

Low Power Design of Memory Intensive Functions

Case Study: Vector Quantization

David B. Lidsky, Jan M. Rabaey

Department of Electrical Engineering and Computer Sciences
University of California, Berkeley
Berkeley, California 94720 USA

ABSTRACT -This paper demonstrates techniques to optimize power consumption of memory intensive applications. A design example -a video, vector quantizer encoder-demonstrates how optimization at the algorithm, architecture and circuit level can reduce power consumption by reducing both the effective switched capacitance and the required speed of the system's memory.

1. INTRODUCTION

Traditionally, the two most important criteria used for measuring the performance of a circuit have been speed and area. However, due to both increased transistor density and the advent of portable electronics an increasingly important cost measure in VLSI design is power consumption. While recently a great deal of effort has been put into low power techniques for computation-intensive applications, the impact that large memory accesses have on power consumption, and methods to effectively minimize this power are often neglected. This paper provides a summary of the techniques used in the design of a low-power vector quantizer (VQ), many or all of which can be applied to other memory intensive applications.

The paper is organized in the following manner. Sections 2 and 3 give an overview of low-power design and vector quantization respectively. Section 4 explores the different levels of power optimization. Sections 5 and 6 gives the results of different implementations focusing on architectural optimizations.

2. LOW-POWER VLSI

Neglecting short circuit current, power consumption of static CMOS circuits is quantified by its dynamic component:

$$P = C_{sw} V^2 f \quad (1)$$

where C_{sw} is the effective capacitance switched per clock cycle, V is the supply voltage, and f is the clock frequency[1],[2].

It has been shown that the most effective way to reduce power consumption is to reduce the supply voltage resulting in a quadratic reduction in power. Clock frequency may also be reduced by the judicious use of parallel or pipelined structures. Parallel architectures and dedicated hardware can also be used to reduce the effective switching capacitance[3].

Low-power design must be approached at all levels. The high level decisions have profound effects on power consumption (section 4.1), because they set bounds on the amount of computation required. Architectural choices can be made to

strongly reduce power (section 4.2). Finally, circuit level optimizations can reduce power by minimizing the physical capacitance being switched (section 4.3).

More important than the individual levels of design, it is imperative to keep an overall view of how the different levels of design interact. As will be discussed, certain implications of high level decisions may not become apparent until the details of lower level design are explored.

3. VECTOR QUANTIZATION

Vector Quantization (VQ) is a data compression method used in voice recognition and video systems. For example, the wireless InfoPad terminal being developed at UC Berkeley [4] requires low power video compression of grey scale images. The VQ encoder, whose design is discussed herein, is specifically customized for future use on such portable units.

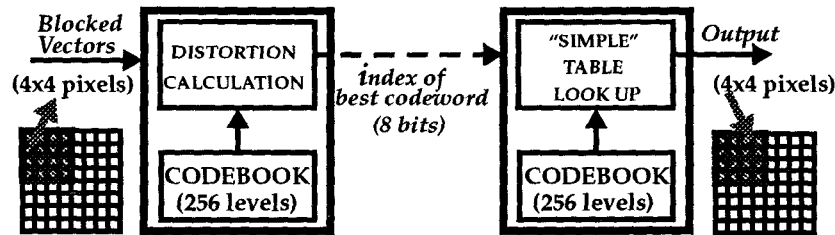


Figure 1: Vector Quantization, Encoding and Decoding

In video quantization, a video image is broken up into a sequence of 4x4 pixel images. Each pixel is represented by an 8-bit word representing luminance. The 4x4 image, therefore, can be thought of as a vector of 16 words, each 8 bits in length. Each of these vectors are compared with a previously generated codebook of, in this case, 256 different vectors. This codebook is generated a priori with the intention of covering enough of the vector space to give a good representation of all probable vectors. The creation of the codebook is studied in greater detail in [5]. After compression, an 8 bit word is generated delineating the address of a codeword which approximates the original 4x4 vector image. This corresponds to a compression ratio of 16:1, since 16 8-bit words are now represented by a single word.

