[21] V. Paretsky, "The role of hardware in exposing security breaches," (2005). [Online]. Available: http://www.ddj.com/print
[22] C. Percival, "Cache missing for fun and profit," (2005). [Online]. Available: http://www.daemonology.net/papers/htt.pdf
[23] R. J. Anderson and M. G. Kuhn, "Tamper resistance—A cautionary note," in *Proc. USENIX Workshop Electron. Commerce*, 1996, pp. 1–11.
[24] D. Farmer and W. Venema, "The coroner's toolkit," (2005). [Online]. Available: http://www.porcupine.org/forensics/tct.html
[25] J. Whittaker, "Why secure applications are difficult to write," *IEEE Security Privacy*, vol. 1, no. 2, pp. 81–83, Mar. 2003.
[26] Valgrind, "Valgrind memory profiler," [Online]. Available: http://www.valgrind.org
[27] *Federal Information Processing Standard*, 46-3, National Institute of Standards and Technology, 1999. [Online]. Available: http://www.csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf
[28] *"Xtensa Application Specific Microprocessor Solutions—Overview Handbook"* Tensilica Inc, (2001). [Online]. Available: http://www.tensilica.com
[29] B. Schneier, *Applied Cryptography: Protocols, Algorithms and Source Code in C*. New York: Wiley, 1996.
[30] N. Potlapally, A. Raghunathan, S. Ravi, N. Jha, and R. Lee, "Satisfiability-based framework for enabling side-channel attacks on cryptographic software," in *Proc. Des. Autom. Test Eur. Conf.*, 2006, pp. 18–23.
[31] C. M. Li, "Integrating equivalency reasoning into Davis-Putnam procedure," in *Proc. AAAI: 17th Nat. Conf. Artif. Intell.*, 2000, pp. 291–296.
[32] B. Selman, D. G. Mitchell, and H. J. Levesque, "Generating hard satisfiability problems," *Artif. Intell.*, vol. 81, no. 1–2, pp. 17–29, 1996.

High-Speed Recursion Architectures for MAP-Based Turbo Decoders

Zhongfeng Wang

***Abstract*—The maximum *a posterior* probability (MAP) algorithm has been widely used in Turbo decoding for its outstanding performance. However, it is very challenging to design high-speed MAP decoders because of inherent recursive computations. This paper presents two novel high-speed recursion architectures for MAP-based Turbo decoders. Algorithmic transformation, approximation, and architectural optimization are incorporated in the proposed designs to reduce the critical path. Simulations show that neither of the proposed designs has observable decoding performance loss compared to the true MAP algorithm when applied in Turbo decoding. Synthesis results show that the proposed Radix-2 recursion architecture can achieve comparable processing speed to that of the state-of-the-art recursion (Radix-4) architecture with significantly lower complexity while the proposed Radix-4 architecture is 32% faster than the best existing design.**

***Index Terms*—Error correction codes, high-speed design, maximum *a posterior* probability (MAP) decoder, Turbo code, VLSI.**

I. Introduction

Turbo code [1] invented in 1993, has attracted tremendous attentions in both academics and industry for its outstanding performance, rich applications can be found in wireless and satellite communications [2], [3]. Practical Turbo decoders usually employ serial decoding architectures [4] for area efficiency. Thus, the throughput of a Turbo decoder is highly limited by the clock speed and the maximum number of iterations to be performed. To facilitate iterative decoding, Turbo decoders require soft-input soft-output decoding algorithms, among which the maximum *a posterior* probability (MAP) algorithm [5] is widely adopted for its excellent performance.

Manuscript received March 3, 2006; revised August 27, 2006 and November 26, 2006.

The author is with the School of Electrical Engineering and Computer Science, Oregon State University, Corvallis, OR 97331 USA (e-mail: zwang@eecs.oregonstate.edu).

Digital Object Identifier 10.1109/TVLSI.2007.893668

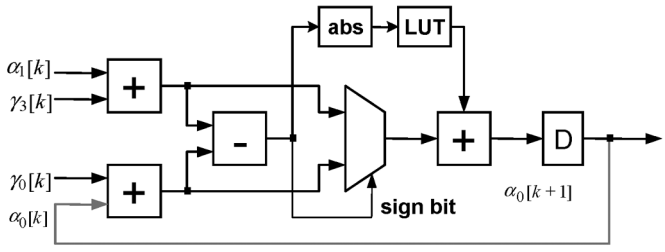

Fig. 1. Traditional recursion architecture: Arch-O.

