GALDS: A Complete Framework for Designing Multiclock ASICs and SoCs

Atanu Chattopadhyay and Zeljko Zilic, *Member, IEEE*

***Abstract*—A Globally Asynchronous, Locally Synchronous (GALS) system with dynamic voltage and frequency scaling can use the slowest frequency possible to accomplish a task with minimal power consumption. With the mechanism for implementing dynamic voltage scaling at each synchronous domain left up to the designer, our Globally Asynchronous, Locally Dynamic System (GALDS) provides a top-down, system-level means to maximize power reduction in an integrated circuit and facilitate system-on-a-chip (SoC) design. Our solution includes three distinct components: a novel bidirectional asynchronous FIFO to communicate between independently clocked synchronous blocks [5], an all-digital dynamic clock generator to quickly and glitchlessly switch between frequencies and a digitally controlled oscillator to generate the global fixed frequency clocks required by the all-digital dynamic clock generator. In addition to being capable of reducing power consumption when combined with dynamic voltage scaling, a GALDS design benefits from numerous other advantages such as simplified clock distribution, high performance operation and faster time-to-market through the modular nature of the architecture.**

***Index Terms*—Application-specific integrated circuits (ASICs), asynchronous logic circuits, circuit topology, clocks, synchronization, tunable oscillators.**

I. Introduction

In the past, high-speed operation and minimum silicon area were the two most important criteria in creating an integrated circuit (IC) [6]–[8]. Today, power consumption of an IC has reached the point where it is becoming increasingly difficult for a package to dissipate enough heat to ensure proper operation [1], [9], [10]. Intel currently ships Pentium 4 processors designed to throttle back execution when power dissipation crosses a pre-defined threshold in an effort to protect the device from over-heating [10], [11]. Clock generation and distribution components are among the costliest in terms of power consumption [12].

Depending on the specific application involved and the design style, clock distributions can consume anywhere from 15% to over 45% of the total system power [6], [13]. Generating and distributing multigigahertz clock signals has become a difficult problem [14], [15] mainly because multiple clock cycles are required by clock signals to cross a chip [16], [17]. This task is even more difficult with the increasing popularity of multiclock architectures [18]–[20] since each clock generally requires a dedicated distribution network. The power requirement of these circuits is given by the dynamic power consumption equation

$$P_{\mathrm{dyn}} = KC_L f V_{dd}^2 \tag{1}$$

where C_L represents the loading capacitance and K is the switching activity factor. For CMOS circuits, the switching activity factor is generally much lower than 1 [21]. However, for clock distribution networks, switching activity occurs twice per clock cycle, resulting in a K equal to 2.

Clock gating [22] is one technique that can reduce power consumption by shutting down the clock distribution network for inactive areas of VLSI circuits. This method works by running blocks at full speed when work is required and having them turned off otherwise. The theoretical power consumption of this technique is roughly the same as using dynamic frequency scaling of the local clock, reducing the idle time of the unit. One key benefit of frequency scaling with respect to clock gating is the reduced electromagnetic interference for the device [23], due to using a lower clock rate.

Using a Globally Asynchronous, Locally Dynamic System (GALDS), we provide a collection of hardware blocks to incorporate dynamically clocked synchronous local blocks into a Globally Asynchronous, Locally Synchronous (GALS) system [1], [2], [24]–[26] system. The architecture, shown in Fig. 1, allows global clocks to be distributed with less concern for clock skew and facilitates communication between the multiple independently clocked domains. Clock skew between local blocks is a desirable trait since it reduces the number of simultaneously switching transistors and thus minimizes the instantaneous current consumption. This skew has the added benefit of reducing electromagnetic interference (EMI) [12]. With GALDS, the distance between local blocks will affect the rate and the latency

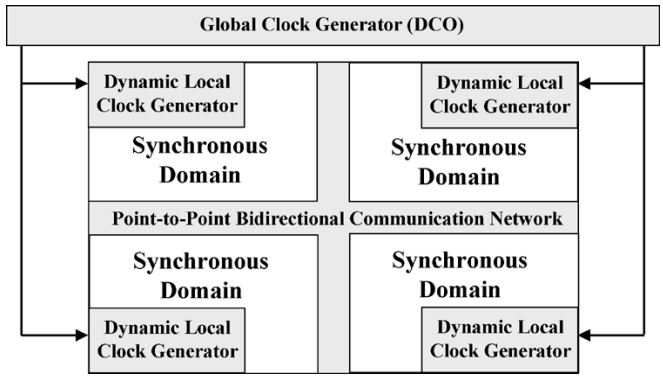

Fig. 1. A globally asynchronous, locally dynamic system.

Manuscript received September 10, 2003; revised June 8, 2004. This work was supported in part through a grant from Altera Corporation.

The authors are with the Department of Electrical and Computer Engineering, McGill University, Montreal, QC H3A 2A7, Canada (e-mail: atanu.chattopadhyay@mail.mcgill.ca; zeljko@macs.ece.mcgill.ca).

Digital Object Identifier 10.1109/TVLSI.2005.848825

1063-8210/$20.00 © 2005 IEEE

of interclock domain communication, but not the correctness of data transfers.

Dynamically clocked and multiclock domain processors and ASICs [11], [14], [19], [27]–[33] are becoming more and more popular due to the increased demands placed on today's integrated circuits. GALDS is designed to be suitable for a number of different clocking architectures, ranging from a single clock integrated circuit, eliminating problems with skew, to a multiple clock domain ASIC, incorporating dynamic clocking into each of its local clock domains. In an ASIC design like [31], dynamic frequency clocking allows each individual functional block to operate at power-of-2 division of a global base clock. Our solution extends this principle by performing the clock division at each functional block (instead of using a single global divider), allowing multiple clock frequencies to be active at any given time in the IC, making pipelining easier in each independent clock domain. Using frequency scaling in each locally synchronous block allows designers to incorporate dynamic voltage scaling with the addition of a frequency and voltage scheduler. Strategies for dynamic voltage scaling have already been the subject of many studies [34]–[37] and these techniques can be extended to the scheduling problem for each dynamically clocked local domain. For dynamic voltage scaling to be effective, its associated dynamic frequency scaling solution must be capable of high-speed frequency changes that are possible with GALDS. Otherwise, changing the frequency/voltage settings could make the system slow to respond, hurting performance and eliminating the potential power savings benefit of this architecture. The power savings achieved by such systems can be quite significant. Single clock, globally distributed solutions with dynamic voltage and frequency scaling are able to achieve an average energy savings of 53.5% in [32] and 32% in [33] with roughly a 50% relaxation in the required clock frequency.

Early work concerning the proposed GALDS architecture was previously reported in [3] and [5], but significant detail has been added here. GALDS contains a number of novel subsystems that are discussed in this manuscript. All the circuits discussed here have been designed for TSMC's 0.18-μm P-well process using the Cadence Virtuoso design environment and simulated with SpectreS using the associated Analog Artist simulation tool. First, the bidirectional synchronizer in Section II uses asynchronous-to-synchronous converters at ends of a novel bidirectional asynchronous first-in–first-out (FIFO) to pass data and control information between domains. The FIFO permits bidirectional data to be multiplexed over a common datapath, allowing simultaneous communication both directions. The FIFO can ease the interconnect burden of an IC by making unidirectional communication paths obsolete, thereby providing a significant resource-sharing advantage by halving the number of data lines required for bidirectional data flow. The asynchronous architecture also mitigates the effect of clock skew between independent dynamically varying clock domains on an IC.

Next, an all-digital clock generator was developed to generate each dynamic local clock, as described in Section III. In contrast to traditional methods of clock generation using PLLs and DLLs that are expensive in area and design effort, as well as require long lock times to execute a frequency change [6], [38], [39], our GALDS solution uses components small enough to be replicated for each locally synchronous block and fast enough to dynamically change frequency with low latency.