To decode the quantized image, the receiver, which has an identical codebook, executes a memory lookup to recover the encoded vector. The simplicity of the VQ decoder is the principle reason that VQ is utilized for sending video to portable, low-power terminals. A low-power encoder, however, is required to enable a terminal to compress and transmit moving images.

The InfoPad employs a 240 x 128 pixel grey scale display. Processing the standard 30 frames/sec moving picture necessitates that one 4x4 pixel-vector be compressed every 17.3ms.

Distortion Measure: The standard distortion measure used in VQ is Mean Squared Error ($MSE = \sum (C_i - X_i)^2$) where C is the codebook codevector, X is the original 4x4 vector representation, and i is the index of the individual pixel word.

Another distortion measure, Absolute Error ($AE = \sum |C_i - X_i|$) has the advantage of less computations, at the possible expense of a less accurate reproduction of the original image.

Video accuracy is a subjective science, so designers must make decisions on what is "accurate enough" for their purposes. For example, a designer may choose AE over MSE to save multiplication operations, if it is decided that the power savings overshadows image considerations. In this analysis, the standard, MSE, was chosen. Using AE in its place would reduce power due to computation, further accentuating the effects of memory power savings.

Full Search Vector Quantization (FSVQ): Possibly the most straight forward method to scan the codebook is a full search. The 4x4 vector is compared with each of the codevectors. The vector which yields the smallest distortion measure is chosen to code the image.

The main drawback of FSVQ is the relatively large amount of processing required to compress each vector. This is expensive in terms of both area and power consumption. In order to meet real time constraints the intense computation of FSVQ necessitates increased chip area (parallelism or pipelining) as well as a lower bound on the clock frequency and supply voltage. The switched capacitance per vector is also comparatively large due to the amount of computation required.

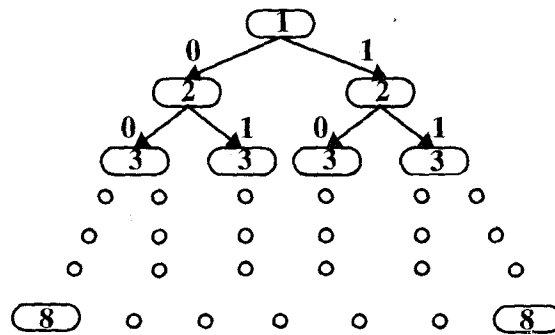


Figure 2: Tree Search Encoding

Tree Search Vector Quantization (TSVQ): Tree Search Vector Quantizer (TSVQ) encoding[5] requires far less computation. TSVQ performs a binary search, instead of a full search, of the vector space. As a result, the computational complexity is proportional to $\log_2 N$, rather than N , where N is the number of vectors in the codebook. Figure 2 diagrams the structure of the tree search. At each level of the tree, the input vector is compared with two codebook entries. If for example at level 1, the input vector is closer to the left entry, then the right portion of the tree is never compared below level 2 and an index bit 0 is transmitted. This process is repeated until a leaf of the tree is reached. Here only $2 * \log_2 256 = 16$ dis-

tortion comparisons have to be made, compared to 256 distortion calculations in the FSVQ.

There are many other vector quantization options that may be explored, such as a Pruned Tree Search Vector Quantizer(PTSVQ), which has the advantage of requiring fewer memory accesses. Another option, using adaptive codebooks for greater accuracy, can be used in both FSVQ and TSVQ[5].

4. POWER OPTIMIZATIONS

When designing for low power, decisions must be made on all levels including compression method, algorithmic choices and possible mathematical optimizations or transformations. Furthermore, there are architecture and partitioning choices, and, at the lowest level, gate and transistor level optimizations.

4.1 Algorithmic Level

There are many options which should be considered in algorithmic choice. The first is the type of coding, herein chosen to be vector quantization. This section discusses the different high level algorithmic options.