Due to the recursive computations inherent with the MAP algorithm, the conventional pipelining technique is not applicable for raising the effective processing speed unless one MAP decoder is used to process more than one Turbo code blocks or sub-blocks as discussed in [6]. Among various high-speed recursion architectures in [6]–[10], the designs presented in [7] and [10] are most attractive. In [7], an offset-add-compare-select (OACS) architecture [8] is adopted to replace the traditional add-compare-select-offset (ACSO) architecture. In addition, the lookup table (LUT) is simplified with only 1-bit output, and the computation of absolute value is avoided through introduction of the reverse difference of two competing path metrics. An approximate 17% speedup over the traditional Radix-2 ASCO architecture was reported. With one-step look-ahead operation, a Radix-4 ACSO architecture can be derived. Practical Radix-4 architectures such as those presented in [9] and [10] always involve approximations in order to achieve higher effective speed-ups. For instance, the following approximation is adopted in [10]:

$$\max*(\max*(A,B),\max*(C,D)) \approx \max*(\max(A,B),\max(C,D)) \quad (1)$$

where

$$\max*(A,B) \equiv \max(A,B) + \log\left(1+e^{-|A-B|}\right). \quad (2)$$

This Radix-4 architecture can generally improve the processing speed (equals twice of its clock speed) by over 40% over the traditional Radix-2 architecture, and it has *de facto* the highest processing speed among all existing (MAP decoder) designs found in the literature. However, the hardware will be nearly doubled compared to the traditional ACSO [7] architecture.

The contributions of this paper include the following: 1) an advanced Radix-2 recursion architecture based on *algorithmic transformation, approximation and architectural level optimization*, which can achieve comparable processing speed as the state-of-the-art Radix-4 design with significantly lower complexity and 2) an improved Radix-4 architecture that is 32% faster than the best existing approach.

This paper is organized as follows. Section II presents an advanced Radix-2 recursion architecture for MAP decoders. Section III studies an improved Radix-4 recursion architecture. Section IV presents bit-error-rate performance comparisons between the original MAP algorithm and various approximations when applied in Turbo decoding. The synthesis results about hardware complexity and processing speed for various architectures are also provided in this section.

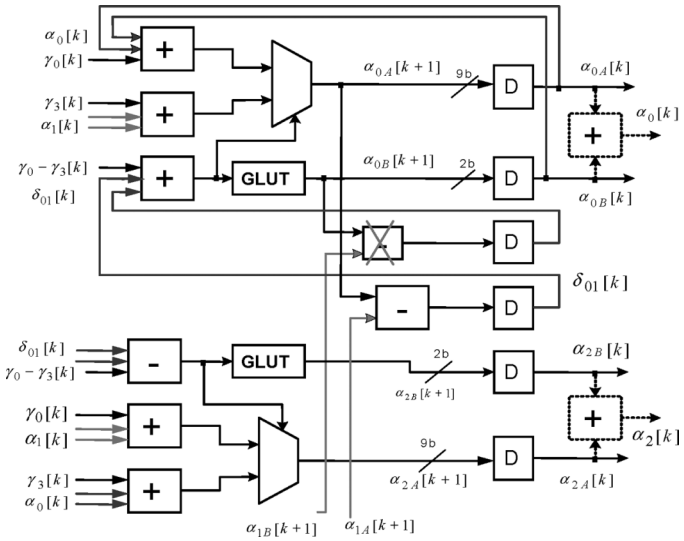

Fig. 2. Advanced Radix-2 fast recursion arch.: Arch-A.

II. Advanced High-Speed Radix-2 Recursion Architecture or MAP Decoders

For convenience in later discussion, we want to give a brief introduction of MAP-based Turbo decoder structure at the beginning of this section. The MAP algorithm is generally implemented in log domain and thus called Log-MAP algorithm. MAP-based Turbo decoders normally adopted a sliding window approach [11] in order to reduce computation latency and memory for storing state metrics. As it is explained in [4], three recursive computation units: $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$, and pre-$\boldsymbol{\beta}$ units are needed for a Log-MAP decoder. This paper is focused on the design of high-speed recursive computation units as they form the bottleneck in high speed circuit design.