Finally, a novel DCO discussed in Section IV incorporates an asymmetric delay control structure to minimize the layout area required for the delay cell. These clock generation circuits are used by the all-digital clock generator, but can also be used autonomously in systems requiring dynamic frequency scaling. Since each block is modular, the overall design is highly scalable and can be used for a wide range of applications, regardless of die size. The ability to incorporate blocks into the overall system regardless of interconnect delays allows ICs to be designed rapidly and robustly by using these pre-existing blocks.

II. Inter-Clock Domain Communication

A. Overview

To provide robust communication between independently clocked blocks, data is transferred through an intermediate asynchronous FIFO stage. Synchronous-to-asynchronous conversions are trivial since asynchronous blocks do not differentiate between the two. The only difference is that an asynchronous signal can be de-asserted as soon as it has been processed, whereas a synchronous signal needs to wait for the next active clock edge. Asynchronous-to-synchronous conversions are performed by restricting the time during which control signals are allowed to propagate to the synchronous output buffers, thereby drastically reducing the possibility of metastability at a clock edge [5].

The problem of communication between clock domains is not new. Several groups have investigated hardware similar to the work discussed here, notably [40]–[43]. Chelcea and Nowick [42], [43] achieve low latency in their FIFO by placing data in static memory cells. Instead of moving data items through the FIFO, tokens are moved in their place. Separate read and write tokens exist within a circular queue with synchronization only required when accessing the FIFO empty or the FIFO full status registers. While this approach is useful between adjacent clock domains, the technique will suffer a performance penalty when synchronization is required over long distances. Then, either the destination's and/or the source's synchronous control signals need to be brought into proximity of the data depending on the physical location of the FIFO with respect to the source and destination domains. In addition, moving one data item from the source domain to the memory block to the destination domain may require long delays, since this transportation will likely occur in more than one step. Our FIFO structure takes intro account the transport nature of the application to break down the transport delay, dividing it amongst the different FIFO stages, allowing for higher throughput. While Chakraborty and Greenstreet's source-synchronous interface design [40], [41] also requires that information be gathered from both the source and destination domains' clocks, they do account for the finite clock skew resulting from such a scenario. Their technique restricts the required number of synchronizations to a very small number of "high-risk" transfers, as opposed to our design that requires synchronization for every data item. However, their

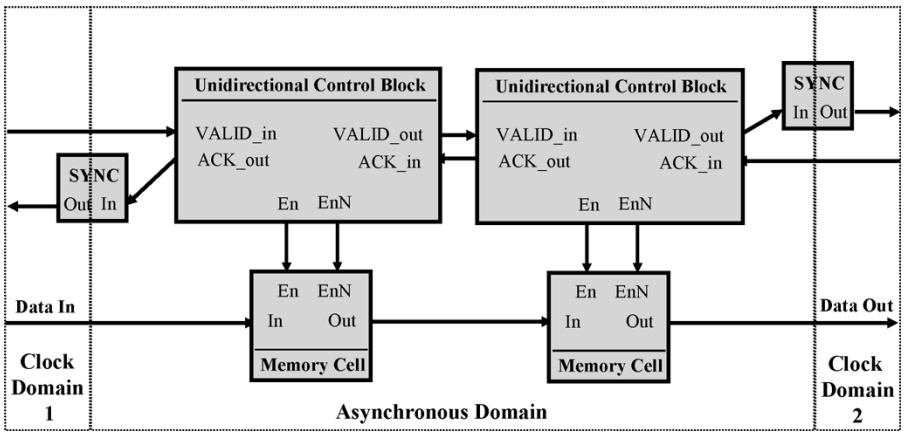

Fig. 2. Two-cell unidirectional synchronizer for transferring data between independent clock domains. The control lines Ack (acknowledge) and Req (valid or request) occur in between each FIFO cell and at both ends of the FIFO. Only the control signals leaving the asynchronous domain and entering a synchronous one need to be synchronized through a SYNC element (see Fig. 7).

design also requires identical frequencies (or stable frequencies with predictable drift) for the clock domains to function correctly. Since our interdomain solution is used with dynamically varying frequencies, including frequency changes that occur midway through a burst data transfer, the assumptions in [40] and [41] are too restrictive to be compatible with GALDS.

B. New FIFO Structure

Using the clock domain synchronizer (see Fig. 7) and the asynchronous control cell (see Fig. 5), it is possible to construct the unidirectional interclock domain FIFO shown in Fig. 2. For this FIFO, the optimal conditions (in terms of maximum throughput) occur when the FIFO is exactly half full with data present in every alternate cell. In this way, every alternate control cell has synchronized control signals, allowing every element in the FIFO to be in movement during every asynchronous handshake. Once data is present in adjacent cells, it must be removed from the target cell before new data can enter from the source cell and thus there is a decrease in throughput. Thus in the optimal case, the design works in two distinct steps. The first step has all the odd cells receiving data and all the even cells sending data. In the second step, this pattern is reversed. Data is presented to the "Data In" port of the unidirectional synchronizer along with a request at "Req In." The data gets read into the memory cell using the same acknowledge signal leaving the unidirectional control block to feed the memory cell's "En" (along with an inverted version of "Ack Out" for "EnN"). Due to the clock domain synchronizer (SYNC) in the "Ack Out" path, the data is guaranteed to be read into the asynchronous domain before the synchronized "Ack Out" is available to clock domain 1. Once the data has been read into the first memory cell, it can pass between any number of other memory cells (and associated unidirectional control blocks) as deemed necessary. If no buffering is required, the total number of unidirectional control blocks (and associated memory cells) can be as little as 1. If there is a significant distance to travel, the asynchronous FIFO stages can act as repeaters [44] or as an interlocking asynchronous pipeline [28] throughout the propagation path to improve throughout. Fig. 2 is shown with two FIFO stages for convenience.

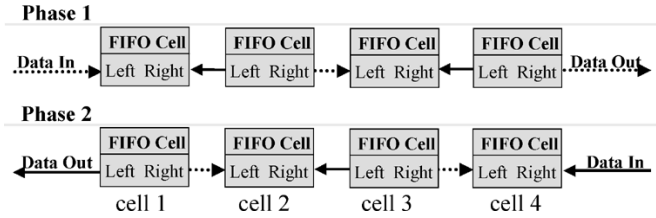

Fig. 3. Operating phases of a bidirectional FIFO.

Using the FIFO at its maximum operating throughput leads to a situation where every alternate data line (bus) between FIFO cells is unused at any given time. To make better use of this leftover bandwidth, it is possible to interleave data flowing in the opposite direction. This will reduce the required wiring costs while increasing throughput for a single datapath by using our bidirectional asynchronous FIFO. With this architecture, the two-phase pattern gets modified slightly, so that every cell either performs two simultaneous reads or two simultaneous writes to its adjacent neighbors. Fig. 3 shows a generic example of how a bidirectional FIFO operates under maximum throughput conditions. Each FIFO cell contains a bidirectional control block and a memory cell. The memory cell can either be a "terminal" memory cell for an end-node or an "intermediate" memory cell otherwise. Both memory cells contain dedicated memory elements for each direction to perform two simultaneous reads or two simultaneous writes. The only difference is that the intermediate memory cell contains a single access port on either side, whereas a terminal memory cell contains dedicated read and write ports at the end-point side of the memory cell. The lines between the FIFO cells in Fig. 3 represent the bidirectional data lines whose values are held using the state conductors used in Fig. 4. The dotted arrows represent data traveling in the left-to-right direction, while the solid arrows show data traveling in the right-to-left direction. During Phase 1 of the transfer, cells 1 and 3 are performing two reads simultaneously and cells 2 and 4 are performing two writes. During Phase 2, the direction of data is reversed for each cell. While it is possible to create a bidirectional FIFO with a single controller, this would require that data be present in both directions at every handshake for any data to flow. This