Full Search versus Tree Search: The computational intensity per vector for any kind of search can be quantified by enumerating executions (e.g. memory accesses, multiplications, additions, etc.) required to search the codebook. This gives a reasonable high order estimate of relative power consumption.

Computing the MSE between two vectors requires 16 memory accesses, 16 subtractions, 16 multiplies and 16 additions. In FSVQ, this is done for each vector in the codebook, and each of these vectors are compared with the leading MSE candidate at the time.

A tree search, on the other hand, requires far fewer distortion measures, and therefore fewer computations. For each node in the tree, it is necessary to compute the difference between two MSE's and use that result to compute the location of the next node to be compared. This requires 32 additions, 32 multiplies and 33 subtractions. One subtraction performs the comparison where the outcomes sign bit is used to select the next node in the tree. This distortion measure need only be calculated 8 times per vector as opposed to the 256 distortion calculations required FSVQ. Computational intensity is summarized in table 1.

While there is significant power savings in TSVQ as compared to FSVQ, there is a slight degradation in signal quality. This distortion, however, is not as great in most cases as to be prohibitive [5], thus TSVQ is chosen for its greater power efficiency.

Mathematical Optimizations: In TSVQ, there is a large computational reduction available by mathematically rearranging the computation of the difference between the original vector, X , and two codevectors C_a and C_b [6]:

$$MSE_{ab} = \sum_{i=0}^{15} (C_{ai} - X_i)^2 - \sum_{i=0}^{15} (C_{bi} - X_i)^2 \quad (2)$$

Since in TSVQ the same two codevectors are always compared, the calculation of the errors can be combined under one summation. With the quadratics expanded:

$$MSE_{ab} = \sum_{i=0}^{15} C_{ai}^2 - 2X_i C_{ai} + X_i^2 - \left(C_{bi}^2 - 2X_i C_{bi} + X_i^2 \right) \quad (3)$$

and then simplified and regrouped:

$$MSE_{ab} = \sum_{i=0}^{15} \left(C_{ai}^2 - C_{bi}^2 \right) + \sum_{i=0}^{15} 2X_i (C_{bi} - C_{ai}) \quad (4)$$

the first summation can be precomputed in software and stored in one memory location. The quantities $(C_{bi} - C_{ai})$ may also be calculated and pre-stored. The multiplication by 2 is achieved with a hardwired shift operation dissipating no power.

Therefore, for each level of the tree, the number of multiplications is reduced from 32 to 16, memory accesses from 32 to 17, and add/subtracts from 33 to 17. Table 1 quantifies the number of operations for an entire 256 vector codebook for the different algorithms considered. With the mathematical transformations, however, each of the memory accesses are now extended from 8 bits to 9 (the first summation can be quantized without observable error), but there is still significant reduction in memory size and power dissipation.

TABLE 1. Computational Complexity

	Memory Access	Multiplic- ation	Add/ Subtract
Full Search	4096	4096	8448
Tree Search	256	256	520
Optimized Tree	136	128	136

It should be observed that the choice of algorithm is an iterative process. For example, the TSVQ may be the right algorithm for a specific application, but its advantages may not be known until well into the design process. As shown here, mathematical optimizations can save dramatically on power. The next section illustrates how architectural decisions effect power consumption.

4.2 Architecture

Once an algorithm is chosen, the next step is to chose an architecture which minimizes power consumption while meeting timing and area requirements.

Single Memory/Processor: In one approach, all the units are grouped together and time multiplexed as shown in figure 3. To process one node of the tree requires 17 memory accesses, 16 multiply/accumulate operations, and a final add to produce the comparison bit indicating the location of the next node of the tree. A total of eighteen clock cycles are required per comparison: the first cycle to fetch the first word to be compared, then 16 operations for the multiply/accumulate, and the 18th cycle to produce the next location.

The same number of calculations are required for each node of the tree. This example requires $8 \times 18 = 146$ clock cycles. Processing a vector every 17.3ms neces-

sitates a clocking frequency of 8.3MHz and memory access time less than 115ns, which in turn puts a lower bound on the supply voltage.