It is known from the Log-MAP algorithm that all three recursion units have similar architectures. So we will focus our discussion on the design of $\boldsymbol{\alpha}$ units. The traditional design for $\boldsymbol{\alpha}$ computation is illustrated in Fig. 1, where the **ABS** block is used to compute the absolute value of the input and the **LUT** block is used to implement a nonlinear function $\log(1+e^{-x}))$, where $x > 0$. For simplicity, only one branch (i.e., one state) is drawn. The overflow approach [14] is assumed for normalization of state metrics as used in conventional Viterbi decoders.

It can be seen that the computation of the recursive loop consists of three multibit additions, the computation of absolute value and a random logic to implement the LUT. As there is only one delay element in each recursive loop, the traditional retiming technique [12] cannot be used to reduce the critical path.

In this paper, we propose an advanced Radix-2 recursion architecture shown in Fig. 2. Here, we first introduce a difference metric for each competing pair of states metrics (e.g., $\alpha 0$ and $\alpha 1$ in Fig. 2) so that we can perform the front-end addition and the subtraction operations simultaneously in order to reduce the computation delay of the loop. Second, we employ a generalized LUT (see GLUT in Fig. 2) that can efficiently avoid the computation of absolute value instead of introducing another subtraction operation as in [7]. Third, we move the final addition to the input side as with the OACS architecture [8] and then utilize one stage carry-save structure to convert a three-number addition to a two-number addition. Finally, we make an intelligent approximation in order to further reduce the critical path.

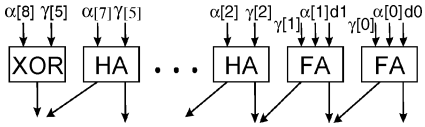

Fig. 3. Carry–save structure in the front end of Arch-A.

The following equations are assumed for the considered recursive computation shown in Fig. 2:

$$\begin{aligned}\alpha_0[k+1] &= \max*(\alpha_0[k]+\gamma_0[k], \alpha_1[k]+\gamma_3[k])\\ \alpha_2[k+1] &= \max*(\alpha_0 k]+\gamma_3[k], \alpha_1[k]+\gamma_0[k])\end{aligned} \tag{3}$$

where $\max^*$ function is defined in (2).

In addition, we split each state metric into two terms as follows:

$$\begin{aligned}\alpha_0[k] &= \alpha_{0A}[k]+\alpha_{0B}[k]\\ \alpha_1[k] &= \alpha_{1A}[k]+\alpha_{1B}[k]\\ \alpha_2[k] &= \alpha_{2A}[k]+\alpha_{2B}[k].\end{aligned} \tag{4}$$

Similarly, the corresponding difference metric is also split into the following two terms:

$$\begin{aligned}\delta_{01}[k] &= \delta_{01A}[k]+\delta_{01B}[k]\\ \delta_{01A}[k] &= \alpha_{0A}[k]-\alpha_{1A}[k]\\ \delta_{01B}[k] &= \alpha_{0B}[k]-\alpha_{1B}[k].\end{aligned} \tag{5}$$

In this way, the original add-and-compare operation is converted as an addition of three numbers, i.e.,

$$(\alpha_0+\gamma_0)-(\alpha_1+\gamma_3) = (\gamma_0-\gamma_3)+\delta_{01A}+\delta_{01B} \tag{6}$$

where $(\gamma_0-\gamma_3)$ is computed by branch metric unit (BMU), the time index $[k]$ is omitted for simplicity. In addition, the difference between the two outputs from two GLUTs, i.e., δ_{01B} in the figure, can be neglected. From extensive simulations, we found that this small approximation does not cause any performance loss in Turbo decoding with either AWGN channels or Raleigh fading channels. This fact is simply explained in the following. If one competing path metrics (e.g., $p0 = \alpha 0+\gamma 0$) is significantly larger than the other one (e.g., $p1 = \alpha 1+\gamma 3$), the GLUT output will not change the decision anyway due to their small magnitudes. On the other hand, if the two competing path metrics are so close that adding or removing a small value output from one GLUT may change the decision (e.g., from $p0 > p1$ to $p1 > p0$), picking any survivor should not make big difference.

At the input side, a small circuitry shown in Fig. 3 is employed to convert an addition of three numbers to an addition of two numbers, where FA and HA represents full-adder and half-adder, respectively, XOR stands for exclusive OR gate, d0 and d1 correspond to the 2-bit output of GLUT. The state metrics and branch metrics are represented with 9 and 6 bits, respectively, in this example. The sign extension is only applied to the branch metrics. It should be noted that an extra addition operation (see dashed adder boxes) might be required to integrate each state metric before storing it into the α memory.

