HARDWARE EFFICIENT LOSSLESS IMAGE COMPRESSION ENGINE

Lane Brooks and Keith Fife

SMaL Camera Technologies
10 Wilson Rd
Cambridge, MA 02138

ABSTRACT

A complete in-stream lossless hardware image compression engine is implemented with a channel splitting and division-free arithmetic encoding technique. The hardware is less than 7000 gates and requires 0.3 mm^2 of area in a 0.35 μm process. An average of 46% compression is achieved over a diverse set of images.

1. INTRODUCTION

For DSC applications, image compression is desirable for increasing the number of images stored in non-volatile memory and for reducing the time required to download images from camera to host. A frame buffer is usually required in cameras because non-volatile memory write speeds are lower than the desired data rate of image sensors. Typically, image compression is performed after the transfer from sensor to frame buffer since the buffer is readily accessible for performing complex image compression techniques.

In systems where typical compression techniques such as JPEG are not appropriate, positioning the compression between the sensor and the frame buffer has an advantage of reducing the size and cost of the frame buffer itself. In order to exploit this benefit, a hardware efficient compression engine has been developed that performs in-stream image compression.

Data compression can be broken into a modeling and an encoding phase. The modeling phase reduces the entropy by modeling any redundancy in the data set. The encoding phase then compresses the stream to match the entropy of the data set.

2. DATA MODELING

The data stream from the image sensor contains raw Bayer data where pixels in each row alternate between either red and green or green and blue. With this color scheme, a data set that skips every other pixel is likely to show a stronger correlation than just a simple sequence of adjacent pixels. Many different techniques such as CALIC [1] have been developed to do predictive modeling of image data streams. Since row buffers are not available for this particular in-stream compression scheme, all modeling must be limited to one dimension. Thus, this work presents a first-order predictive model that subtracts every other pixel and encodes the result as shown in the block diagram of Figure 4. Table 1 shows the significant performance improvement the Bayer Differencing method offers.

Acknowledgements to Charles Sodini, Harry Lee, and Anantha Chandrakasan.

	Raw Image	Bayer Differencing	Improvement
	bits/symbol	bits/symbol	%
Image A	6.4	3.6	43.7%
Image B	6.7	4.0	40.3%
Image C	7.2	5.5	23.6%

Table 1. Average entropy (bits/symbol) of sample images.

3. DATA ENCODING

Arithmetic encoding is employed to encode the data stream because it does not require large hardware lookup tables or trees. The encoded symbol can be efficiently obtained in a single clock cycle as the result of a calculation. Furthermore, arithmetic encoding can reach the entropy of the data set regardless of the probability distribution [2]. Techniques such as Huffman encoding require each symbol to be encoded with an integer number of bits. Since arithmetic encoding encoding does not have this constraint, a channel splitting technique described later may be employed.

The use of adaptive arithmetic encoding is essential for high compression performance. This requires two hardware intense operations. One is a division to calculate the probability from the adaptive histogram, and the other is a multiplication to rescale the state variables of the encoder. There are some techniques [3] which facilitate multiplication-free arithmetic encoders in order to reduce the hardware requirements of that function. For further reduction in hardware, a division-free adaptive histogram technique is developed here.

4. DIVISION-FREE ARITHMETIC ENCODING

A histogram of M bins has bin counts of m_1, m_2, $\ldots$, m_M and the total number of elements in the histogram is $N = m_1 + m_2 + \cdots + m_M$. When the arithmetic encoder seeks to encode a symbol, the probability of that symbol must be calculated as $p_x = m_x/N$. To remove the need for this division, we use an adaptive histogram building technique that keeps N fixed at a power of 2 such that the division reduces to a simple bit shift. To keep N fixed, we remove elements from the histogram as new ones arrive. One way to do this is to track the order in which symbols are received and remove them in a FIFO fashion. To remove the need for a potentially large FIFO, a weighted round-robin removal approach is constructed as shown in the flow chart of Figure 1. The basic idea is that a histogram removal pointer cycles through the histogram removing elements in a weighted manner as each new symbol arrives. A FIFO for a 256 bin histogram would need to be 256 bytes deep. The round-robin removal approach reduces this requirement to just a single register for the removal pointer and single register for the removal count (x and k respectively as indicated in Figure 1). This reduction in hardware comes without a sacrifice in performance as shown in Table 2.