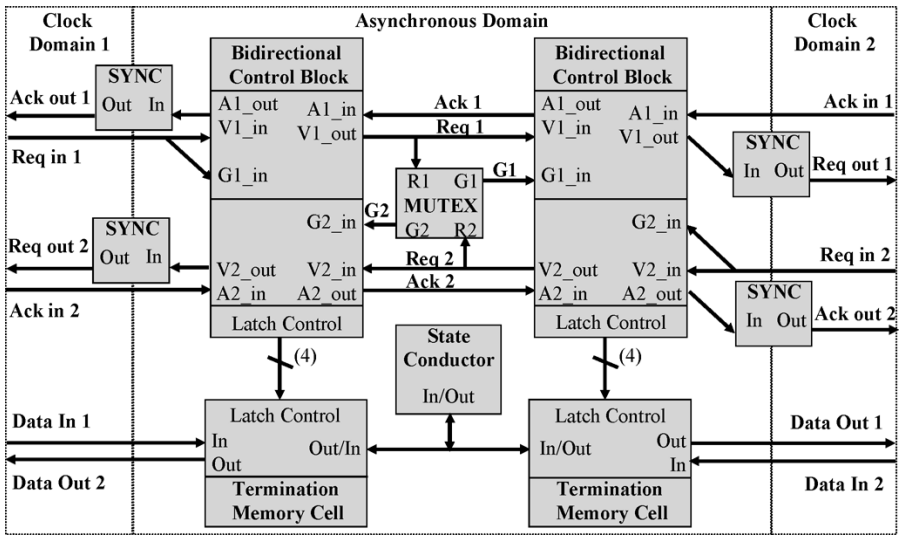

Fig. 4. Two-cell bidirectional synchronizer for transferring data between independent clock domains. The control lines Ack (acknowledge), Req (valid or request) and G (grant) occur twice between every FIFO cell (numbered with 1's for the left to right direction, and numbered with 2's for the right to left direction).

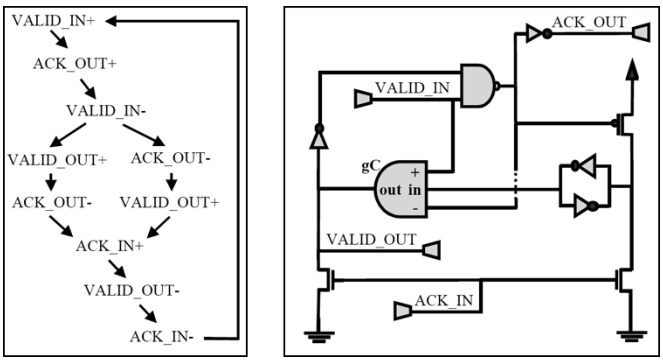

Fig. 5. Four-phase asynchronous control cell and associated state transition graph (STG).

constraint is incompatible with common systems where data is bursty and not entirely predictable. Instead, two independent controllers are used for each FIFO cell, with each one controlling data flowing in a given direction. This technique allows data to stall in one direction while continuing to flow unaffected in the other.

The bidirectional control block in the two-cell bidirectional FIFO in Fig. 4 uses a pair of interlinked four-phase asynchronous control cells (Fig. 5) to control the two memory bits (or registers if used in a bundled data system) required for that particular FIFO cell. The interlinking is performed by the mutual exclusion element in Fig. 8 to prevent adjacent nodes from simultaneously writing to the common data line. The synchronizer (SYNC) blocks in Fig. 7 are used to perform an asynchronous-synchronous domain conversion for the control lines. The termination memory cells used are modified versions of the 12-transistor data latch (see Fig. 9), with the data line split into separate input and output ports on the terminal side of the memory cell. A two-inverter feedback loop is used to construct the state conductors. The asynchronous intermediate step between clock domains renders any clock skew between the blocks harmless and is an inherently low-power approach due to the nature of asynchronous circuits [45]. This solution is thus suitable for dynamic clock management in a multiclock system. Similar to the unidirectional FIFO, any number of bidirectional FIFO cells (consisting of a bidirectional control block and associated intermediate memory cell) can be added to the structure as needed. Again, each buffer cell also acts as a repeater, lowering latency by distributing the line impedance [44]. These buffers can also increase throughput by having multiple data items in transit simultaneously.

C. Four-Phase Asynchronous Handshake Cell

The four-phase asynchronous handshake cell developed for this application is shown in Fig. 5. We selected a four-phase scheme for the controller to ensure that the control lines are encoded as return-to-zero (RTZ). Many high-speed designs [46], [47] use two-phase nonreturn-to-zero (NRZ) control to minimize the number of transitions required per data item. However, this scheme is not appropriate for a bidirectional FIFO: the end of a transaction must be known precisely to allow data in one direction to be removed in favor of data traveling in the opposite direction. A transition on the VALID (request) input causes the data to be latched into the current cell and generates an output request when the data is ready to be transferred to a subsequent cell. The states of adjacent FIFO cells are used to determine when to allow traffic to freely commute through the bidirectional FIFO. Alternative solutions that incorporate bidirectional data flow such as the one-two-one six-phase FIFO [48] require that data movement occurs in strictly alternating directions. The solution described here imposes no such restriction, making it suitable for on-chip communication where the exact direction and throughput requirements cannot be determined in advance.

To increase the speed of the control cell, critical control signals are allowed to trigger events early, bypassing their normal signal path. For example, the ACK_IN signal (Fig. 5) begins to reset the VALID_OUT signal immediately after it is asserted. Normally, the signal path for this ACK_IN signal would pass through the generalized-C element (gC) in Fig. 5 before affecting VALID_OUT. This results in a "signal fight" where two

transistors try to temporarily drive the same output node with different values. This phenomenon is only temporary since the initiating control signal will ultimately propagate through the slower path, terminating the fight condition by eliminating the signal conflict. "Signal fights" are a commonly used practice in asynchronous circuits like GasP [49] to increase overall [50] speed.

Our control cell uses a "push channel" protocol, by which a control cell initiates a transaction to a subsequent cell to send data by generating a request signal. Data is placed on the bus lines at the same time as the Request signal (for the end nodes) or as the Grant signal (for an intermediate node) and is only guaranteed to be present while the request is asserted. Grant signals are discussed in greater detail in Section II-E. This convention for the propagation of data is known as an "Early" data scheme. The control cell behavior is delay insensitive as each transition can only occur when triggered by a specific transition in a previous or subsequent cell, regardless of delay. Hence, once a request (VALID_IN in Fig. 5) passes into the asynchronous control cell through the NAND gate, no other request will be processed until after the current data item has been shipped. A cell should not be allowed to send and receive data simultaneously since this would corrupt data in the FIFO cell's memory. As long as the synchronous input circuitry cannot respond to the ACK_OUT signal (by resetting the VALID_IN signal that initiated the transfer) faster than roughly two gate delays (T_1), the control cell will operate correctly. The two gate delays (T_1) represents the amount of time required for the output of the NAND gate to store the request in the memory element on the right side of Fig. 5 minus the inverter delay in the ACK_OUT path. The frequency of any synchronous domain used in this architecture should thus be limited to around 1/T_1, which is roughly 10 GHz in this technology. The throughput of the FIFO will be limited by the frequency of the surrounding domains due to the synchronization requirements of the control signals. Since the control signals are held for one complete clock cycle in the source and destination's synchronous domains, the FIFO's throughput is limited by the lower of the source or destination's frequencies, or by the maximum inherent throughput of the asynchronous FIFO: 2.90 giga-items/second (bidirectionally), as reported in Section II-F.

D. Control Synchronizer

A synchronizer is required to translate the asynchronous control signals into synchronous ones for each local clock domain. C-elements such as the one used in the synchronizer [shown in Fig. 7(a)], can come in a number of different varieties [51], [52]. A Muller two-input C element drives the output high when both inputs are high, drives the output low when both inputs are low, and holds the previous value otherwise. A two-input asymmetric-C element has an input removed from either the pull-up or pull-down stack, with a "+" on the gate symbol denoting an input that is only used in the pull-down stack, and a "−" denoting an input that is only used in the pull-up. A generalized-C element has multiple (possibly independent) pull-up and pull-down stacks; however for proper operation, no pull-up stack should be asserted while a pull-down stack is asserted and *vice versa*. A weak feedback inverter is usually used to hold the C-element's value when no value is asserted through the input stack. Fig. 6 shows the two-input C-element circuits in more detail.

