Design of a New Squaring Function for the Viterbi Algorithm

Aria Eshraghi, Terri Fiez and Thomas Fischer
Washington State University
Pullman WA

Kel Winters
Advanced Hardware Architectures
Moscow ID 83843

Abstract- **A new algorithm and hardware implementation of the Viterbi squaring function has been developed. The use of an approximation squaring technique preserves the Viterbi performance as is demonstrated by Monte-Carlo simulations. Additionally, the 16-bit approximate squaring implementation is expected to require one-fourth the area and operate at three times the speed of the conventional squaring implementation.**

1 Introduction

The Viterbi algorithm [1] is used to encode and decode data in communication systems. This algorithm has been utilized in trellis coded modulation (TCM), trellis shaping (TS), and trellis coded quantization (TCQ). Use of the Viterbi algorithm at the TCM receiver results in 3-6 dB improvement in signal-to-noise ratio (SNR) for a given error rate [2]. In TS, the use of the Viterbi algorithm at the transmitter results in a reduction of 0.9 dB in the transmitted energy [3]. In TCQ, the Viterbi algorithm is used as a means of data compression, showing better performance than realizable vector quantizers [4] for the encoding of a memoryless source. The recent use of the Viterbi algorithm in communication products such as disk drives, tape drives, and modems has triggered the development of a single chip Viterbi processor [5].

This paper focuses on an efficient algorithm for performing the squaring function in the Viterbi processor. The new squaring function uses significantly less area than the conventional squaring techniques and, at the same time, boosts the speed of the processor without reducing its accuracy. In the first portion of this paper, the algorithm for the approximated squaring (APSQR) function is presented. Through Monte-Carlo simulations, it is shown in section two that this new squaring function results in accurate performance of the Viterbi algorithm. Section three covers the hardware implementation of the APSQR and compares the APSQR with a conventional squaring scheme.

2 Approximated Squaring Function

The Viterbi algorithm also known as forward dynamic programming [6], is an efficient search technique for determining the minimum cost sequence of states in a finite state machine. The

state transitional behavior of the finite state machine in time is mapped into a digraph known as the trellis diagram. As a result, only the path in the trellis diagram that agrees most closely with the received sequence is retained. The optimum path is determined by the path with the minimum mean square distance in the trellis diagram. Thus, the core computation in the Viterbi algorithm is:

$$\lambda_{t+1,j} = \min(\alpha_{k1,j} + \lambda_{t,k1} \; or \; \alpha_{k2,j} + \lambda_{t,k2}). \tag{1}$$

Where λ is the *path length*, α is the *path metric*, and $\lambda_{t+1,j}$ is the optimum path terminated at state j. Computing the path metric requires a dedicated squaring function ($\alpha_{k,j} = \beta^2_{k,j}$) which occupies approximately 30% of the total Viterbi processor chip area using conventional circuit techniques.

The conventional squaring technique relies on decomposing a number into the sum of the least significant bit and the remaining bits. The square is calculated using $(x+y)^2 = x^2 + 2 \cdot x \cdot y + y^2$ and applying it recursively to a shifted version of the remaining bits. By decomposing the number into the sum of the most significant bit and the remaining bits, it is possible to approximate the square of the number without degrading the accuracy of a Viterbi algorithm.

We will now describe the approximated squaring algorithm. Let $A = a_n a_{n-1} a_{n-2}..a_4 a_3 a_2 a_1$ be the number to be squared. Next, decompose A into the sum of the most significant bit and the remaining bits:

$$A = a_n 0\ 0..0\ 0\ 0\ 0 + a_{n-1} a_{n-2}..a_4 a_3 a_2 a_1. \tag{2}$$

We expand A^2, or rather A^{10} in binary, into:

$$A^{10} = (a_n 0\ 0\ ..0\ 0\ 0\ 0 + a_{n-1} a_{n-2}..a_4 a_3 a_2 a_1)^{10}. \tag{3}$$

Now applying $(x+y)^2 = x^2 + 2 \cdot x \cdot y + y^2$, A^{10} becomes:

$$A^{10} = a_n 0\ 0..0\ 0\ 0\ 0^{10} + (10)(a_{n-1} a_{n-2}..a_4 a_3 a_2 a_1)(a_n 0\ 0..0\ 0\ 0\ 0) + (a_{n-1} a_{n-2}..a_4 a_3 a_2 a_1)^{10}. \tag{4}$$