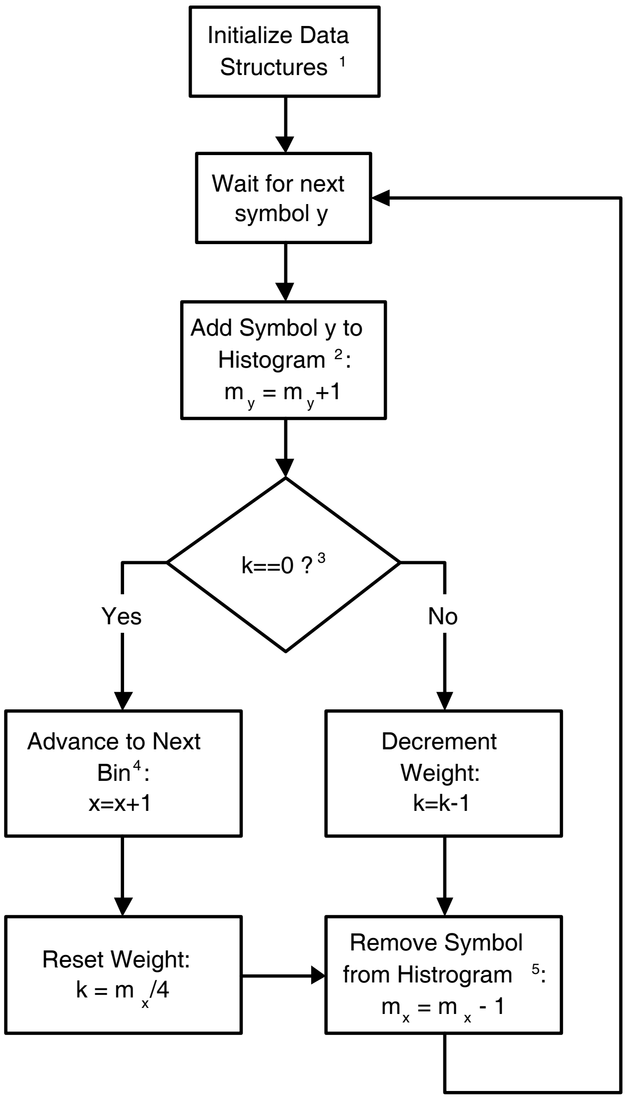
	FIFO Removal	Round-Robin Removal
Image A	55.2%	55.6%
Image B	49.5%	49.7%
Image C	30.5%	30.7%

Table 2. Comparison of compression ratios from arithmetic encoding utilizing different division-free encoding techniques.

5. CHANNEL SPLITTING

One severe problem with using an adaptive compression technique is storing the histogram. In the case of an 8 bit per pixel image stream, 256 bins are needed. Experimentation shows that a depth of 10 bits (i.e. $N = 1024$) in each bin provides a good trade-off between letting the adaptive histogram go stale versus having enough data to yield good statistics. This requires that the histogram be made of 10x256 = 2560 registers. To reduce this requirement, a channel splitting technique was developed.

Channel splitting takes the 8 bit stream and breaks it into smaller independent streams. For example, one can split the 8 bit stream into a stream of the 4 MSBs and a stream of the



Initialize Data Structures [1]
Wait for next symbol y
Add Symbol y to Histogram [2]: $m_y = m_y+1$
k==0 ? [3]
Yes
No
Advance to Next Bin[4]: x=x+1
Decrement Weight: k=k-1
Reset Weight: $k = m_x/4$
Remove Symbol from Histrogram [5]: $m_x = m_x - 1$

1. Set $k = 0$, $x = 0$, & initialize histogram bins.
2. m_y is the number of elements in bin y.
3. k is the removal weight factor. When entering a bin for removal, k is the number of elements that will be removed.
4. x represents the bin from which elements are being removed. This count will wrap to zero when x equals the number of bins in the histogram.
5. m_x is the number of elements in bin x. m_x must always be greater than 0, so if m_x equals 1, then neither the addition or the removal of an element occurs.

