A 4-GHz 130-nm Address Generation Unit With 32-bit Sparse-Tree Adder Core

Sanu Mathew, *Member, IEEE*, Mark Anders, *Member, IEEE*, Ram K. Krishnamurthy, *Member, IEEE*, and Shekhar Borkar, *Member, IEEE*

***Abstract*—This paper describes a 32-bit address generation unit designed for 4-GHz operation in 1.2-V 130-nm technology. The AGU utilizes a 152-ps sparse-tree adder core to achieve 20% delay reduction, 80% lower interconnect complexity, and a low (1%) active energy leakage component. The dual-V_T semidynamic implementation of the adder core provides the performance of a dynamic CMOS design with an average energy profile similar to static CMOS, enabling 71% savings in average energy with a good sub-130-nm scaling trend.**

***Index Terms*—Address generation unit (AGU), high-performance adders, semidynamic design, sparse-tree adder.**

I. Introduction

HIGH-PERFORMANCE microprocessors use a variety of memory management techniques to map a logical address to a physical address space [1]. These techniques include features such as segmentation and paging, which allow memory to be managed efficiently and reliably. The address generation unit (AGU) is a key component of the memory management block and is used to compute the effective address of the location being addressed in memory. This operation is defined as

$$Effective\ Address = (Segment + Displacement) + Base + (Index * Scale).$$

Of the five operands involved in effective address computation, two operands, *Segment* and *Displacement*, are available ahead of time. Therefore, their sum may be precomputed in the previous cycle, reducing the AGU operation to a three-operand 32-bit addition. *Base* and *Index* addresses are register operands that are available at the start of the cycle and *Scale* is a value of 1, 2, 4, or 8 that is multiplied to the *Index* address. A variety of addressing modes can be implemented by choosing appropriate values for each of the five address components [1].

Effective address computation is a performance-critical single-cycle operation that requires a high-performance AGU. However, the high activity of the AGU creates thermal hotspots and sharp temperature gradients [2] in the execution core (Fig. 1) that can severely affect circuit reliability and increase cooling costs. The presence of multiple execution engines in current processors [3] further aggravates the problem. Therefore, there is a strong motivation for designing high-performance energy-efficient AGUs which satisfy the single-cycle performance requirement while reducing peak and average power dissipation.

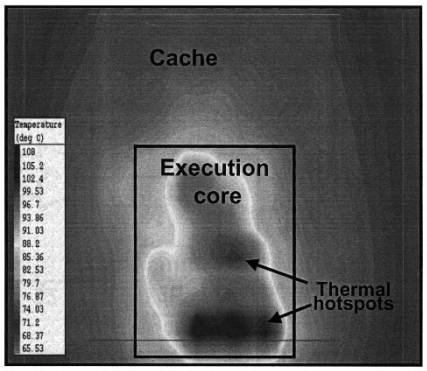


Fig. 1. Thermal map of a typical superscalar processor showing hot-spots over the AGU.

The remainder of this paper is organized as follows. Section II describes the major blocks in the AGU. In Section III, we look at an existing high-performance adder architecture and highlight its limitations for use in the AGU. We present our sparse-tree adder core design in Section IV and describe unique features of this design that enabled an energy-efficient AGU implementation. The semidynamic design of the adder core is described in Section V. Section VI presents the simulation results and scaling trends of this design. Finally, we conclude the paper in Section VII.

II. AGU Architecture

There are three main blocks in the AGU: a programmable 3-bit *Index* shifter, an array of $3:2$ compressors and a high-performance 32-bit adder (Fig. 2). In addition, a precompute block is used to calculate the sum of *Segment* and *Displacement* addresses. Effective address computation occurs in two phases, with the negative-level latches at the input and the output of the AGU representing the cycle boundaries. The clock period is divided into two phases by the positive-level latch that is incorporated into the first dynamic stage of the adder core. (A clocked footer transistor added to the pulldown evaluation stack of a dynamic gate converts it to a dynamic latch).

In the first phase ($clk = 0$), the input latches are transparent. The three inputs (*Base*, *Scaled-Index*, and the precomputed sum of *Segment* and *Displacement* addresses) propagate through the input latches and are added using a static $3:2$ compressor. The scaling of the *Index* address is performed by a left-shift of 0, 1, 2, or 3 bits using a programmable transmission-gate

Manuscript received August 15, 2002; revised December 20, 2002.

The authors are with the Circuit Research Laboratories, Intel Corporation, Hillsboro, OR 97124 USA (e-mail: sanu.k.mathew@intel.com; mark.a.anders@intel.com; ram.krishnamurthy@intel.com; shekhar.borkar@intel.com).