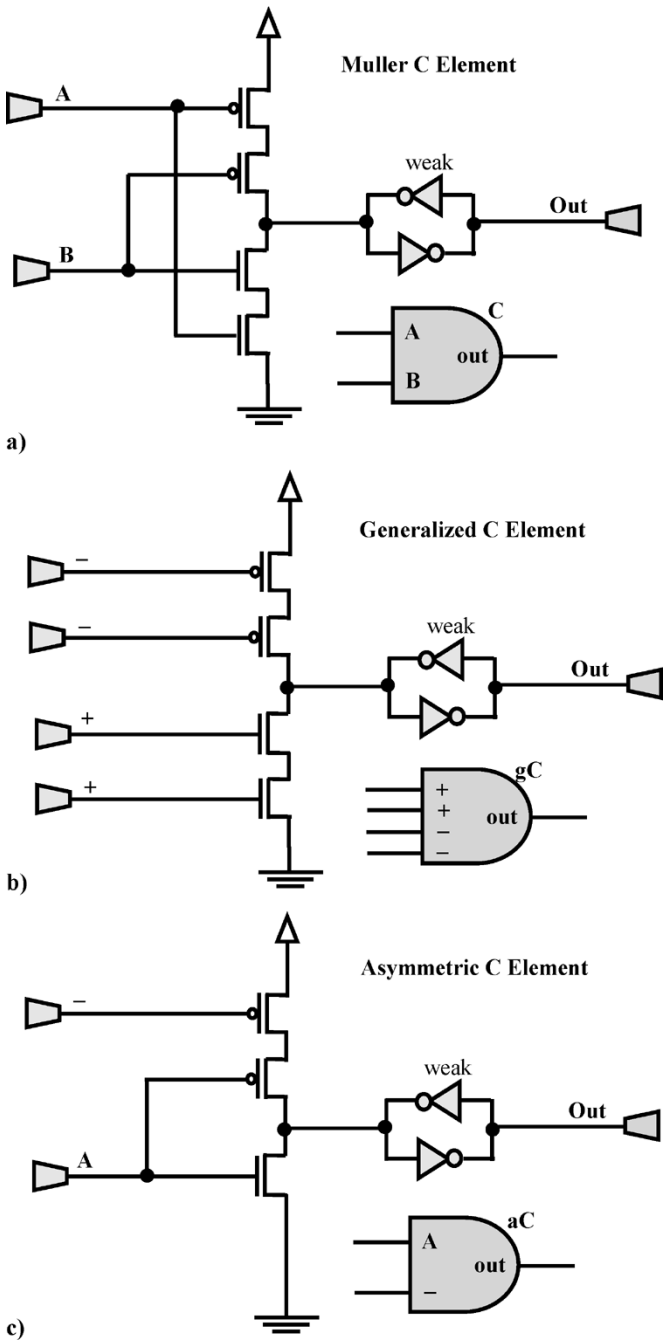

Fig. 6. (a) Muller C element. (b) Generalized C element. (c) Asymmetric C element. In the context of this paper, the C elements used have only one pull-up and one pull-down transistor stack. If an input is being used at a "+" port, it is used in only the pull-down branch to assert a high output. If an input is being used at a "−" port, it is used in only the pull-up branch to assert a low output. Any input not associated with a "+" or a "−" is used by both the pull-up and pull-down branches.

The synchronizer operates in three distinct phases using two skewed clocks. Its structure and operation are shown in Fig. 7(a) and (b), respectively. In the transparent phase, when both clocks are high, the data is allowed to pass through the generalized-C element to the latch. The pull-up branch of the gC will drive a 1 to node X when CLK_LAG, CLK_LEAD and IN are high, and the

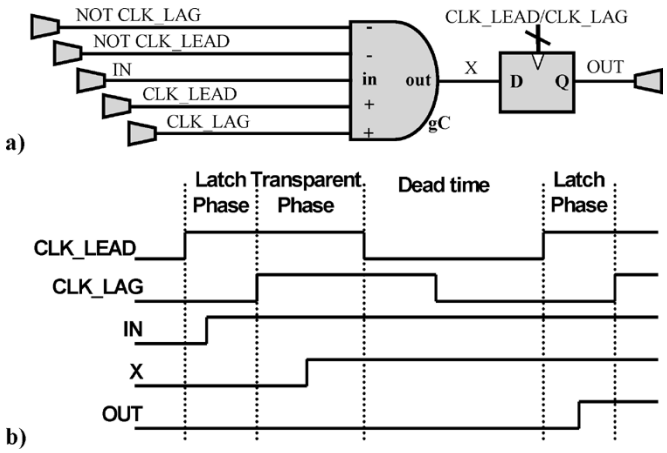

Fig. 7. (a) ASYNC-SYNC converter (synchronizer). (b) Waveform showing how converter latches asynchronous control signals into a local clock domain.

pull-down branch will drive a 0 to node X when CLK_LAG and CLK_LEAD are high and IN is low. While CLK_LEAD is low, a finite "dead-time" exists to allow transients to settle before the register latches the data. Finally, at the rising edge of the leading clock (while the lagging clock is still low), the signal is latched into the local clock domain. For this system to be successful, transitions occurring at the input IN must settle at node X before the beginning of the latch phase. It can only fail when an input change at the very end of the transparent phase does not have enough time to settle before the next rising edge of the leading clock. While this method may increase latency, the probability of a metastability error is reduced since a transition cannot be initiated at X during the dead-time phase of operation. If the input (IN) changes during the dead-time phase, the value at node X will not be affected until after the current dead-time/latch phase combination by the ASYNC-SYNC converter is complete.

There are two high-risk control signal synchronizations that need to be discussed in further detail. The first is the Request (VALID or REQ) that is sent to a synchronous block when it reads from the FIFO. The second is the Acknowledge (Ack) that a synchronous block must wait for when it writes to the FIFO. All other cross-domain synchronizations involve moving a signal from a synchronous domain to an asynchronous one, which is a simpler case since the moment a signal arrives does not affect if an asynchronous circuit will correctly read it. In the read (read-from-FIFO) path, the data is ready in advance of the synchronized request (VALID) signal due to nonzero delay through the synchronizer in Fig. 7(a). Thus if the Request signal satisfies the setup time requirements for the latch, the data will satisfy them too. In the write (write-to-FIFO) path, an asserted signal at IN of the synchronizer (an Ack in this case) will ultimately cause the output OUT to become asserted. If it is not properly detected on the initial clock edge, it will be on a following one. So if a metastability error occurs in either the read or the write cases, it would cause a stall in the data transfer, but once the signal is resolved, the circuitry would return to normal operation without any bit failures. While no simulation could be generated to cause the circuitry to fail, it is physically impossible to fully eliminate metastability.

The overall performance is limited by the frequency of the local clock domains. Each local clock domain holds its control

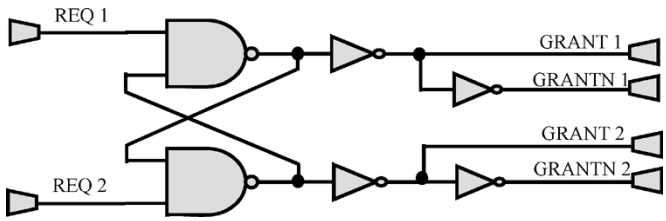

Fig. 8. Mutual exclusion element.

lines (Ack, VALID) for a full clock cycle and the FIFO cannot proceed until the control lines return to their idle states. While employing self-resetting control lines could eliminate this limitation, it would hurt the delay insensitivity of the circuit. One benefit of this design is that there is no possibility of the FIFO overflowing since the asynchronous handshake will not complete until space is available for new data. The implication here is that the synchronous domain attached to the asynchronous interconnect blocks must be able to react appropriately to any of the actions that may occur at the interface. These actions are as follows.

- — Responding to a synchronized Acknowledge by removing data from the data_in port.
- — Responding to a synchronized Request signal by reading data from the data_out port.
- — Making sure that an Acknowledge signal has been received (from the asynchronous domain) before signaling a Request and writing any new data to the asynchronous FIFO's entry (data_in) port. Any new Request must be held until an Acknowledge for it has been received.
- — Holding an Acknowledge signal after having read data from the asynchronous domain until the Request that generated the Acknowledge has been de-asserted.