Neglecting the last term, A^{10} is approximated as:

$$A^{10} \simeq a_n 0\ 0..0\ 0\ 0\ 0^{10} + (10)(a_{n-1} a_{n-2}..a_4 a_3 a_2 a_1)(a_n 0\ 0..0\ 0\ 0\ 0). \tag{5}$$

This can be rewritten as:

$$A^{10} \simeq (a_n 0\ 0..0\ 0\ 0\ 0 + a_{n-1} a_{n-2}..a_4 a_3 a_2 a_1 0)(a_n 0\ 0..0\ 0\ 0\ 0). \tag{6}$$

The approximate squaring of A requires summing the two most significant bits (first term) and a left shift of $(a_n 0\ 0..0\ 0\ 0\ 0 + a_{n-1} a_{n-2}..a_4 a_3 a_2 a_1 0)$ by n-1 bits.

Fig.1 shows a plot of the output versus the input for both the approximated and the actual squaring functions. The maximum error is 25% and corresponds to input amplitudes of $2^n - 1$ where n is an integer. Using the approximated squaring function, the average error is approximately 10%.

3 Simulation Results

Although the average error due to the APSQR function is relatively high, the Viterbi algorithm inherently compensates for noise (or errors) in the data. To illustrate this property, Monte-Carlo simulations have been used to demonstrate the effect of approximating the squaring function on the TCQ.

The performance of the TCQ was measured for a Memoryless uniform source with a uniformly distributed codebook. The simulation results were based on encoding 1,000 different blocks of length 10,000 random data samples. The resolution for the source was selected to be 15 bits. This reduces the effect of finite resolution used in TCQ which results in better measurement of the APSQR function performance. The simulation results are shown in Table 1.

Table 1
The performance versus the bit rate
for the conventional and the approximated
squaring function.

Rate (Bits)	*Conventional* SNR *dB*	*Approximated* SNR *dB*
3	18.776	18.748
4	24.940	24.912
5	31.029	31.002
6	37.085	37.058

The bit rate is the number of bits per sampled input data. SNR is the expected signal to noise ratio, and it is given in dB. The variance was less than 0.006 dB in each case. As one can see, the degradation in performance of the TCQ due to utilization of APSQR is less than 0.03 dB for the expected value of the SNR. Thus, approximating the squared numbers produces a negligible error in the output. In the next section, it is shown that there is a significant saving in circuit area and increased speed with this implementation.

4 Hardware Implementation

To illustrate the efficiency of the APSQR function, the conventional squaring function is first described. The conventional implementation of the squaring function has a cellular architecture such that a single block is repetitively used in the design[7, 8]. Fig. 2 shows a cellular implementation of a 7-bit squaring function of [7]. Each cell contains a full adder and a multiplexer with connections shown. The input signals, $a_1..a_7$, enter at the top of Fig. 2 and propagate vertically through the cells. Simultaneously, the carry bits, C_i and C_o, propagate from right to left. This implementation uses $\sum_{i=3}^{n} i$ full adders and multiplexers for an n-bit squaring function. For example, a 7-bit squaring function requires:

$$\sum_{i=3}^{7} i = 3 + 4 + 5 + 6 + 7 = 25 \quad (7)$$

or 25 adders and 25 multiplexers. The number of adders and multiplexers increases quadratically with the number of input bits, i.e. $\sum_{i=3}^{n} i = (n^2 + n + 6)/2$. The increased hardware resulting from the increased number of input bits consumes excessive area and dynamic power, and significantly reduces the speed compared to the APSQR function.

The speed of the cellular squaring function is limited by the propagation of the carry bit through the last chain of adders. In Fig. 2 the worst case delay occurs as the carry bit propagates from cell 1 through cell 7. Note that the delay increases linearly as the number of input bits increases. Additionally, this design is not suitable for pipelining because of the two dimensional signal flow.

The proposed hardware implementation for a 7-bit APSQR is shown in Fig. 3. The controller circuit is responsible for detecting the most significant bit, and controlling the multiplexers for the proper number of left-shift operations. Three layers of multiplexers pass or shift their input by one, two, and three bits to the left. The final layer consists of modifier cells which sum the two most significant bits. The modifier cell becomes active if its input corresponds to the most significant bit. The detection of the most significant bits by the modifier cell is accomplished through observing the output of the controller.

The number of multiplexers used in this architecture with n input bits is upper bound by:

$$\text{Number of Multiplexers} = (n-1)\log_2(4 \cdot n) \tag{8}$$

As an example, a 7-bit input requires 35 multiplexers. Based on this equation, the number of transistors used by an n-bit APSQR function is:

$$\text{Number of Transistors} = 4 \cdot (n-1)\log_2(4 \cdot n) + 28 \cdot n + \delta \tag{9}$$

The first term corresponds to the number of transistors in each multiplexer and in this implementation, the multiplexers are composed of two bilateral switches (4 transistors). The term $28 \cdot n$ represents the number of transistors used in the design of modifier cells, and δ represents the number of transistors used in the controller design.