Digital Object Identifier 10.1109/JSSC.2003.810056

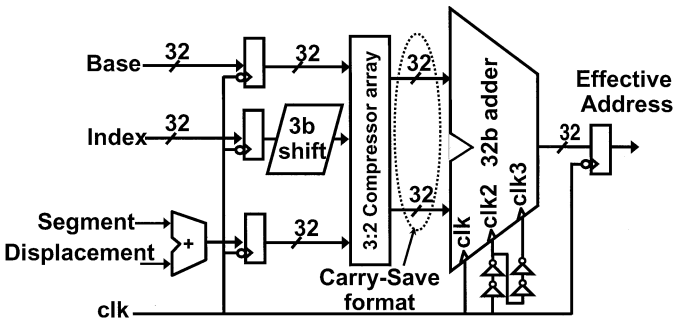


Fig. 2. AGU architecture.

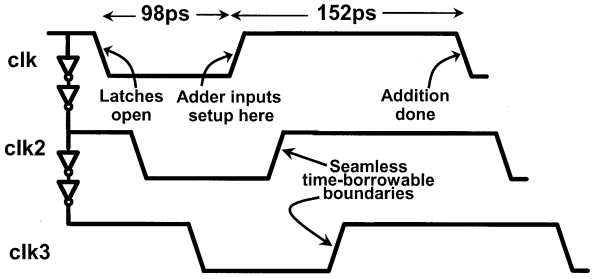


Fig. 3. AGU timing diagram.

multiplexer-based shifter circuit. The 3 : 2 compressor is implemented using a static transmission-gate XOR-based full-adder circuit that adds three input bits, generating a sum and carry output. Thus, the output of the compressor array is a pair of 32-bit numbers that represent the carry–save format of the effective address. This phase of address computation includes the latch data-to-Q delay, shifter delay, 3 : 2 compressor delay and setup time at the adder core inputs and takes 98 ps in a 1.2-V 130-nm technology [4]. During this phase, the adder core will be in the precharge state.

In the next phase, the rising edge of the clock disables the input and output latches and puts the adder in evaluation mode. The 32-bit adder core converts the carry–save output of the compressor into two's-complement format, delivering the final effective address at the end of the clock cycle. Staggered clocks (*clk2* and *clk3*) are used to avoid precharge contention and present seamless time-borrowable boundaries (Fig. 3) within the core. These clocks are locally generated in the adder core by delaying the main clock *clk* using a pair of double-inverter buffers. A target frequency of 4 GHz leaves the designer with 152 ps for the second phase of address computation, requiring a high-performance 6.6-GHz adder core. Next, we will explore design choices for building a 32-bit adder core.

III. High-Performance 32-bit Adders

A well-known high-performance adder architecture is the Kogge-Stone (KS) adder [5], [6]. This adder uses parallel prefix logic that employs a logarithmic binary carry–merge tree to generate carries for every adder bit. As shown in Fig. 4, even and odd bitslices of the KS adder core are identical and include the PG block followed by five stages of carry–merge (two dynamic gates [CM2 and CM4] interspersed between three static gates

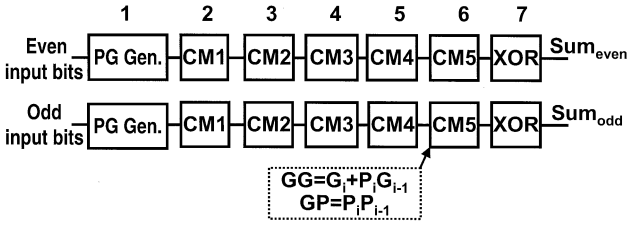


Fig. 4. Critical paths of a 32-bit Kogge–Stone adder.

[CM1, CM3, CM5]) that perform radix-2 carry–merge operation in both the static and dynamic stages. The first stage of the adder is the Propagate–Generate (PG) block that generates *propagate* ($P_i = A_i + B_i$) and *generate* ($G_i = A_i \cdot B_i$) signals from the adder inputs $A_i\#$ and $B_i\#$. This block, implemented in single-rail dynamic logic, has a worst-case evaluation stack of 3-nMOS (including the clocked footer transistor) and is followed by the static carry–merge block CM1 that generates the two-way group generate ($GG_i = G_i + P_iG_{i-1}$) and group-propagate ($GP_i = P_iP_{i-1}$) signals. The output of CM1 will predischarge low (since its inputs are precharged high) and has a worst-case 2-pMOS pullup evaluation path. Thus, the KS adder has a worst-case evaluation path of 3N-2P-2N-2P-2N-2P to generate the carry. Finally, an XOR operation of the partial sum with the generated carry delivers the final sum. Thus, a 32-bit KS adder generates all 32 carries in parallel using a full-blown carry–merge tree, resulting in a critical path of seven gate stages with generate and propagate fanouts of 2 and 3, respectively, and a maximum interconnect that spans 16 bitslices. The high fanout, combined with the high interconnect complexity (Fig. 5) makes the KS design an energy-inefficient implementation. We will now propose a sparse-tree adder core [7] that improves on this design.