This interface method compares well to other synchronization methods such as double buffering or pausible clocking [1], [53], [54]. Double buffering can achieve similar throughput with simpler circuitry and reduced metastability tolerance [55]. Pausible clocking involves pausing or stretching clock pulses to ensure that data and control signals have sufficient setup time at a synchronous/asynchronous boundary. This method is significantly more complex than our interface scheme since it is difficult to detect a potential metastability error quickly and to subsequently pause the clock robustly. There may also be synchronization issues resulting from the pausible clock that will have all the characteristics of a jittery clock. Pausible clocking is also used in [1] to create the synchronizers needed for asynchronous wrapping. Asynchronous wrapping involves making the external interface of a synchronous block completely asynchronous by surrounding the block with asynchronous circuitry. By encircling a synchronous block with asynchronous circuitry, our method can be considered as an asynchronous wrapper, albeit using a synchronizing method that alters the moment that data is handled instead of modifying the clock pulses as in [1].

E. Mutual Exclusion Element

A mutual exclusion element (mutex) using cross-coupled NAND gates (Fig. 8) is used to ensure that adjacent cells do not simultaneously write to the same data line. The inclusion

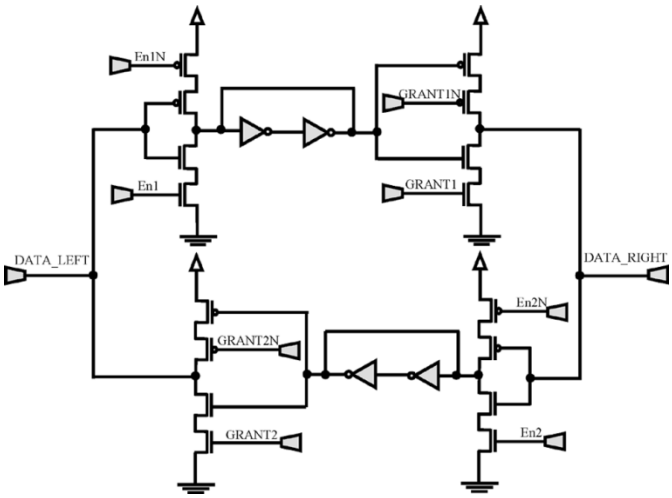

Fig. 9. Twelve-transistor data latch used in the intermediate memory cell. En signals are controlled by the state of the asynchronous control cell (Fig. 4).

of the mutex element creates a three-track handshake design with distinct request (VALID or REQ), acknowledge (Ack) and GRANT signals. The bidirectional data latches shown in Fig. 9 need to include both read and write enables to ensure data integrity on the common data lines DATA_LEFT and DATA_RIGHT. When a GRANT (GRANT1 in Fig. 8) is given for left-to-right propagation, the memory cell write port on the left hand side (GRANT1 in Fig. 9) and the read port of the memory cell on the right hand side (En1 in Fig. 9) must both be enabled. The receiver's read enable signal is controlled by the same grant signal wired to the write enable of the sender's latch, albeit delayed by a noninverting buffer to ensure that the data is stable when latched at the receiving end. When a right-to-left data transaction is required (GRANT2 in Fig. 8), the write port of the right hand memory cell needs to be enabled (GRANT2 in Fig. 9) along with the read port of the memory cell to the left side of the common data line (En2 in Fig. 9). Since the data lines are not always driven by the output of a latch, state conductors are required to hold the bit-values on the data lines. A state conductor holds a value by feeding the output of a noninverting buffer back to its input.

No mutual exclusion can entirely eliminate the possibility of a metastability error; however, good mutual exclusion elements will reduce the probability of a metastable instance from occurring [56], [57]. The design discussed here is similar in behavior to the interlock element proposed in [56] with the exception that NAND gates are used in place of the NOR gates and pass transistors disable the unselected output branch in [56]. One key similarity is that both designs have the possibility of going metastable, with simultaneous input assertions resulting in an infinitely long delay before the output settles at a stable value. However, in practice, only finite delays until the output reaches a fully resolved state can be observed due to variances present in the competing portions of the mutual exclusion element. Although arbitrarily long resolution times will affect performance, the possibility of simultaneous requests is very low and the resolution time will most likely be very short. Since the mutex output is used in an entirely asynchronous domain, the surrounding circuitry will not fail or create errors while waiting for the mutex to resolve any simultaneous requests.

TABLE I
COMPARISON WITH OTHER ASYNCHRONOUS FIFOs

FIFO Type	Throughput (G-items/s)	Notes
Bidirectional FIFO	2.90	Effective throughput
Bidirectional FIFO	1.69	One direction only
Bidirectional FIFO	1.45	Both directions, each direction
Unidirectional FIFO	2.38	Simplified version of bi-directional FIFO
One-Two-One track asynchronous FIFO [48]	0.5	Requires strictly alternating flow 0.8 μm process
MOUSETRAP [46]	3.51	No resource sharing possible due to unidirectional flow
Asynchronous interlocked pipeline [47]	3.3+	No resource sharing possible due to unidirectional flow
Asynchronous FIFO with fights [50]	2.4	Very similar unidirectional throughput
Low-latency FIFO using token rings [42]	0.454	Incompatible with interblock transport

F. Simulation Results

The bidirectional interclock domain communication structure is capable of operating with a maximum throughput of 1.69 giga-items/s when communicating in a single direction and 1.45 giga-items/s in each direction when communicating bidirectionally, yielding an effective throughput of 2.90 giga-items/s. All tests were performed at standard operating conditions and all signals were generated using 50-ps rise and fall times. The resource sharing notes in Table I refer to the ability of our FIFO to multiplex traffic onto a single set of data lines. Our solution compares favorably to other asynchronous FIFO designs such as MOUSETRAP [46] and interlocking pipelines [43], as shown in Table I.

Two unidirectional FIFOs (see Fig. 2) can be used to provide the same functionality as the bidirectional one with higher overall throughput (roughly 4.75 giga-items/s). However, the bidirectional solution is ideal for bundled data systems since the additional area cost of the more complex control cell (versus a unidirectional FIFO) becomes less significant when one considers that two unidirectional FIFOs require two independent datapaths and thus twice the wiring cost of our bidirectional design. In our comparison, only the one-two-one track asynchronous FIFO [48] can make the same claim. Built in an older generation process (0.8 μm), their bidirectional FIFO is slower than our design, as expected. However, our design allows for fully independent data flow in both directions, where [48] requires the direction of data transfers to strictly alternate (right-to-left and then left-to-right). The other unidirectional FIFOs [46], [47], [50] can be replicated to provide bidirectional functionality, but cannot utilize a common, shared datapath. The Chelcea/Nowick designs [42], [43] are not efficient for transporting data from one part of a chip to another and hence would not be a good fit with GALDS. The results obtained represent schematic-level simulations of circuits constructed using TSMC's 0.18-μm P-well process. Fig. 10 shows the Request

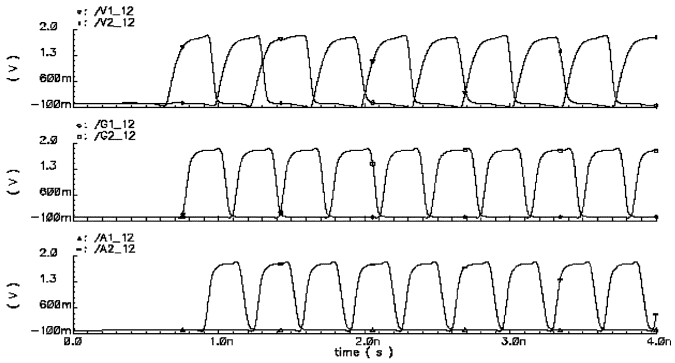

Fig. 10. Simulated operation of the bidirectional FIFO.