The GLUT structure is shown in Fig. 4, where the computation of absolute value is eliminated by including the sign bit into two logic blocks, i.e., **Ls2** and ELUT, where the **Ls2** function block is used to detect if the absolute value of the input is less than 2.0, and the ELUT block is a small LUT with 3-bit inputs and 2-bit outputs. It can be derived that $Z = \overline{S}\,\overline{b_7,\ldots,+b_1+b} + S(b_7,\ldots,b_1b_0)$. It was reported

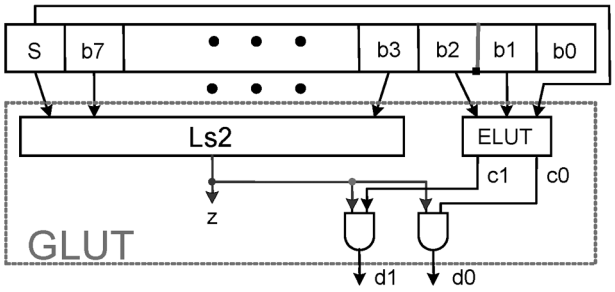

Fig. 4. Structure of GLUT used in Arch-A.

TABLE I
PROPOSED LUT APPROXIMATION

|x|	0.0	0.50	1.0	1.50
f(x)	3⁄4	2/4	1⁄4	1⁄4

in [13] that using two output values for the LUT only caused a performance loss of 0.03 dB from the floating point simulation for a four-state Turbo code. The approximation is described as follows:

$$\text{If } |x| < 2,\ f(x) = 3/8, \qquad \text{else } f(x) = 0 \tag{7}$$

where x and $f(x)$ stand for the input and the output of the LUT, respectively. In this approach, we only need to check if the absolute value of the input is less than 2, which can be performed by the **Ls2** block in Fig. 4. A drawback of this method is that its performance would be significantly degraded if only two bits are kept for the fractional part of the state metrics, which is generally the case.

In our design, both the inputs and outputs of the LUT are quantized in four levels. The details are shown in Table I. The inputs to ELUT are treated as a 3-bit signed binary number. The outputs of ELUT are ANDed with the output of **Ls2** block. This means, if the absolute value of the input is greater than 2.0, the output from the GLUT is 0. Otherwise, the output from ELUT will be the final output. The ELUT can be implemented with combinational logic for high-speed applications. The computation latency is smaller than the latency of **Ls2** block. Therefore, the overall latency of the GLUT is almost the same as the previously discussed simplified method whose total delay consists of one 2:1 multiplexer gate delay and the computation delay of logic block **Ls2**.

After all the previous optimization, the critical path of the recursive architecture is reduced to two multibit additions, one 2:1 MUX operation, and 1-bit addition operation, which saves nearly two multibit adder delay compared to the traditional ACSO architecture. We will show detailed comparisons in Section IV.

III. IMPROVED RADIX-4 ARCHITECTURE FOR MAP DECODERS

In the following, we discuss an improved Radix-4 recursion architecture. The computation for $\alpha_0[k+2]$ is expressed as follows:

$$\begin{aligned}\alpha_0[k+2] &= \max*(\alpha_0[k+1]+\gamma_0[k+1], \alpha_1[k+1]+\gamma_3[k+1]) \\ &= \max*(\max*(\alpha_0[k]+\gamma_0[k], \alpha_1[k]+\gamma_3[k]) \\ &\quad + \gamma_0[k+1]), \max*(\alpha_1[k]+\gamma_2[k], \alpha_3[k]+\gamma_1[k]) \\ &\quad +\gamma_3[k+1]) \end{aligned} \tag{8}$$

where $\alpha_1[k+1] = \max^*(\alpha_2[k]+\gamma_2[k], \alpha_3[k]+\gamma_1[k])$.