IV. Sparse-Tree Adder Architecture

A. Critical Sparse-Tree Circuit

Contrary to the dense carry–merge tree approach of the KS adder (Fig. 5), we propose a sparse-tree adder that divides the carry–merge tree into critical and noncritical sections. The purpose is to speed up the critical path by moving a portion of the carry–merge logic to a noncritical sidepath. Instead of generating the carry for each bit ($C_0, C_1, \ldots C_{30}, C_{31}$), as in the KS approach, the sparse-tree adder (Fig. 6) generates every fourth carry ($C_3, C_7, \ldots C_{23}$, and C_{27}). Consequently, the critical path reduces to a pruned-down carry–merge tree that consists of a PG generator followed by five stages of carry–merge logic, resulting in a worst-case evaluation path of 3N-2P-2N-2P-2N-2P (same as KS). Note that the carry–merge tree has been pruned down to the maximum extent possible while keeping the total number of logic stages the same as in a KS tree (five stages), resulting in *generate* and *propagate* fanouts of 1 and 2, respectively. Thus, the sparse-tree approach leads to 33%/50% reduction in PG fanouts per stage and a 25% reduction in maximum inter-stage interconnect length (spans 12 bits versus 16 bits in the KS design). Further, the 80% reduction in wiring complexity permits the use of wider/shielded wires on the few

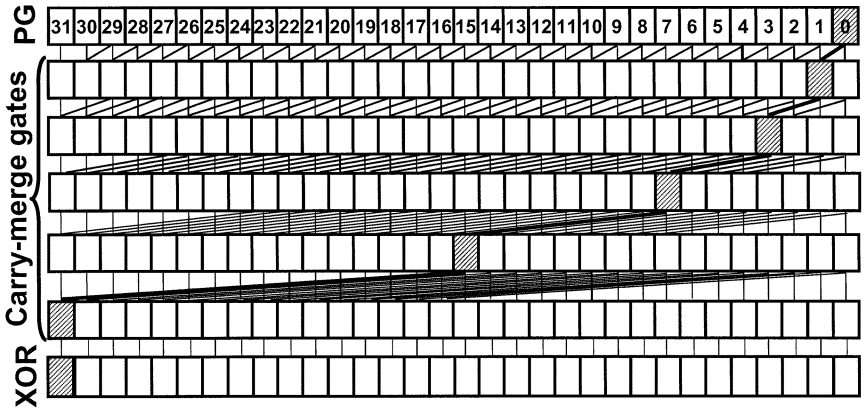


Fig. 5. 32-bit KS adder. High gate density and wiring complexity.

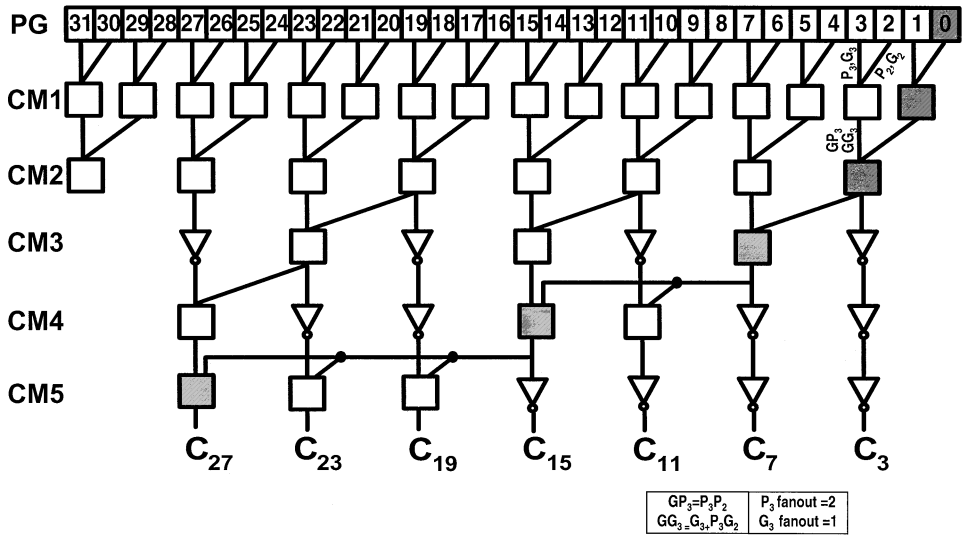


Fig. 6. Critical sparse carry–merge tree.

performance-critical inter-stage group generate/propagate signals. Thus, we have sped up the critical path of the adder by retaining the same number of gate stages as the KS architecture with reduced fanouts/stage and reduced interconnect *RC* between stages.

