.5 Symbol tables

To get all symbol tables correct the VHDL code for the symbol decoding as well as for the length decoding has been generated from a thoroughly verified template file. However, this method also yielded a sub-optimal symbol decoding that can be further optimized, but with an increasing probability to introduce design errors in the code tables. This was not done in this implementation because of lack of time and that it was considered more important to get a functional correct implementation.

4. CONCLUSIONS

In this paper an implementation of a novel Huffman decoder architecture has been presented. We have shown that the new structure can be used for fast Huffman decoding while still keeping a simple architecture. The throughput has been increased by using a serial input combined with a serial/parallel length evaluation. Since the current implementation uses standard cells it is reasonable to believe that a full custom version of the same circuit can reach significantly higher speed.

5. REFERENCES

[1] M. K. Rudberg and L. Wanhammar, New Approaches to High Speed Huffman Decoding, IEEE Proc. ISCAS ´96, May 1996.

[2] ISO/IEC DIS 13818-2 Generic coding of moving pictures and associated audio information, part 2: Video, (MPEG-2), June 1994.

[3] S. F. Chang and D. G. Messerschmitt, Designing High-Throughput VLC Decoder Part I – Concurrent VLSI Architectures, IEEE Trans. on Circuits and Systems for Video Technology, Vol. 2, No. 2, pp. 187-196, June 1992.

[4] H. D. Lin and D. G. Messerschmitt, Designing High-Throughput VLC Decoder Part II – Parallel Decoding Methods, IEEE Trans. on Circuits and Systems for Video Technology, Vol. 2, No. 2, pp. 197-206, June 1992.

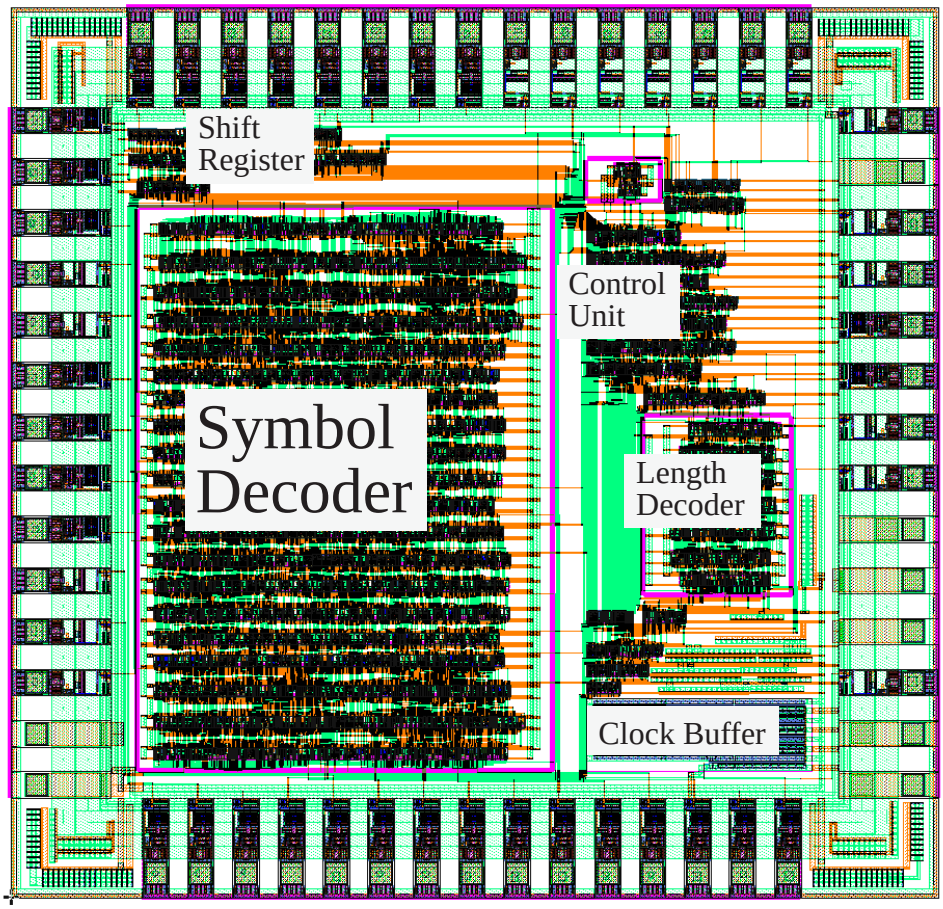

Figure 6. Layout of the Huffman decoder.