In [10], Lucent Bell Labs proposed the following approximation:

$$\begin{aligned}\alpha_0[k+2] \approx \max*(&\max(\alpha_0[k]+\gamma_0[k], \alpha_1[k]+\gamma_3[k]) \\ &+\gamma_0[k+1], \max(\alpha_2[k]+\gamma_2[k], \alpha_3[k]+\gamma_1[k])+\gamma_3[k+1]). \end{aligned} \tag{9}$$

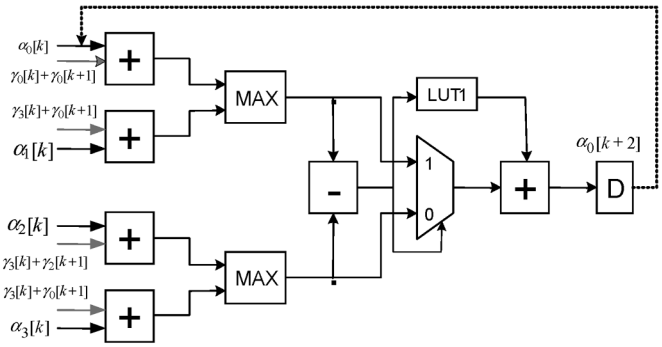

Fig. 5. Radix-4 architecture proposed by Lucent Bell Labs: Arch-L.

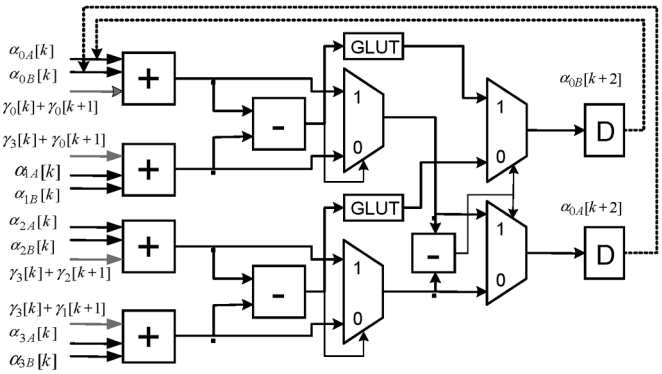

Fig. 6. Improved Radix-4 recursion arch.: Arch-B.

This approximation is reported to have a 0.04-dB performance loss compared to the original Log-MAP algorithm. The architecture to implement the previous computation is shown in Fig. 5 for convenience in later discussion. As it can be seen, the critical path consists of four multibit adder delays, one generalized LUT delay, and one 2:1 MUX delay (Note: the LUT1 block includes absolute value computation and a normal LUT operation; the MAX block includes one subtractor and one 2:1 MUX).

Similarly, we can take an alternative approximation as follows:

$$\begin{aligned}\alpha_0[k+2] \approx \max(&\max*(\alpha_0[k]+\gamma_0[k], \alpha_1[k]+\gamma_3[k])+\gamma_0[k+1], \\ &\max*(\alpha_2 k] + \gamma_2[k], \alpha_3[k] + \gamma_1[k])+ \gamma_3[k+1]). \end{aligned} \tag{10}$$

Intuitively, Turbo decoder employing this new approximation should have the same decoding performance as using (8). While directly implementing (10) does not bring any advantage to the critical path, we intend to take advantages of the techniques that we developed in Section II. The details are shown in Fig. 6. Here, we split each state metric into two terms and we adopt the same GLUT structure as we did before. In addition, a similar approximation is incorporated as with Arch-A. In this case, the outputs from GLUT are not involved in the final stage comparison operation. It can be observed that the critical path of the new architecture is close to a three multibit adder delay. To compensate for all the approximation introduced, the extrinsic information generated by the MAP decoder based on this new Radix-4 architecture should be scaled by a factor around 0.75.

IV. PERFORMANCE COMPARISONS

For quantitative comparison in hardware complexity and processing speed, we used TSMC 0.18-μm standard cells to synthesize one α unit

TABLE II
COMPARISON FOR VARIOUS RECURSION ARCH'S

	Max Clock Freq. (Mhz)	Relative Area	Relative Proc. Speed
Arch-O	241	1.0	1.0
Arch-R	333	0.87	1.38
Arch-U	335	1.14	1.39
Arch-L	182	1.82	1.51
Arch-A	370	1.03	1.54
Arch-B	241	1.99	2.0