Sparse trees have been used to design silicon-on-insulator (SOI)-optimal adder architectures using taller transistor stacks to reduce the logic depth without increasing stage fanouts [8]. However, this being a bulk CMOS design, the stack height was limited to 2P in the static stages to minimize the stack penalty, due to body effect in series-connected devices. Different configurations of the sparse tree are possible by trading off gate fanouts, interconnect complexity, and number of logic stages. For instance, the Ladner–Fischer adder [9] reduces interconnect complexity at the cost of exponentially increasing fanouts of 1, 2, 4, 8, and 16 in the critical sections of the carry-chain. In our proposed design, we opt for an irregular sparse tree where the fanout on two carry–merge gates ($GG_{7:0}$ in CM_3 and $GG_{15:0}$ in CM_4) is increased while keeping fanouts on the remaining 121 generate/propagate gates to 1 and 2, respectively. This allows us to achieve the same logic depth of a KS adder and simultaneously retain the low-power advantages of a sparse-tree design.

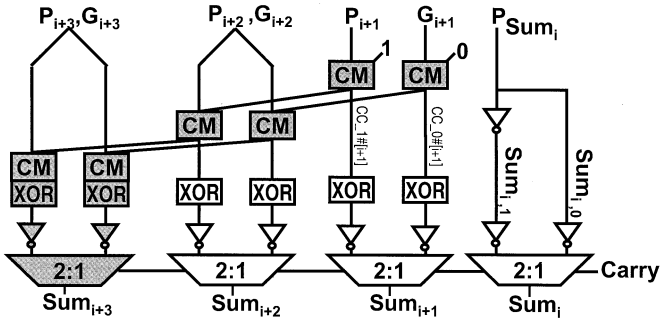


Fig. 7. Non-critical 4-bit conditional sum-generator.

B. Noncritical Conditional Sum Generator

The noncritical section of the adder consists of a 4-bit conditional sum generator that generates two sets of sums, assuming an input carry of 0 and 1, respectively (Fig. 7). The noncriticality of the sum generator permits the use of the ripple carry–merge scheme to generate the conditional carries. Thus, as shown in Fig. 7, the carryin at the first level of each conditional carry rail is tied off to 0 and 1, respectively, generating two rails of conditional carries. An XOR of the partial sum with the conditional carries generates the conditional sums.

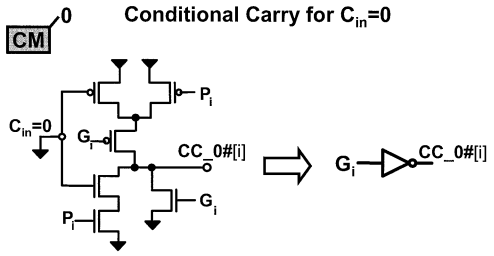
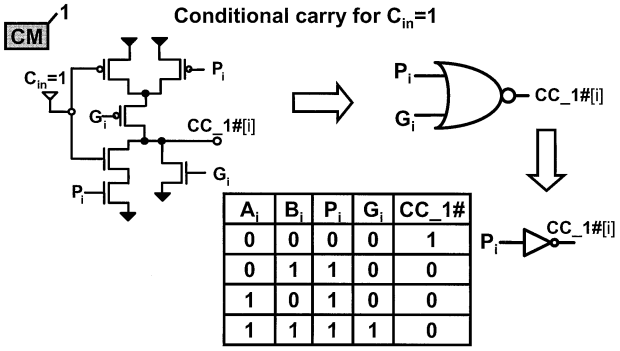


Fig. 8. Optimizing conditional-carry0 rail.



A_i	B_i	P_i	G_i	CC_1#
0	0	0	0	1
0	1	1	0	0
1	0	1	0	0
1	1	1	1	0

Fig. 9. Optimizing conditional-carry1 rail.

The critical and noncritical sections converge at the 2 : 1 multiplexer where the 1-in-4 carries generated by the sparse tree choose the appropriate conditional sum to deliver the final sum. Thus energy-inefficient parallel carry–lookahead logic removed from the main carry–merge tree is implemented in a sidepath using the energy-efficient ripple-carry design, without negatively impacting performance. Such an approach results in smaller area, reduced energy consumption and lower leakage. In the next section, we will further optimize the conditional sum generator to increase its noncriticality and convert the slack thereby obtained into a savings in energy.

C. Optimized 4-bit Conditional Sum Generator

The conditional carry rail corresponding to $C_{\rm in} = 0$ is generated by tying off the input carry to the first-level carry–merge gate ($CC = G_i + P_i C_{\rm in}$) to 0. Thus, the conditional carry $CC\_0\#[i]$ for bit i is logically equivalent to $G_i\#$ for that bit, reducing this gate to an inverter (Fig. 8). Similarly, the first-level carry–merge gate for the conditional-carry rail corresponding to $C_{\rm in} = 1$ reduces to a NOR function of the propagate P_i and generate G_i signals for that bit (Fig. 9). Since P_i and G_i are generated from the same inputs A_i and B_i, there exists a correlation between these signals that can be exploited to further reduce this NOR function to an inverter ($CC\_1\#[i] = P\#_i$). This optimization eliminates the first stage of carry–merge logic in the conditional-sum generators and pushes the inverters required to generate CC_0#[0] and CC_1#[0] to the next level, reducing the number of logic stages in the conditional-sum generators from five stages (Fig. 7) to four stages (Fig. 10). In the next section, we will show how the additional slack obtained as a result of this optimization may be used to reduce power consumption.