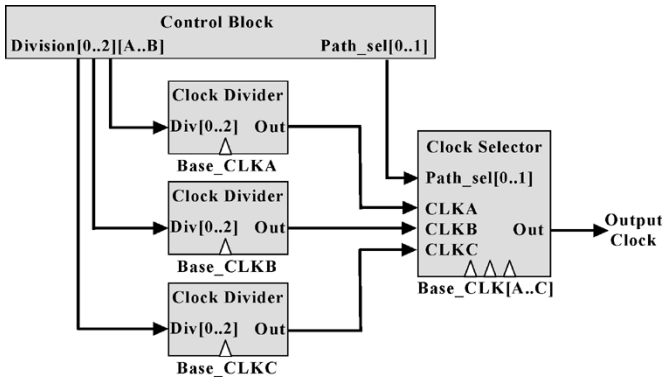

Fig. 11. Block diagram of the all-digital clock generator.

(V1_12 and V2_12), Acknowledge (A1_12 and A2_12) and Grant (G1_12, G2_12) control signals at a FIFO cell during bidirectional communication.

III. All-Digital Clock Generator

A. Overview

For a dynamically clocked system to generate significant power savings without sacrificing performance, it is essential that a clock generation system adapts quickly to the ever-changing needs of the system. The all-digital clock generator used here consists of a set of clock dividers to switch glitchlessly between divisions of a high-speed base clock. A clock selector is used to switch between these dividers. Since this architecture switches between frequencies much quicker than a traditional PLL or DLL, it is much better suited to clock management applications. To demonstrate the effectiveness of such a system, the clock generator shown in Fig. 11 was designed with three independent clock dividers capable of generating integer divisions between 1 and 8. The frequency settings can be changed at any point using either a hardware or a software based frequency selection unit to overwrite the current value in the frequency control register. The Control Block of the clock generator traps this frequency change request and waits until the next rising edge of the current clock cycle (at the old frequency) to initiate the frequency change. The frequency control consists of two parts: the first is a division setting; the second is a path code. There are three different global input frequencies being used by the set of dividers (Base_CLKA, Base_CLKB and Base_CLKC). Each divider is pausible to minimize energy consumption and speed up frequency change requests (since they are paused at the optimal switch points). All three dividers are controlled by the same division code, with only the active divider operating in an unpaused state. When the frequency changes to a different division of the same base clock, it can occur at the clock divider itself without involving the clock selector. If the frequency change involves changing the required base clock, the current clock path (CLKA, CLKB, or CLKC) is first masked out at the Clock Selector and then at the current path's Clock Divider. The masking only occurs at the end of the low time of the current clock to ensure sufficient low time between clock pulses. Meanwhile, once the path is masked out at the Clock Selector, the new path's clock divider is restarted at the required division setting while the Clock Selector unmasks the new clock's output path.

B. Clock Divider

The clock divider contains a series of high-speed double-edged latches with taps between each latch. Each tap can be selected, inverted and used as the feedback signal. Each latch in the signal path generates a half base clock period delay, providing an integer increment in the division factor by 1. In designing a data latch for this clock management application, several important criteria had to be met. The latch needs to be double-edged, accepting data twice per clock cycle to simplify odd-numbered clock divisions. For example, a divide-by-5 requires counting 2.5 clock cycles twice, once for each low and high voltage level, to generate a complete clock period at the output. The next requirement is that the transition times of the latch must be matched to maintain 50% duty cycle clocks. Thus the latch is designed with nearly identical clock-to-output delays regardless of whether the transition is occurring on the rising or the falling edges of the latch clock for both low-to-high and high-to-low output transitions. A third important requirement is that the latch creates consistent and comparable rise and fall times between all possible divisions. The final consideration concerns the transparent time (T_{trans}), or the time that the latch is capable of updating its internal memory. This time must ensure that a latch cannot sequentially read both the current input and the intended input for the next clock cycle during the same transparency time, producing a race condition. Thus, T_{trans} must satisfy the setup and hold time requirements, without exceeding the clock-to-output time of a previous latch plus the setup time of the current latch. Transparency is created using a set of four related clocks. The first clock is a reference clock. The second is a skewed version of the first clock with a phase shift of T_{trans}. The final two clocks are inverted versions of the first two clocks. The result is a transparent period of T_{trans} preceding each clock edge of the reference clock. A simple, specially sized inverter chain can be used to create the four clock signals from a single input clock. While the delay through the inverter chain will generate clock skew with respect to the input clock, this is not harmful in a GALDS system due to the skew tolerance afforded by the asynchronous interconnect fabric. Analysis using the above criteria for T_{trans} has shown that a value of 125 ps is appropriate for this technology because it allows sufficient time for data to be latched without generating race conditions through adjacent latches.

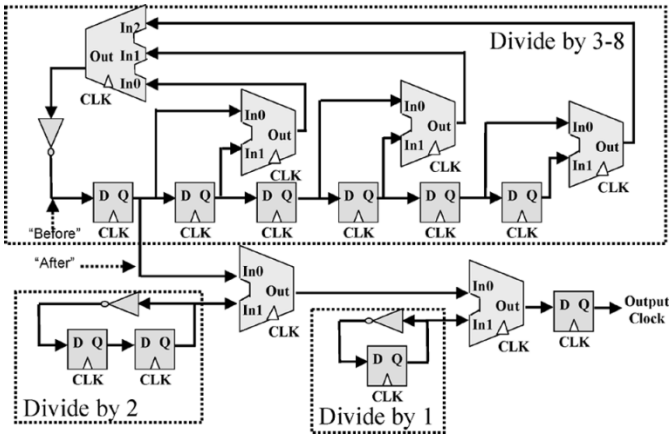

Fig. 12. Structure of the 1-to-8 clock divider.

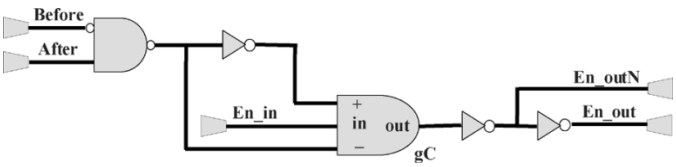

Fig. 13. Sample control block for the div/3 through div/8 enable lines.

The simulated clock divider is capable of operating at base clock rates of up to 2 GHz. At this frequency, two clock cycles are required to multiplex the six taps of the clock divider, creating the need for the separate divide-by-1 and -2 blocks shown in Fig. 12. The unequal time spent within each division is a result of waiting for the output of the divider to reach an appropriate state before switching, thereby maintaining predictable duty cycles for the output clocks. The low-time delay is bounded to

$$\max(T_{\text{initial}}, T_{\text{final}}) + T_{\text{base}} > t_{\text{low}} > \frac{\min(T_{\text{initial}}, T_{\text{final}})}{2} \tag{2}$$

where T represents a clock period and t_{low} represents the low time between clock pulses during a frequency division change. These simulations were performed without regard for the eventual parasitic resistances and capacitances associated with a layout of the circuitry. Typical of other designs manufactured in this technology, we expect a 40%–50% performance penalty if tests were to be performed using fabricated devices [50], [58].

The clock divider should only be allowed to switch between divisions when this path does not instantaneously affect the logic level of the output. The clock divider performs most of the division changes using the 6:1 tap multiplexer in the feedback path of the oscillator. The multiplexer control lines are fed from the control block to each of the Clock Dividers in Fig. 11, but are not explicitly shown in Fig. 12 as the inputs to the three 2:1 and single 3:1 multiplexers used to create the 6:1 multiplexer in the feedback path of the clock divider. Changes in the multiplexer settings are only allowed when the circuitry is on the verge of a 1-0 transition to ensure that an active pulse terminates at a proper end of half-period and not due to a change in the divider setting. These conditions are necessary to maintain the desired clock pulse width. The 1-0 transition is detected by a NAND gate in the sample control

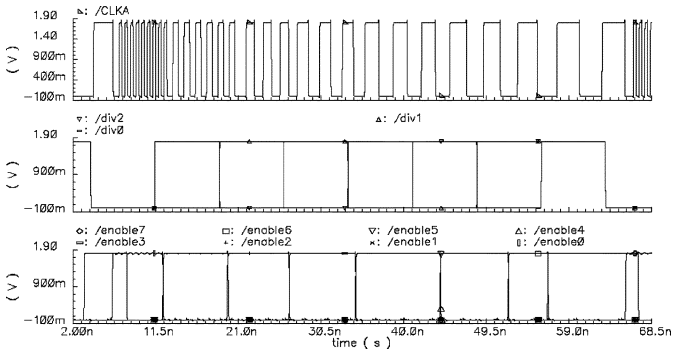

Fig. 14. Simulated operation of the clock divider.