Fig. 1. Flow chart of weighted round-robin histogram update procedure for division-free arithmetic encoding.

Fig. 2. Sample images used to demonstrate results.

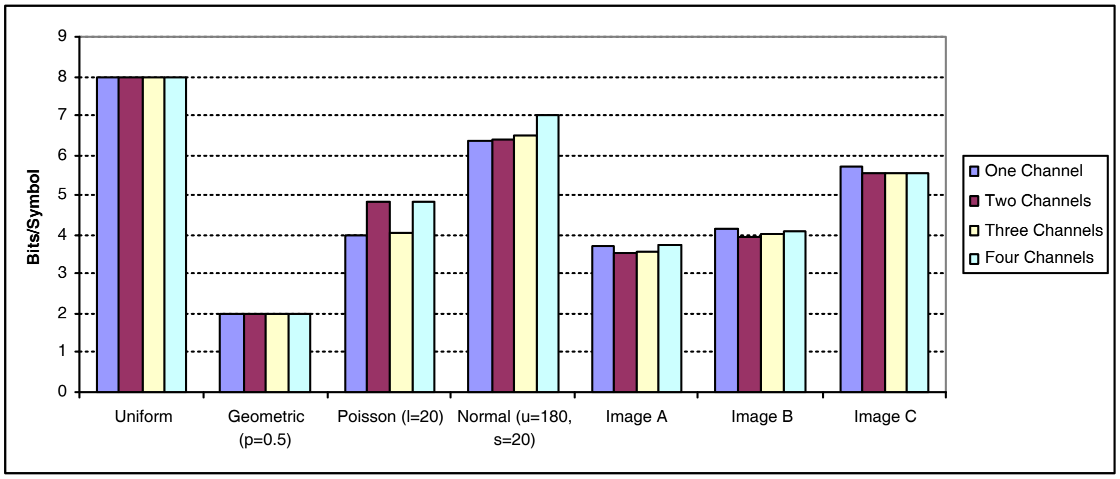


Fig. 3. Entropy comparison for probability distributions and sample images under different channel-splitting configurations.

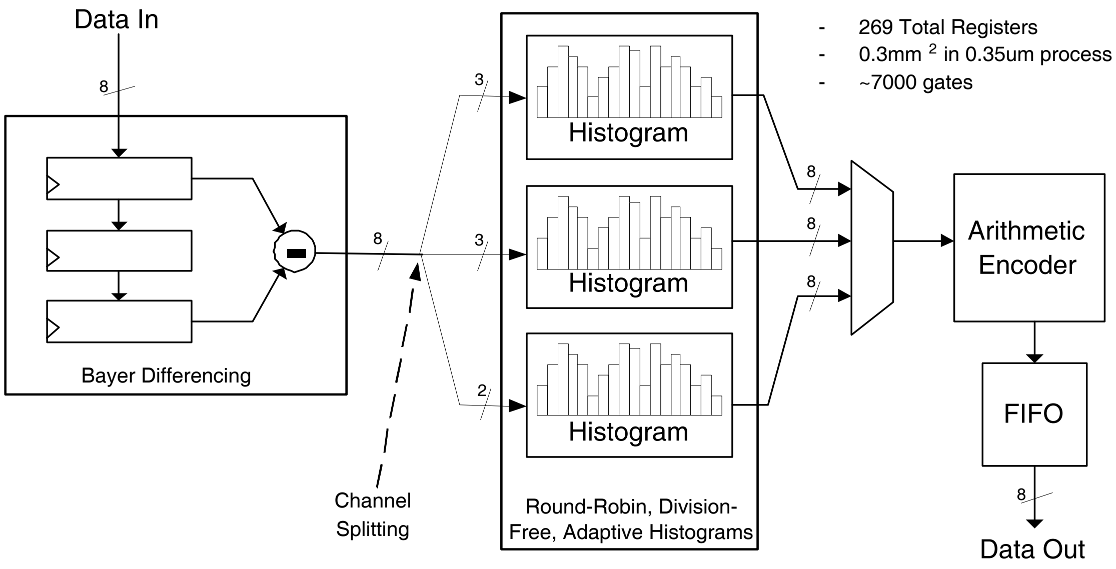


Fig. 4. Block diagram of complete encoder.