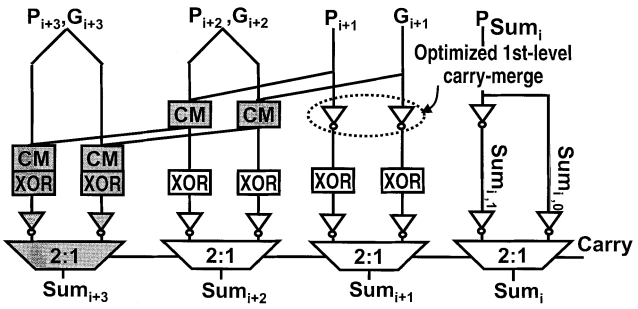


Fig. 10. Optimized 4-bit conditional-sum generator.

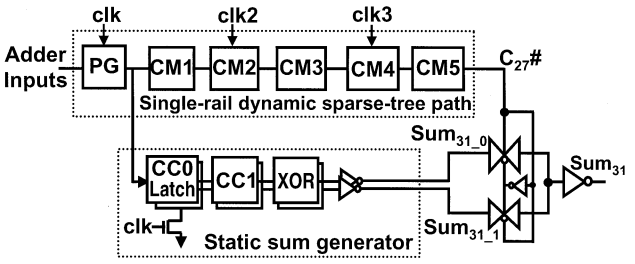


Fig. 11. Critical and noncritical paths in sparse-tree adder.

V. Semidynamic Design

The performance criticality of the AGU demands a dynamic adder implementation. Partitioning the carry–merge tree into critical and noncritical sections enables an energy-efficient implementation by leveraging dynamic, static, and dual-$V_{\rm T}$ techniques. Fig. 11 shows the critical and noncritical paths in the adder core. The critical path, implemented in single-rail dynamic logic begins with the PG block generating the P_i and G_i signals from the inputs $A_i\#$ and $B_i\#$. This is followed by the sparse tree implemented in five stages (CM1–CM5) with the final 1-in-4 carry ($C_{27}\#$) selecting between the two conditional sums (Sum31_0 and Sum31_1) using a 2 : 1 transmission-gate multiplexer. Thus, we have six stages (PG, CM1–CM5) in the critical path in contrast to the five stages (PG, CC0, CC1, XOR, and inverter) in the noncritical path.

To meet the performance requirement of 152 ps, the critical path is implemented in single-rail dynamic logic. Compared to other dual-rail domino implementations [10], this provides $\sim$ 50% reduction in transistor count and interconnect complexity resulting in reduced leakage power consumption and higher performance. The critical sparse-tree path includes the PG block implemented in dynamic logic and controlled by the main clock *clk*, followed by the carry–merge logic gates. These include the three static gates (CM1, CM3, CM5) interspersed with the dynamic carry–merge gates CM2 and CM4, which are controlled by staggered clocks *clk2* and *clk3*, respectively. We exploit the noncriticality of the sum generator to reduce average power consumption by implementing it completely in static CMOS logic. The objective is to leverage the low switching activity of static gates to reduce overall activity in the AGU, thereby lowering the average power consumption. This helps to alleviate the hotspots shown in Fig. 1.

Note that the inputs (G_i, P_i, and G_{i-1}) to the static sum generator are dynamic signals that go high during precharge.

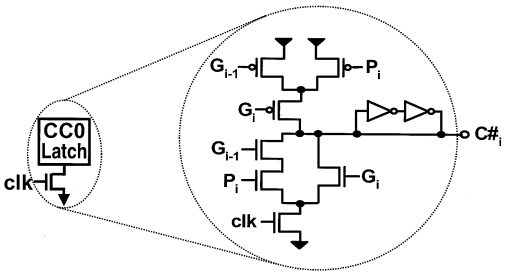


Fig. 12. Stage 1 of sum generator: Set-domino latch.

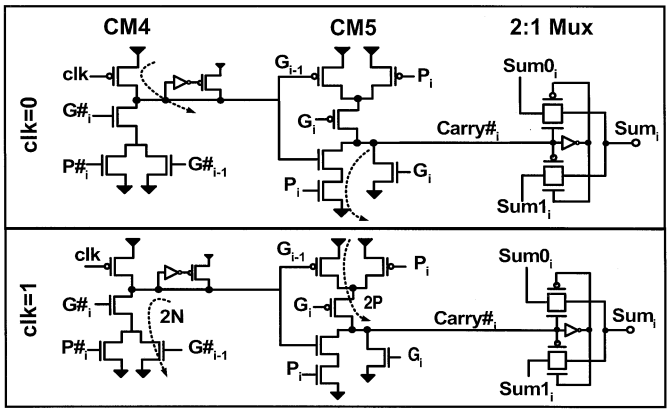


Fig. 13. Interfacing static and domino signals.