The usual methods for shortening the critical path, and hence the operating voltage and clocking frequencies, are difficult to apply using the single memory configuration. The memory must be accessed 16 of the 18 clock cycles making it impossible to process more than one vector at a time. Therefore methods to reduce the critical path, such as pipelining and parallelism, can not be implemented without duplicating the entire memory.

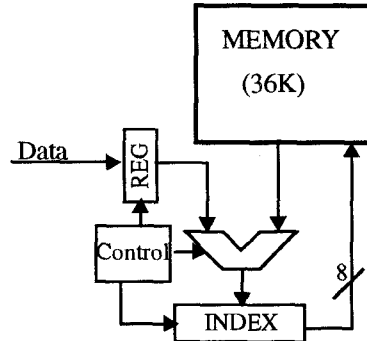


Figure 3: Centralized Memory

If the memory is partitioned, however, and a distributed memory approach is used [7], pipeline stages can be introduced, enabling reductions in voltage, clocking frequency, and switching capacitance. The distributive memory architecture is discussed in a later section.

Memory Architecture: For each vector comparison, there are sixteen eight-bit words that must be accessed. A technique often used in high throughput designs is to grab bytes of words rather than single words in order to reduce the critical path, allowing the reduction of clock speed and voltage[8]. In the case of the TSVQ, however, pre-fetching can not be used to lower the critical path since the location of the vector to be chosen is dependent on the operations at the previous node of the tree. This parallel architecture (figure 4), however, is quite useful in reducing the switched capacitance by reducing the number of memory accesses.

In this implementation, the words are grabbed in 4 word bytes and read off in nibbles. The energy required for a single read is greater, but the memory is accessed 4 times less frequently. Therefore, there is a savings in capacitance switched since the power dissipated in multiplexing is significantly less than the overhead power required by the memory. The overall control logic remains identical since the same control signals are required for both implementations.

The advantages of this architecture decreases with the size of the memory. This architecture may also prove unattractive from an area standpoint. In the case of the smaller memories of the distributive architecture, for example, using a bit width of

36 bits in a memory less than a kilobyte has too great an area cost, and lower power was sacrificed for area.

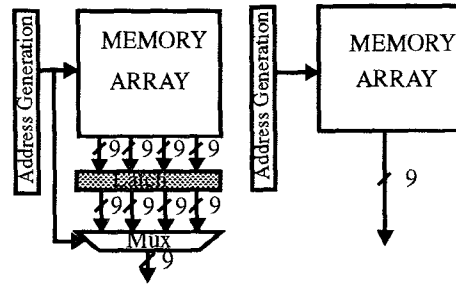


Figure 4: Parallel versus Serial Access

Table 2 illustrates the power savings using parallel accessing of an on-chip SRAM. Power savings are even higher if accessing an off-chip memory due to the larger interconnect capacitance.

TABLE 2. Energy Per Vector Access vs. Memory Organization

	4 access + 0 multiplex	2 access + 2 multiplex	1 access + 4 multiplex
16 Kbits	360.4 pJ	320.0 pJ	290 pJ
8 Kbits	243.2 pJ	200.2 pJ	186.3 pJ
4 Kbits	186.0 pJ	150.6 pJ	135.5 pJ

Distributive Memory: In TSVQ each level of a tree has specific codevectors associated with it, and the codevectors from each level are found only at that level. Therefore the memory can be partitioned into separate memories for each level of the tree[7]. Associated with each memory are identical processing elements and controllers. With this architecture, a pipelined structure may now be utilized and the critical path is reduced drastically (figure 5).

The chip can now process eight vectors simultaneously with only a negligible increase in latency. The frequency of the clock can be reduced by eight, and since the execution units need only process at 960ns, the supply voltage can be dropped from 3.0V to 1.1Volts.