The number of transistors used in the cellular design is estimated as 4 transistors per multiplexer and 26 transistors per adder [9]. Thus, the number of transistors used in each cell of the conventional squaring function is 30, and the number of transistors used in an n-bit cellular squaring function is approximated as:

$$\text{Number of transistors} = (n^2 + n + 6)15 \tag{10}$$

Figure 4 shows the comparison between the two designs. A worst case number of transistors is estimated for the APSQR controller, $\delta = 200$. The dashed line represents the number of transistors used in the cellular design of the squaring function, and the solid line represents the number of transistors used in the APSQR function. With a 7-bit input, the APSQR is approximately 50% more area efficient than the cellular squaring function. As the number of input bits increases, the APSQR function becomes significantly more efficient. With a 16-bit input, the APSQR requires less than one-fourth the area of the conventional design.

The delay through the APSQR function is the sum of the delay through the controller, the delay through the multiplexers, and the delay through the modifier cell. Assuming the

controller is designed in two layers of logic, and the modifier cells are designed in three layers of logic, then the speed of the APSQR function can be approximated as 5-gate delays (plus a small delay through the tapered buffer at the output stage of the controller). This delay remains nearly constant despite the size of the input. For this reason, the speed of the APSQR is almost independent of the number of input bits. This is not the case with the conventional squaring scheme. The worst case delay is the propagation of the carry bit through the last chain of adders. Each adder introduces 2 gate delays. Thus for the number of input bits greater than 3, the APSQR function is faster than the conventional scheme. Additionally the APSQR architecture is inherently pipelinable.

5 Conclusion

The APSQR function is an appropriate squaring function for the Viterbi algorithm. The APSQR requires less hardware, and at same time, due to low input capacitance, its dynamic power consumption is less than the conventional squaring scheme. In addition, the APSQR function provides an improvement in the speed due to the shorter critical paths. Monte-Carlo simulations have shown negligible degradation in the performance of a Viterbi processor which utilizes the APSQR function.

References

[1] G.D. Forney,"The Viterbi algorithm," *Proc. of the IEEE*, vol. 61, pp. 268-276, March 1973.

[2] G. Ungerboeck,"Trellis-Coded modulation with redundant signal sets; part i: introduction," *IEEE Communications Magazine*, vol. 25, no.2, pp. 5-21, February 1987.

[3] G.D. Forney,"Trellis shaping," *IEEE Trans. on Information Theory*, vol.38, no.2, pp. 281-300, March 1992.

[4] M.W. Marcellin and T.R. Fisher,"Trellis coded quantization of memoryless and gauss-Markov source," *IEEE Trans. on Communication*, vol.38, no.1, pp. 82-93, January 1990.

[5] G. Fettweis and H. Meyr,"High-speed parallel decoding: algorithm and VLSI-architecture," *IEEE Communications Magazine*, pp.46-55, May 199.

[6] Bellman, R.E., and Dreyfus, S.E., "Applied dynamic programming," *Princeton University Press*, 1962.

[7] M. Shammanna, S. Whitaker and J. Canaris,"Cellular logic array for computation of squares," *3rd NASA Symposium on VLSI Design 1991*, pp. 2.4.1-2.4.7.

[8] K. Hwang, Computer Arithmetic: Principle, Architecture and Design, John Wiley and Sons, 1979.

[9] N. Zhuang and H. Wu,"A new design of the CMOS full adder," *IEEE J. Solid-State Circuits*, vol. 27, no.5, pp. 840-844, May 1992.