To prevent this precharge activity from propagating through the sum generators when the clock goes low, the first gate in the sum generator is converted to a set-domino latch (Fig. 12) by the addition of the clocked footer nMOS device to the static carry–merge gate. This transistor cuts off the discharge path for the output during the precharge phase. A full keeper is added to the output node to hold state during this phase. Thus, we use a static carry–merge latch to hold state during the precharge phase, thereby reducing switching activity in the static blocks, resulting in a semidynamic design that reduces average power consumption.

A. Domino-Static Interface

Potential false evaluations and race conditions that may occur at a static-domino interface are avoided by positioning this boundary at a transmission-gate multiplexer. Fig. 13 shows the state of the final carry–merge gates (CM4 and CM5) and the static-domino interface during both phases of the clock. Note that the 1-in-4 carry ($\mathrm{Carry}\#_i$) is a predischarged dynamic signal and the two conditional sums ($\mathrm{Sum}0_i$ and $\mathrm{Sum}1_i$) are static signals from the conditional sum generator. These signals meet at the transmission gate multiplexer.

During the precharge phase ($clk = 0$), the output of CM4 will be precharged high, which turns ON the pulldown stack of CM5. Therefore, during the precharge phase, $\mathrm{Carry}\#_i = 0$. This turns ON the lower transmission gate and sets the output sum (Sum_i) to be equal to $\mathrm{Sum}1_i$. During evaluation ($clk = 1$), $\mathrm{Carry}\#_i$ either remains low or goes high in response to switching activity in the carry–merge gates CM4 and CM5. Accordingly, the multiplexer will select the appropriate conditional sum ($\mathrm{Sum}0_i$ or $\mathrm{Sum}1_i$) to deliver the final sum. Thus, the use of a static transmission gate multiplexer avoids the possibility of any false evaluations or race conditions from occurring at the static-domino interface.

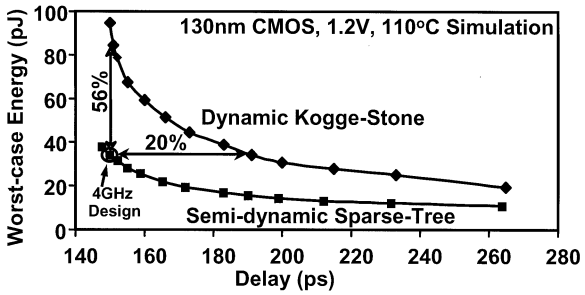


Fig. 14. Energy-delay curve: Kogge-Stone vs. sparse-tree adders.

VI. Energy-Delay Comparisons and Discussion

The energy-delay space (Fig. 14) comparing this design with the KS adder shows the benefits of the sparse-tree adder architecture in a 1.2-V 130-nm technology. The 33%–50% reduction in generate/propagate fanouts and the 30% reduction in maximum interconnect span results in 20% speedup in adder performance. The 80% reduction in wiring complexity has an indirect effect on performance by facilitating a dense layout (Fig. 15) and enabling the widening and shielding of the few performance-critical interconnects in the adder core. Comparing the power consumption of both adders at the design target of 152 ps, we see a 56% reduction in worst-case energy. This was obtained by the elimination of 73% of the carry–merge gates from the main tree and implementing this logic in a sidepath where the energy-efficient ripple carry–merge scheme is used, resulting in 60% smaller transistor sizes. This confirms the energy-efficiency of the sparse-tree design, while meeting the performance target of 152 ps.

The impact of the semidynamic design is seen when we consider the effect of switching activity on average energy (Fig. 16). For a single-rail dynamic design, we assume a constant activity factor of 0.5, which corresponds to the probability that the input data will be high (in which case, the dynamic gate will precharge and evaluate every cycle). Thus, the dynamic KS adder will have an average energy consumption of 41 pJ with a flat energy profile versus input data switching activity. Note that dual-rail domino designs [10] have a switching activity of 100%, since either of the two complementary paths is bound to switch every cycle. Hence, the average energy consumption of a dual-rail domino design is equal to its worst-case switching energy. In contrast, the switching activity of a semidynamic design scales with data activity. This is due to the presence of static sum generator blocks, which do not have any precharge activity and switch only when the input data changes. (Note that the activity factor of a static gate depends on the probability that the input data changes. It does not depend on the probability of the data being at a particular logic value). Thus, for an adder input data activity of 10% (typical in datapath circuits), the single-rail dynamic sparse-tree section of the adder core will have a constant activity of 0.5, whereas the static sum generators will have an

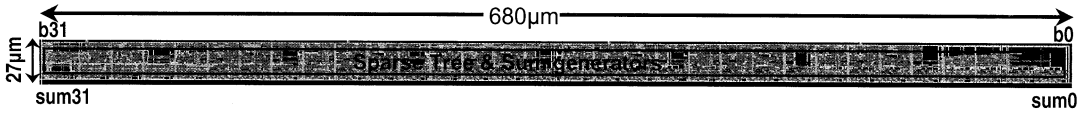


Fig. 15. Adder core layout.