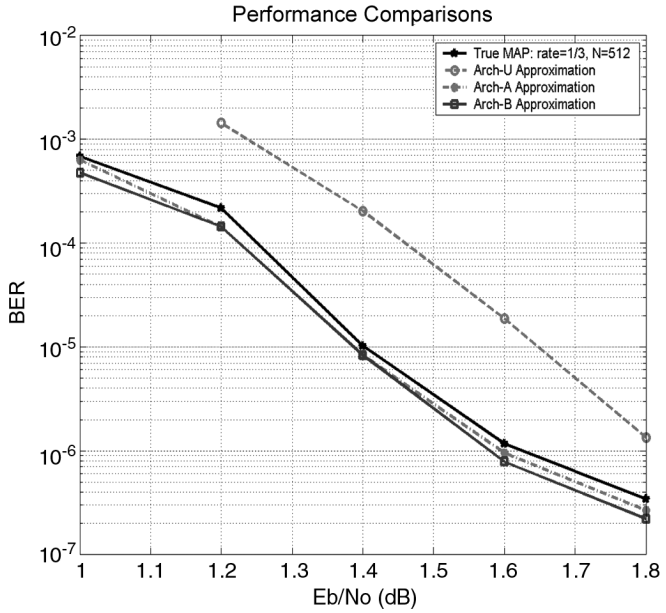

Fig. 7. Performance comparison among the original MAP and two approximated algorithms.

for six different recursion architectures, i.e., 1) the traditional ACSO architecture: **Arch-O**; 2) a reference Radix-2 architecture that employs GLUT and OASC structure: **Arch-R**; 3) the reduced-precision Radix-2 recursion architecture presented in [7]: **Arch-U**; 4) the Radix-4 architecture proposed by Lucent: **Arch-L**; 5) the advanced Radix-2 recursion architecture presented in this work: **Arch-A**; and 6) the improved Radix-4 recursion architecture: **Arch-B**. All the state metrics are quantized as 9 bits while the branch metrics are represented using 6 bits. The synthesis is optimized for speed. The detailed results are listed in Table II.

It can be seen from Table II that the proposed Radix-2 architecture is significantly faster than the conventional Radix-2 architecture. The speedup can be attributed to three factors which are: 1) using GLUT to reduce delay of computing absolute value; 2) moving the offset addition to the front end and employing 3 to 2 compressor to reduce the overall addition delay in the loop; and 3) introducing differential state metrics to reduce delay of comparing two competing path metrics. It can also be observed that neither the speed of Arch-R nor that of Arch-U benefits from the third factor. That is why the proposed Radix-2 architecture is faster than the other two Radix-2 architectures. In fact, the new Radix-2 architecture has comparable processing speed as the Radix-4 architecture proposed by Lucent while having significantly lower complexity. It can be calculated that the new Radix-4 architecture is 32% faster than Lucent Radix-4 architecture with only 9% hardware overhead. This amount of extra hardware is negligible compared to an entire Turbo decoder.

We have performed extensive simulations for Turbo codes using the original MAP and various approximations. Fig. 7 shows the BER performance of a rate-1/3, 8-state, block size of 512 bits, Turbo code using different MAP architectures. The simulations were completed under the assumption of AWGN channel and BPSK signaling. A maximum of eight iterations were performed. More than 40 million random information bits were simulated for both $\mathrm{Eb/No} = 1.6$ dB and $\mathrm{Eb/No} = 1.8$ dB cases. It can be noted from Fig. 8 that there is no observable performance difference between the true MAP algorithm and two approximation methods associated with the proposed recursion architectures while the approximation employed in [7] caused 0.2–0.25-dB performance degradation.

We argue that the proposed Radix-4 recursion architecture is optimal for high-speed MAP decoders. Any (significantly) faster recursion architecture (e.g., a possible Radix-8 architecture) will be at the expense of exponentially increased hardware. On the other hand, when the target throughput is moderate, these fast recursion architectures can be used to reduce power consumption because of their significantly reduced critical paths.

V. CONCLUSION

Through introduction of a difference metric for each pair of competing states, we have successfully reduced the computation delay of the state metric comparison operation. After rearranging the final addition operation for each state metric, we have almost saved a multibit adder delay on the critical path by utilizing a carry-save structure in the front end of each recursive loop. The computation of absolute value for each LUT operation has been efficiently eliminated in the proposed GLUT structure. Synthesis results have shown that the proposed Radix-2 recursion architectures is as fast as the start-of-the-art Radix-4 recursion architecture while having significantly lower complexity while the improved Radix-4 architecture is 32% faster than any fast recursion architecture in the literature. It has been verified with simulation that approximations involved in the proposed design do not cause any (BER) performance degradation.