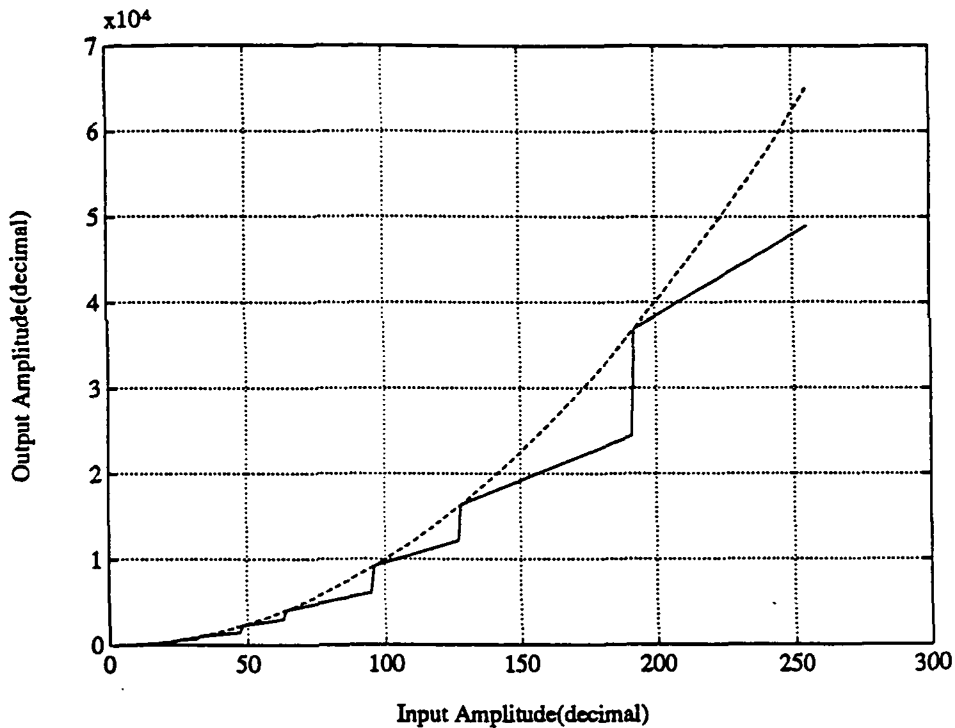

Figure 1: Simulation of approximated squaring function (solid) versus the actual squaring function (dashed).

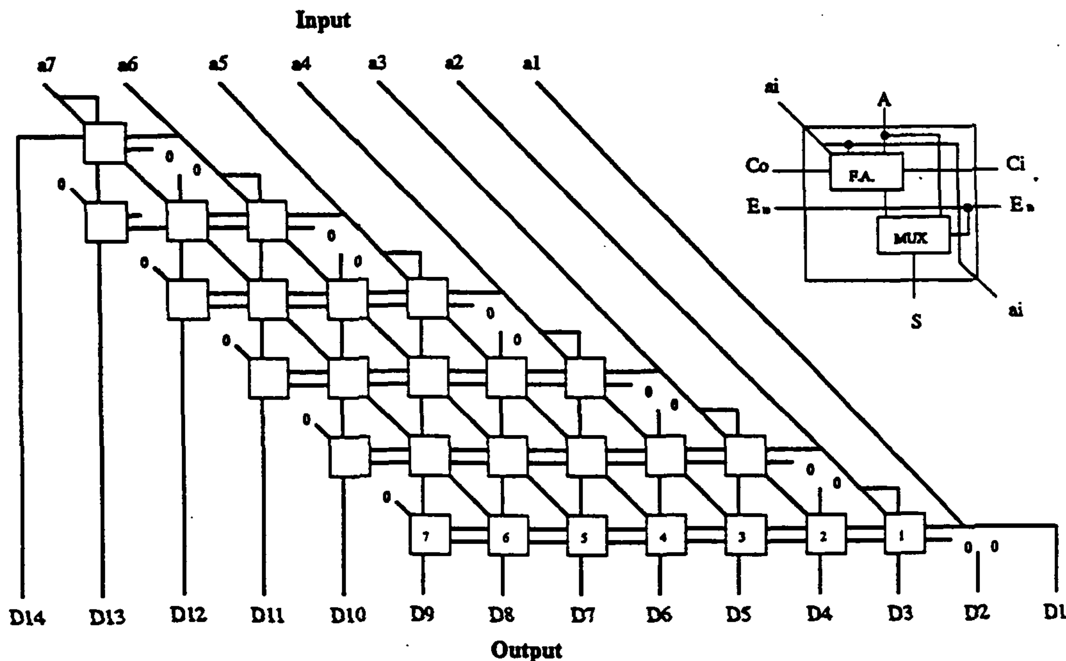

Figure 2: The block diagram of the conventional 7-bit squaring function.