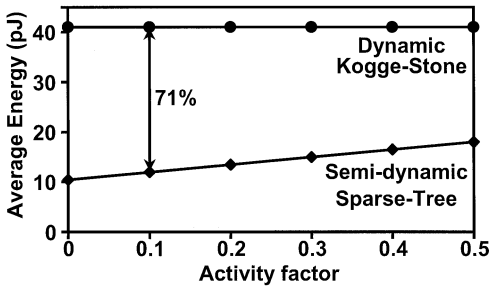


Fig. 16. Scaling of average energy with activity factor.

TABLE I
130-nm CMOS 1.2-V 110 °C ADDER SIMULATION RESULTS

	Low-V_t	Dual-V_t
Delay	152ps	152ps
Switching Energy	36pJ	34pJ (-6%)
Leakage Energy	0.9pJ	0.4pJ (-56%)

activity factor of 0.1, leading to 71% reduction in average energy consumption compared with a dynamic KS implementation.

A. Dual-V_{T} Design

The delay spread between the critical and noncritical sections of the adder can be further exploited to achieve savings in leakage energy. This is achieved by using low V_{T} devices in the critical sparse-tree and high V_{T} devices on the noncritical sum-generator paths. The 10 $\times$ differential in leakage currents between high V_{T} and low V_{T} devices [4] results in an overall 56% reduction in leakage energy consumption without impacting performance (Table I). Note that high-V_{T} allocation was performed on an initial all-low-V_{T} design without transistor resizing. Conversion of the sum-generator devices to high-V_{T} utilizes the remaining slack between the sparse-tree path and the sidepaths. At this point, the main path and sidepath delays are balanced.

B. Scaling Performance

The sparse-tree adder design results in a low average transistor size of 3.5 μm. This property, combined with the dual-V_{T} design described above, results in a low active leakage energy component of $\sim 1\%$ and minimizes the impact of higher leakage in future technologies. Furthermore, the decrease in interstage interconnect and the reduced wiring complexity reduces the effect of increased wire delay in future technologies [11]. In a 100-nm technology, where device leakage is expected to increase by 3–5 $\times$ [12], we project 33% delay improvement and 50% energy reduction (Table II), with a low (4%) leakage energy component, thereby demonstrating the scalability of the semidynamic sparse-tree adder design to future technologies.

TABLE II
SCALING TO 100-nm TECHNOLOGY: SIMULATION RESULTS

	130nm	100nm
Delay	152ps	102ps (-33%)
Switching Energy	36pJ	18pJ (-50%)
Leakage Energy	0.9pJ	0.7pJ (-23%)

VII. SUMMARY AND CONCLUSIONS

The design of an energy-efficient AGU operating at 4 GHz in a 1.2-V 130-nm CMOS technology has been described. We have shown that moving to a sparse-tree adder core design offers 20% delay reduction and 56% energy reduction compared to a KS adder. This relaxes the thermal density issues within the execution core, increasing reliability and reducing cooling costs. The semidynamic design of the adder core provides the performance of a dynamic CMOS design with an energy profile similar to static CMOS, resulting in 71% savings in average energy. The sparse-tree design was also shown to have a low 1% active leakage component with good scaling trends to sub-130-nm technologies.

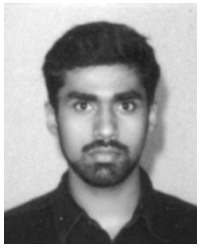
ACKNOWLEDGMENT

The authors would like to thank E. Rosen, R. Saied, S. Jamshidi, G. Gerosa, T. Fletcher, and S. Rusu for discussions, and R. Hofsheier, M. Haycock, and J. Rattner for encouragement and support.