REFERENCES

[1] C. Berrou, A. Clavieux, and P. Thitimajshia, "Near Shannon limit error correcting coding and decoding: Turbo codes," in *Proc. ICC*, 1993, pp. 1064–1070.
[2] *"Technical Specification Group Radio Access Network, Multiplexing and Channel Coding (TS 25.212 Version 3.0.0)"* 3rd Generation Partnership Project (3GPP) [Online]. Available: http://www.3gpp.org
[3] 3rd Generation Partnership Project 2 (3GPP2) [Online]. Available: http://www.3gpp2.org
[4] H. Suzuki, Z. Wang, and K. K. Parhi, "A $K = 3$, 2 Mbps low power Turbo decoder for 3rd generation W-CDMA systems," in *Proc. IEEE Custom Integr. Circuits Conf. (CICC)*, 2000, pp. 39–42.
[5] L. Bahl, J. Jelinek, J. Raviv, and F. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Trans. Inf. Theory*, vol. IT-20, no. 2, pp. 284–287, Mar. 1974.
[6] S.-J. Lee, N. Shanbhag, and A. Singer, "A 285-MHz pipelined MAP decoder in 0.18 μm CMOS," *IEEE J. Solid-State Circuits*, vol. 40, no. 8, pp. 1718–1725, Aug. 2005.
[7] P. Urard *et al.*, "A generic 350 Mb/s Turbo codec based on a 16-state Turbo decoder," in *IEEE ISSCC Dig. Tech. Papers*, 2004, pp. 424–433.
[8] E. Boutillon, W. Gross, and P. Gulak, "VLSI architectures for the MAP algorithm," *IEEE Trans. Commun.*, vol. 51, no. 2, pp. 175–185, Feb. 2003.
[9] T. Miyauchi, K. Yamamoto, and T. Yokokawa, "High-performance programmable SISO decoder VLSI implementation for decoding Turbo codes," in *Proc. IEEE Global Telecommun. Conf.*, 2001, pp. 305–309.
[10] M. Bickerstaff, L. Davis, C. Thomas, D. Garret, and C. Nicol, "A 24 Mb/s radix-4 LogMAP Turbo decoder for 3 GPP-HSDPA mobile wireless," in *IEEE ISSCC Dig. Tech. Papers*, 2003, pp. 150–151.

[11] A. J. Viterbi, "An intuitive justification of the MAP decoder for convolutional codes," *IEEE J. Sel. Areas Commun.*, vol. 16, pp. 260–264, Feb. 1998.
[12] T. C. Denk and K. K. Parhi, "Exhaustive scheduling and retiming of digital signal processing systems," *IEEE Trans. Circuits Syst., Part II: Analog Dig. Signal Process.*, vol. 45, no. 7, pp. 821–838, Jul. 1998.
[13] W. Gross and P. G. Gulak, "Simplified MAP algorithm suitable for implementation of turbo decoders," *Electron. Lett.*, vol. 34, no. 16, pp. 1577–1578, Aug. 1998.
[14] Y. Wu, B. D. Woerner, and T. K. Blankenship, "Data width requirement in SISO decoding with module normalization," *IEEE Trans. Commun.*, vol. 49, no. 11, pp. 1861–1868, Nov. 2001.

A Flexible Architecture for Precise Gamma Correction

Dong-U Lee, Ray C. C. Cheung, and John D. Villasenor

***Abstract*—We present a flexible hardware architecture for precise gamma correction via piece-wise linear polynomial approximations. Arbitrary gamma values, input bit widths, and output bit widths are supported. The gamma correction curve is segmented via a combination of uniform segments and segments whose sizes vary by powers of two. This segmentation method minimizes the number of segments required, while providing an efficient way for indexing the polynomial coefficients. The outputs are guaranteed to be accurate to one unit in the last place through an analytical bit-width analysis methodology. Hardware realizations of various gamma correction designs are demonstrated on a Xilinx Virtex-4 field-programmable gate array (FPGA). A pipelined 12-bit input/8-bit output design on an XC4VLX100-12 FPGA occupies 146 slices and one digital signal processing slice. It is capable of performing 378 million gamma correction operations per second.**

***Index Terms*—Displays, field programmable gate arrays (FPGAs), fixed-point arithmetic, video signal processing.**

I. Introduction

The term "gamma" originates from the nonlinear responses of cathode ray tubes (CRTs) caused by electrostatic effects in the electron gun. The luminance produced by CRTs is not linearly proportional to the input voltage. Instead, the produced luminance L is proportional to the input voltage V raised by a power gamma γ

$$L = V^{\gamma} \tag{1}$$

where V is normalized over zero and one. In order to compensate this nonlinearity, gamma correction is performed on the input signal to achieve correct production of the luminance on the display [1]. Fig. 1 illustrates the signal flow in a typical display system. The most commonly used value for gamma correction is $\gamma = 1/0.45$ which is the basis of various standards including the ITU Rec. 709 specification [1].