A less obvious benefit is the reduction of switching capacitance. A large part of the power on both the single and distributive memory architectures is dissipated by memory accesses. The distributive memory architecture is switching less capacitance reading codevectors compared with the centralized case since there is less overhead in reading from smaller memories. Although there are now eight controllers and eight processors clocking on chip, they are being clocked at one-eighth the frequency, so the energy dissipated per vector by these elements does not change. The effect of the interconnect between blocks is negligible since it is clocked at one

eighteenth of the clocking frequency. With the distributed architecture, memory power is no longer dominant (Figure 6).

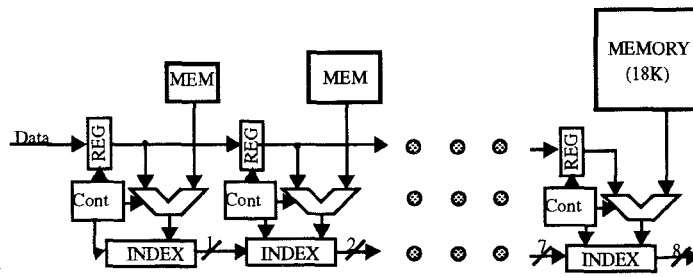


Figure 5: Distributive Memory

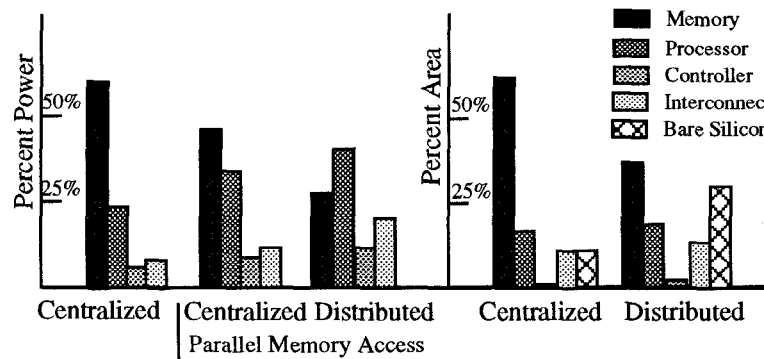


Figure 6: Power and Area Breakdown

4.3 Circuit Level Optimizations

All circuits were designed using UC Berkeley's low-power cell library which employs minimal transistor sizes to reduce gate and diffusion capacitance, and tiled datapath structures to reduce interconnect capacitance.

To reduce glitching, registers were used to hold previous inputs on adders during idle clock cycles ensuring that adders would not dissipate energy computing irrelevant inputs. To minimize clock power, a single clock is used and registers are implemented using minimum sized True Single Phase Clock Registers (TSPCR)[8].

Memory is implemented with a low-power, on-chip SRAM. To reduce power dissipation, the bit-lines are pre-charged to an NMOS threshold below the supply voltage to limit voltage swing. The memory is broken up into blocks with only one active at a time reducing spurious transitions[9].

5. IMPLEMENTATION

To illustrate the effectiveness of the multi-stage approach to power reduction, three architectures were implemented: the single memory with serial access, and the single and distributive memory TSVQ encoders with parallel memory access.

At this time, complete testing have not been completed, so all numbers are from extensive IRSIM and SPICE simulations.

TABLE 3. Simulation Results

	Power	Memory Power		Core Area	Operating Voltage
		Relative	Absolute		
Centralized Serial Access	6.96 mW	60 %	4.15mW	53.3 mm ²	3.0 Volts
Centralized	4.28 mW	46 %	1.95mW	46.6 mm ²	3.0 Volts
Distributed	412 μW	28 %	114 μW	92.9 mm ²	1.1 Volts

5.1 Area

The area required for the distributed memory implementation is 2.3 times that of the single memory case. The area breakdown of the chip is shown in figure 6. As can be seen in the chip photo of the single memory implementation (figure 8), memory dominates chip area. The memory is split into two sections. Memory 1 contains the codevectors for the first summation of (EQ4); memory 2 contains the remaining codevectors. In the distributive case, it is difficult to get all blocks to intermesh resulting in greater empty area. By meshing the layout of the different levels together, an area reduction of about 15% should be achieved.