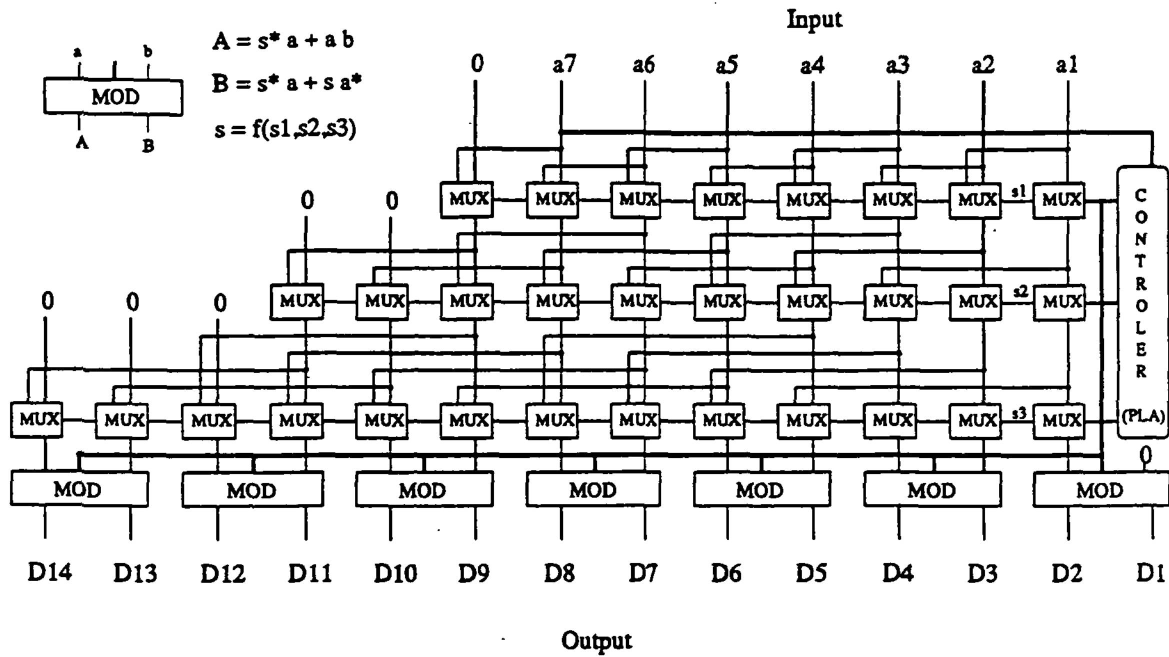

Figure 3: The block diagram of 7-bit approximated squaring function.

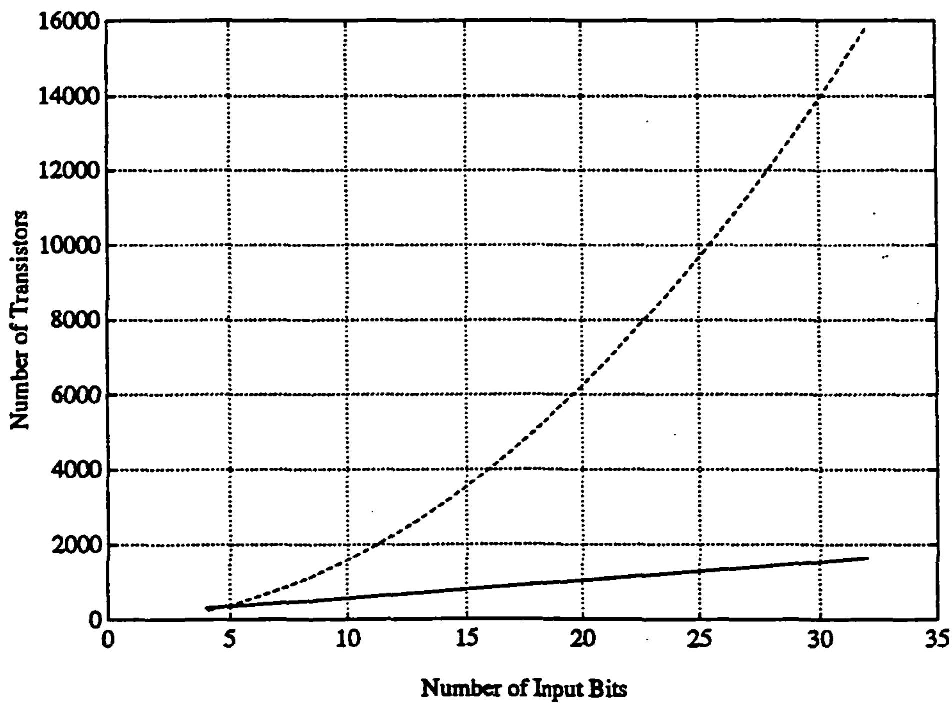

Figure 4: The numbers of transistors used in conventional technique (dashed line) and APSQR (solid line).