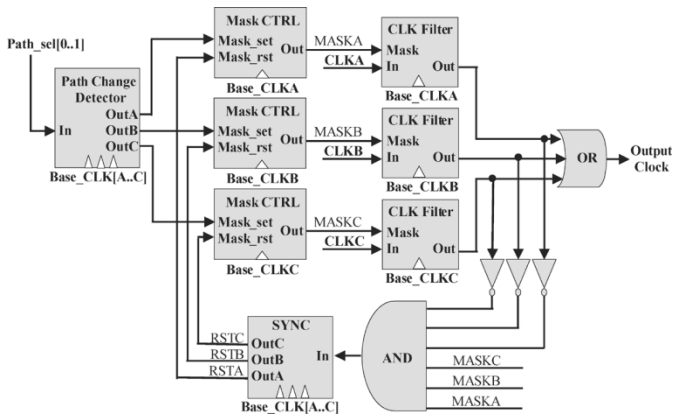

Fig. 15. Simplified structure of three-input clock selector.

block (Fig. 13) monitoring the "Before" and "After" lines in the clock divider in Fig. 12. Once this transition is detected, the En and EnN (Fig. 13) signals tell the multiplexer setting registers of the clock divider to update their values, initiating the division change. The circuitry designed to switch into and out of the divide-by-1 and 2 states also needs to monitor the logic level of the dedicated oscillators for the output to remain glitchless. Fig. 14 shows the clock divider output resulting from a frequency ramp down.

C. Clock Selector

The clock selector shown in Fig. 15 can quickly and glitchlessly select between three independent clocks (provided that the range of frequencies does not exceed half of the highest frequency input clock. Clock gating circuitry exists along each input clock's path, with at least two of these clock gating circuits active at any given time. The clock selector combines the three independent clock paths using a three-input NOR gate designed to provide equal rise and fall times, regardless of which input is selected. The control block monitors the state of the inputs to control the clock gating circuitry (MASK CTRL), providing a glitchless output. The system uses an asynchronous control block to detect a change in the desired clock path using the two-bit path_sel code (path code) that comes from the Control Block in Fig. 11. When a new path code is read from the path code registers, the Clock Selector must switch from one of the three clock dividers to another. To perform this path change robustly without creating glitches or frequency transients, the

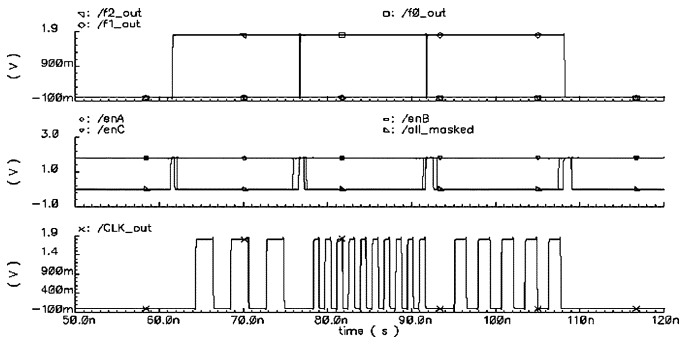

Fig. 16. Simulated operation of the clock selector.

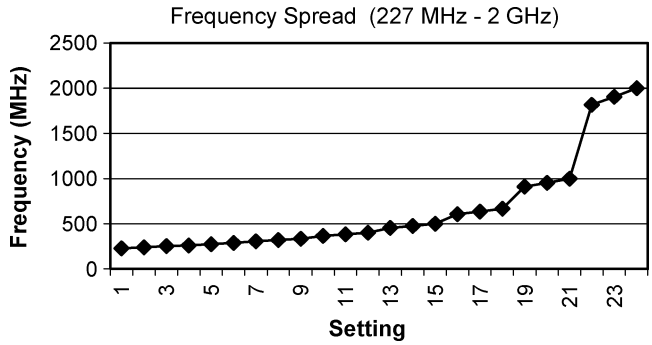

Fig. 17. Possible frequency spread achievable by the all-digital clock generator.

new path code is first read (and synchronized) into each of the three independent base clock domains and held for at least one clock cycle in the new domain. Once the new path code is visible in each of the three BASE_CLK domains, the selector masks the output of the currently active clock path: CLKA, CLKB or CLKC in Fig. 15, which are the divided versions of BASE_CLKA, BASE_CLKB and BASE_CLKC, respectively. Only one CLK path is active at any given moment, so two out of the three CLK Filters are always in a "clock-masked" state whenever any new clock path code is detected. This ensures that the time to reach an "all clocks masked" state is short, thereby speeding up the path change operation. Once all the masking signals have propagated and all the clocks are idle (verified using the six-input AND gate in Fig. 15), the six-AND output is synchronized into each of the three clock domains using the SYNC block in Fig. 15, which uses three independent instances of the synchronizer in Fig. 7. The only difference is that only one reset (restart) output of the SYNC block (RSTA, RSTB, or RSTC) is asserted reflecting the new path code setting. As such, the nonasserted paths remain masked. The Mask CTRL block (Fig. 15) only allows the appropriate masking signal (MASKA, MASKB, or MASKC) to be re-enabled when the appropriate clock input (CLKA, CLKB, or CLKC) is low to prevent incomplete clock pulses (pulses shorter than half of the desired clock period) from appearing at the output. This technique leads to a variable delay between adjacent clock pulses, based on the difference in base clock phase and frequency. The delay is bounded to

$$T_{\text{initial}} + T_{\text{final}} + 3 * \frac{\max(T_{\text{baseA}}, T_{\text{baseB}}, T_{\text{baseC}})}{2} > t_{\text{low}} > \frac{\min(T_{\text{initial}}, T_{\text{final}})}{2} \quad (3)$$

where t_{low} represents the low time between clock pulses and $\mathrm{T}_{\text{initial}}$ and $\mathrm{T}_{\text{final}}$ respectively represent the period of the output clock in the pre-change state and the period of the output clock in the post-change state. BaseA, BaseB, and BaseC represent the base clocks of the clock selector input sources. Not all of t_{low} is undesirable since there needs to be some low time between clock pulses, equivalent to the lower bound of the inequality. Fig. 16 shows the clock selector switching between three arbitrary clock patterns in simulation. As shown in Fig. 17, a wide frequency range can be produced with good resolution using the three input paths and the divide-by-8 capability of the clock dividers.

D. Simulation Results

Simulations show that the all-digital clock generator designed here consumes 167 mW when producing a 1 GHz local clock (using a 1-GHz base clock). A test circuit operating at 1 GHz with three possible voltage source levels consumes 230 mW at 1.2 V, 362 mW at 1.5 V, and 537 mW at 1.8 V. The combinational logic block tested contains 50K transistors operating with a switching activity of 50% in TSMC's 0.18-μm technology.

IV. Digitally Controlled Oscillator

A. Overview

The digitally controlled oscillator (DCO) is the third component in the proposed ASIC/SoC architecture. The oscillator is versatile enough to be used in an all-digital PLL (ADPLL), replacing the voltage-controlled oscillator (VCO) of a conventional phase-locked loop (PLL). Should the frequency spread of the DCO be sufficient, it can be used as the dynamic local clock generator with the inclusion of additional control circuitry. DCOs and ADPLLs are convenient for digital systems because of their short lock-in times and the convenience of having the output frequency controlled by a digital word [59].