The area figures underscore an important point. Often the most effective way to achieve low-power designs is through the use of greater silicon area. Therefore, the designer must counter-balance seemingly orthogonal properties to achieve the most practical design.

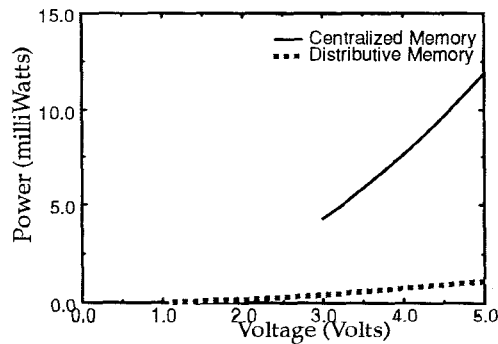


Figure 7: Power versus Voltage

5.2 Power

The power consumption of the designs is summarized in table 3. Modifying the centralized design to include parallel memory accesses reduced the switching capacitance by 40%. The capacitance, as well as the critical path, is further reduced by memory partitioning. As the results indicate, the memory component of the total power consumption was reduced more than a factor of 2 facilitating absolute power reduction by a factor of 17.

6. CONCLUSION

To achieve substantial power reduction in memory intensive functions, a multi-level approach is essential. The most profound impact is typically found through proper choice of algorithm and architecture. Algorithmic level decisions have been shown which substantially reduce computational complexity and lower the number of memory accesses by a factor of 30 for the TSVQ. The architectural benchmarks demonstrated additional power reduction by a factor of 17 primarily due to a reduction in the percentage of power devoted to the memory by a factor of two and due to a reduction of critical path, enabling low voltage operation.

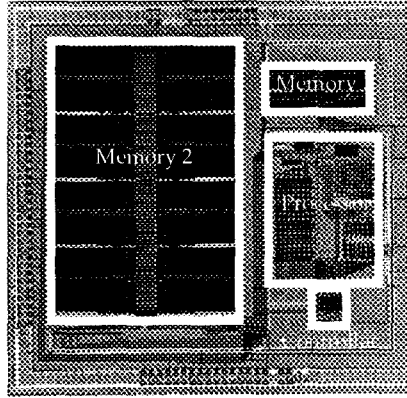


Figure 8: Centralized Memory Chip

ACKNOWLEDGEMENTS

The authors acknowledge Anantha Chandrakasan, for his early work in this field and overall insight and encouragement. This project is sponsored ARPA grant 442427-26003.

References

- [1] J. Rabaey, "Digital Integrated Circuit: A Design Perspective," *Prentice Hall*, to be published in 1994.
- [2] H. Veendrick, "Short-Circuit Dissipation of Static CMOS Circuitry and Its Impact on the Design of Buffer Circuits, *IEEE JSSC*, vol. sc-19, pp. 468-473, August 1984.
- [3] A. Chandrakasan, *et al*, "Low-power CMOS Digital Design," *IEEE JSSC*, Vol.27, No 4, pp. 473-484, April 1992.
- [4] R. Brodersen, *et al*, "Design Considerations for Portable Systems," *IEEE JSSC*, vol 27, no.4 pp. 473-484, April 1992.
- [5] A. Gersho, *et al*, "Vector Quantization and Signal Compression," Kluwer Academic Publishers, Boston, MA, 1992.
- [6] W. Fang, *et al*, "A Systolic Tree-Searched Vector Quantizer for Real-Time Image Compression," *Proc.VLSI Signal Processing IV*, IEEE Press, NY, 1990.
- [7] R. Kolagotla, *et al*, "VLSI Implementation of a Tree Searched Vector Quantizer," *IEEE Trans. on Signal Proc.*, vol. 41, no 2, February 1993
- [8] C. Svensson, *et al*, "High-Speed CMOS Circuit Technique," *IEEE JSSC*, pp. 62-70, February 1989.
- [9] A. Chandrakasan, *et al*, "A low Power Chipset for Portable Multimedia Applications," *ISSCC*, March 1994.