Manuscript received April 27, 2006; revised November 21, 2006. This work was supported in part by the National Science Foundation under Grant CCR-0120778 and Grant CCF-0541453, by the Office of Naval Research under Contract N00014-06-1-0253, and by the Croucher Foundation.

D. Lee and J. D. Villasenor are with the Electrical Engineering Department, University of California, Los Angeles, CA 90095 USA (e-mail: dongu@icsl.ucla.edu; villa@icsl.ucla.edu).

R. C. C. Cheung is with the Department of Computing, Imperial College London, London, SW7 2BZ U.K. (e-mail: r.cheung@imperial.ac.uk).

Digital Object Identifier 10.1109/TVLSI.2007.893671

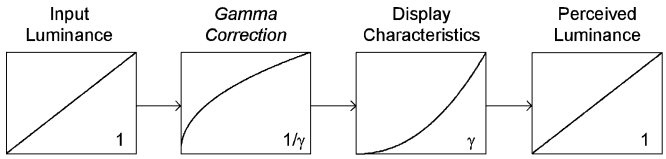

Fig. 1. Signal flow in a typical display system.

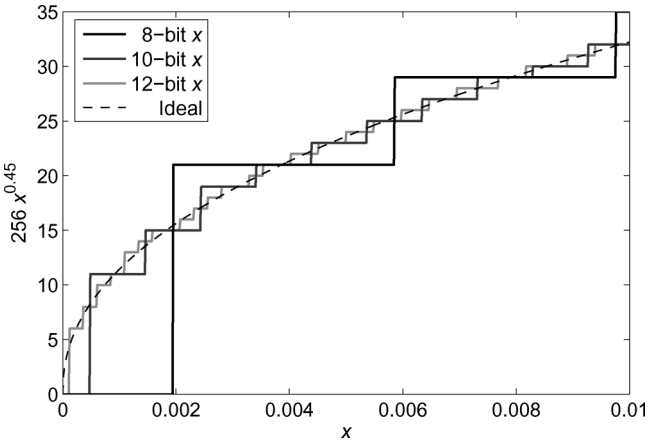

Fig. 2. Quantization effects of gamma correction in low luminance regions for different input x bit widths with the output fixed at eight bits. A gamma of 1/0.45 is assumed.

Although modern display devices such as liquid crystal displays (LCDs) and plasma display panels (PDPs) are inherently linear, gamma correction is still performed for perceptual coding purposes [2]. This is related to the fact that inverse of the CRT's transfer function is remarkably similar to the perceptual uniformity of human vision given by the Weber–Fechner law [3].

The most straightforward realization of gamma correction involves direct table lookups [4], [5]. For a typical display system that supports 8 bits per component, an 8-bit input/8-bit output table is often placed before or after the frame buffer. This approach is simple to implement and requires a table size of just $2^8 \times 8 = 2048$ bits. However, using eight bits to select the table entry can cause significant banding (or contouring) in the low luminance regions. This is because in these regions the gamma correction curve has a slope greater than one, and thus maps quantization step sizes at the input to larger step sizes at the output. When these transformed step sizes are overlayed on the inherent step sizes available at the output, the ability to generate certain luminance levels at the output can be lost. To fully exploit the output resolution in the presence of gamma correction, a higher input resolution is needed. This is illustrated in Fig. 2 which shows gamma correction in the low luminance regions of 8-, 10-, and 12-bit inputs with the output fixed at eight bits. The lower input resolutions create large luminance jumps in the low luminance regions at the display device, causing banding artifacts.

While the severity of these artifacts can be lessened by using more bits at the input, this leads to an exponential increase in storage if a direct lookup table is used. For instance, a 12-bit input/8-bit output table has a memory requirement of 32 768 bits, which can be problematic for resource constrained platforms. For example, many medical imaging displays support 12 bits per component and a recent paper by Kim *et al.* [6] describes an LCD display with ten bits per component. Such high-end displays impose even more stringent requirements on