4 LSBs. Now two histograms are needed to track the statistics of the MSB and the LSB channel independently. These two histograms now only have 16 bins each. Experimentation with the 2 channel approach shows that the bit depth of each histogram should be 9 bits. Now the register count for the histograms is 2x9x16 = 288 bits which is nearly 10 times smaller than the previous requirement. Further experimentation with splitting into three channels (one 2 bit channel and two 3 bit channels) shows that the bin precision should be 8 bits. This reduces the histogram hardware to 2x8x8+4x8 = 160 registers, which is a 16x reduction in the number of registers needed for the adaptive histogram.

Traditional approaches to entropy encoding employ symbol joining to grow the width of each symbol to yield better results. When symbols are joined, however, the size of the histogram to track the statistics of the image grows with the square of the number of bits. For example, combining two bytes to form a 16 bit channel would cause the histogram to grow from 256 bins to 65536 bins. Channel splitting reverses this approach to allow the histograms to shrink and thus be more amenable to a hardware implementation.

In theory, when no channel splitting is used, the entropy of the symbol to be encoded can be expressed as

$$H_1 = \sum_{i=0}^{255} -p_X(x)\log_2(p_X(x)). \tag{1}$$

When a two channel approach is used to track the statistics, we assume the last N bytes of the data stream accurately describe the true probability distribution of the symbol to be encoded so that the entropy can be expressed as

$$H_2 = \sum_{m=0}^{m=15} \sum_{l=0}^{l=15} p_X(M)p_X(L)\log_2(p_X(M)p_X(L)), \tag{2}$$

where $M = 16m+l$ and $L = 16l+m$. A similar expression can be derived for any configuration of channel splitting. Since the entropy of the data set expresses the amount of information in the symbol in bits/symbol, the performance of the compression engine under the various channel splitting scenarios can be compared through their respective entropies. However, the relationship between Equations 1 and 2 is not obvious, so the results of the various channel splitting options have been compared for different probability distributions in the plot of Figure 3. These samples along with three examples of real images show that the channel splitting performance is comparable to the more traditional single channel approach.

6. RESULTS

The above techniques were developed with the need for a hardware efficient in-stream image compression engine. The resulting system is shown in the block diagram of Figure 4 with the addition of a small output FIFO to absorb jitter produced by the variable length encoding. The round-robin removal technique actually works synergisticly with the channel-splitting technique. Since channel-splitting reduces the number of bins in each histogram, the cycling rate of the histogram removal pointer (x in Figure 1) increases. Working together in this manner, the histogram refreshes quicker, which reduces the precision requirements for the histogram bins (i.e. the reduction of N from 10 to 9 to 8 as the channel splitting went from 1 channel to 2 channels to 3 channels).

Experimentation with this compression engine over a wide class of images yields an average 46% compression ratio which is equivalent to reducing the image from 8 bits per pixel to 4.6 bits per pixel without a loss in image quality. The engine is completely described by the block diagram of Figure 4 in that it functions without the need for any peripheral devices such as a RAM or processor. The total number of registers in the complete design is 269. Synthesized for a 0.35μm process, the design takes 0.3 mm^2 of area and approximately 7000 gates.

7. REFERENCES

[1] Khalid Sayood, *Introduction to Data Compression*, Morgan Kaufmann Publishers, 2000.

[2] Ian Witten, Radford Neal, and John Cleary, “Arithmetic coding for data compression,” *Communications of the ACM*, June 1987.

[3] Bin Fu and Keshab K. Parhi, “Generalized multiplaction-free arithmetic codes,” *IEEE Transactions on Communications*, May 1997.