REFERENCES

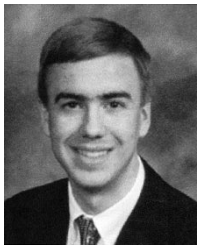
[1] *IA-32 Intel® Architecture Software Developer's Manual*, vol. 1, Intel Corp., Hillsboro, OR, pp. 1–22.
[2] D. Harris and S. Naffziger, "Statistical clock skew modeling with data delay variations," *IEEE Trans. VLSI Syst.*, vol. 9, pp. 888–898, Dec 2001.
[3] D. S. Sager *et al.*, "A 0.18 μm CMOS IA32 microprocessor with a 4 GHz integer execution unit," in *IEEE Int. Solid-State Circuits Conf. Dig. Tech. Papers*, 2001, pp. 324–325.
[4] S. T. Tyagi *et al.*, "A 130 nm generation logic technology featuring 70 nm transistors, dual V_t transistors and 6 layers of Cu interconnects," in *IEDM Tech. Dig.*, Dec. 2000, pp. 567–570.
[5] P. Kogge and H. S. Stone, "A parallel algorithm for the efficient solution of a general class of recurrence equations," *IEEE Trans. Comput.*, vol. C-22, pp. 786–793, Aug 1973.
[6] J. Park, H. Ngo, J. Silberman, and S. Dhong, "470-ps 64-bit parallel binary adder," in *Dig. Tech. Papers, Int. Symp. VLSI Circuits, Systems, and Applications*, 2000, pp. 192–193.
[7] S. Mathew, M. Anders, R. Krishnamurthy, and S. Borkar, "A 4 GHz 130 nm address generation unit with 32-bit sparse-tree adder core," *Dig. Tech. Papers, Int. Symp. VLSI Circuits, Systems, and Applications*, pp. 126–127, June 2002.
[8] S. Mathew, R. Krishnamurthy, M. Anders, R. Rios, K. Mistry, and K. Soumyanath, "Sub-500 ps 64-b ALUs in 0.18 mm SOI/bulk CMOS: Design and scaling trends," *IEEE J. Solid-State Circuits*, vol. 36, pp. 1636–1646, Nov. 01.
[9] S. Knowles, "A family of adders," in *Proc. 14th IEEE Symp. Computer Arithmetic*, Apr. 1999, pp. 277–281.

[10] S. Naffziger, "A sub-nanosecond 0.5 μm 64 b adder design," *IEEE Int. Solid-State Circuits Conf. Dig. Tech. Papers*, pp. 362–363, 1996.
[11] J. Davis, "Interconnect limits on gigascale integration (GSI) in the 21st century," *Proc. IEEE*, vol. 89, pp. 305–324, Mar. 2001.
[12] T. Ghani *et al.*, "Scaling challenges and device design requirements for high performance sub-50–nm gate length planar CMOS transistors," in *Dig. Tech. Papers, Int. Symp. VLSI Circuits, Systems, and Applications*, June 2000, pp. 174–175.



Sanu Mathew (M'99) received the B. Tech. degree in electronics and communications engineering from the College of Engineering, Trivandrum, India, in 1993 and the M.S. and Ph.D. degrees in electrical and computer engineering from the State University of New York at Buffalo in 1996 and 1999, respectively. His Ph.D. research focused on asynchronous circuit design.

He is currently with the High-Performance Circuits Research group at Intel Corporation's Microprocessor Research Laboratories, Hillsboro, OR.



Mark Anders (M'99) received the B.S. and M.S. degrees in electrical engineering from the University of Illinois at Urbana-Champaign in 1998 and 1999, respectively.

Since 1999, he has been with the Microprocessor Research Laboratories, Intel Corporation, Hillsboro, OR, where he is working on high-performance circuits research.

Ram K. Krishnamurthy (S'92–M'98) received the B.E. degree in electrical engineering from the Regional Engineering College, Trichy, India, in 1993 and the Ph.D. degree in electrical and computer engineering from Carnegie Mellon University, Pittsburgh, PA, in 1998. His Ph.D. research focused on low-power DSP circuit design.

Since 1998, he has been with Intel Corporation's Microprocessor Research Laboratories, Hillsboro, OR, where he is currently a Senior Staff Engineer and Manager of the high-performance and low-voltage circuits research group. He is an Adjunct Faculty of the Department of Electrical and Computer Engineering, Oregon State University, Corvallis, where he teaches VLSI system design. He has 21 patents issued, 50 patents pending, and has published over 50 papers in refereed journals and conferences.

Dr. Krishnamurthy serves on the Semiconductor Research Corporation Integrated Circuit and Systems Sciences (ICSS) Task Force and the program committees of the IEEE International Solid-State Circuits Conference, the IEEE Custom Integrated Circuits Conference, the IEEE ASIC Conference, and the IEEE International Symposium on Circuits and Systems. He is the Technical Program Co-Chair for the 2003 IEEE International ASIC/SoC Conference.

Shekhar Borkar (M'97) was born in Mumbai, India. He received the B.S. and M.S. degrees in physics from the University of Bombay, Bombay, India, in 1979 and the M.S. degree in electrical engineering from the University of Notre Dame, Notre Dame, IN, in 1981.

He joined Intel Corporation, Hillsboro, OR, in 1981, where he worked on the design of the 8051 family of microcontrollers, iWarp multicomputer, and high-speed signaling technology for Intel supercomputers. He is currently an Intel Fellow and Director of Circuit Research with the Intel Architecture Group. He is also an Adjunct Member of the Faculty of the Oregon Graduate Institute, Beaverton. He has published 10 articles and holds 11 patents.