There are four general categories of DCOs currently available to designers. The first is a path delay oscillator that uses a chain of logical elements to form a circular ring oscillator. The number of logical elements can be changed to generate different delays through the ring and subsequently, different output frequencies. This kind of device suffers from low precision and is not appropriate for high frequency operation. The second DCO design is the Schmidt-trigger-based current-driven oscillator. This design requires a Schmidt-trigger inverter and a large capacitance whose area overhead makes it difficult to implement on integrated circuits. The third type is the direct digital synthesis (DDS) DCO that is constructed from a phase accumulator, lookup table, and a digital-to-analog (D/A) converter. This method places digital samples of a sinusoid in the lookup table and reads them off periodically to generate an output clock through the D/A. This type of oscillator is also known as a numerically controlled oscillator (NCO) and is useful due to the signal quality and stability of its output [60]. However, it requires a stable reference clock to operate correctly and can only output a signal at half the frequency of the reference clock (due to Nyquist rate considerations) [61]. The fourth DCO design is a current-starved ring oscillator where the output frequency can

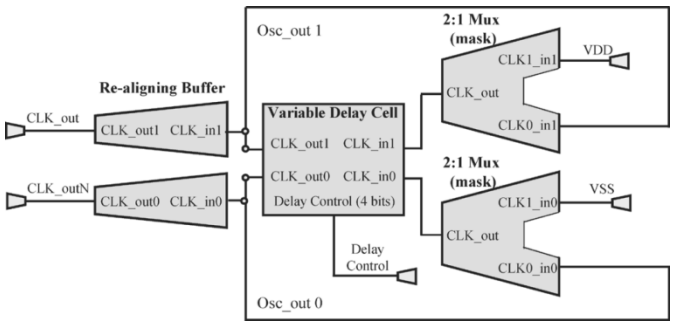

Fig. 18. Architecture of digitally controlled oscillator (DCO).

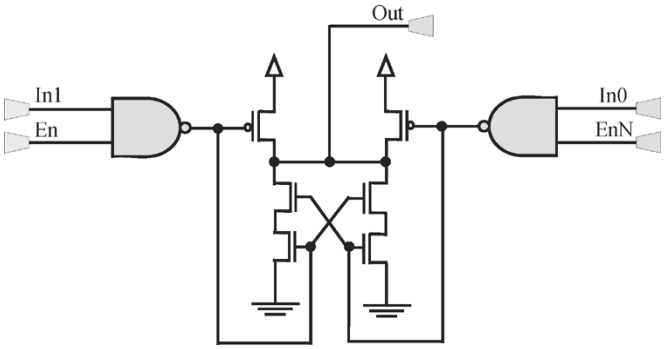

Fig. 19. 2:1 Logical multiplexer used in DCO.

by modified by controlling MOS switches that alter the delay through the oscillator. This last design is generally known to have good frequency linearity, which is an important characteristic of any DCO [62], [63].

B. Our DCO Design

The DCO designed here is a current-starved design with one major modification: the control transistors are placed asymmetrically in only the NMOS group, thereby minimizing the area required to implement the device. Traditionally, silicon area is a major drawback of this form of the DCO [64]. The variable delay cell of the DCO shown in Fig. 20 uses only a third to a quarter of the area of a conventional current-starved variable delay cell requiring both PMOS and NMOS delay control transistors. The DCO shown in Fig. 18 has a complementary clock datapath and uses the variable delay cell (VDC) of Fig. 20 and two instances of the 2:1 multiplexer (mux) shown in Fig. 19. The muxes are used to pause the clock by selecting between the current feedback and a fixed voltage reference. The fixed VSS signal (Fig. 18) is tied to In0 of one Fig. 19 multiplexer and the fixed VDD signal (Fig. 18) is tied to In1 of the other Fig. 19 multiplexer. The clock feedbacks are tied to the unused input of each multiplexer (In1 and In0, respectively). The "Out" signal of each 2:1 mux is then used to drive the VDC of the DCO in Fig. 18. The mux is slightly more complex than a traditional mux to slow down the pull-down time of the multiplexer, creating a slower feedback path for a low signal travelling to the input of the VDC with respect to a high feedback signal to ensure that the VDC does not deadlock (which is a situation that could occur if both the Out0 and Out1 signals in Fig. 20 are allowed to simulatenously be low). As such, the complementary clock signals within the feedback loop of the oscillator are

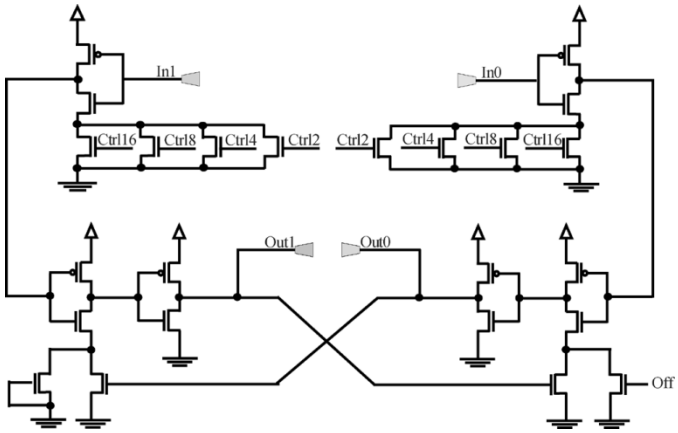

Fig. 20. Digitally controlled variable delay cell (VDC).

designed to always be overlapping. The overlap of these signals is the reason that the re-aligning buffer shown in Fig. 21 can be used to re-shape the resulting asymmetric duty cycles of the complementary clocks. Since the DCO is designed to start from a reset state (with the fixed voltage references chosen and the "Off" transistor in the VDC (Fig. 20) asserted), it can be guaranteed that the deadlock condition will not be present at startup. During normal operation, the shorter delay for the high feedback signal will ensure that the deadlock condition cannot occur. Due to the timing critical nature of this component, it is important that appropriate layout techniques (common centroid transistors, matched wire lengths) are used to minimize any unexpected differences in timing that may exist between the two paths. While process variations may still have an effect on the output, as long as the two output signals remain overlapped, the deadlock condition cannot occur. The shorter the delay through the VDC, the faster a low output (Out0 and Out1 in Fig. 20) travels through the VDC and the closer the DCO is to reaching a deadlock condition. Considering that the feedback paths have been carefully designed to have nontrivially different high and low signal delays and that local process variations are normally quite small, there is low probability of the circuit reaching deadlock. Deadlock could be avoided by ignoring the highest frequency settings of the device, although this is not an ideal solution. If a deadlock state were to occur with the device, it is possible to simply take the NOR of Out0 and Out1 (Fig. 20) to toggle a momentary reset through the oscillator (by switching the feedback muxes in Fig. 19 to their fixed supply settings and asserting the "off" signal in the VDC), restoring the device to a legal operating state.

C. Variable Delay Cell

The key novelty of this DCO is that the control transistors used to vary the delay through the cell are placed asymmetrically in the NMOS group of the VDC as shown in Fig. 20. The four NMOS transistors in the CTRL (control) set are sized as binary weighted powers of 2 (2, 4, 8, and 16) to give the transistor group an effective size range of even numbered multiples between 2 and 30 (inclusive) of the base transistor size (2 μm for this application). By selecting a smaller setting for these CTRL transistors, the delay through the cell is increased, resulting in longer delays for the falling edge of both of the intermediate

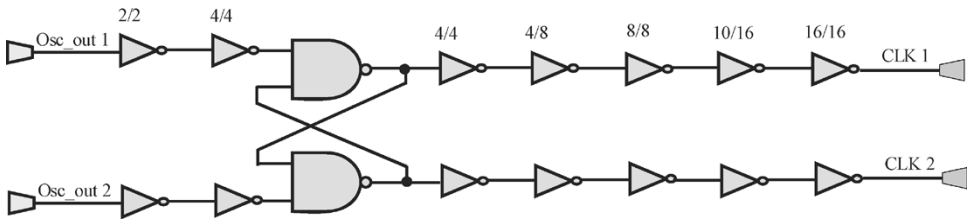

Fig. 21. Re-aligning complementary clock buffer.

complementary clocks at the output of the first inverting elements of Fig. 20. However, the rising edge of this inverting element is still strong due to the lack of controlling transistors in the PMOS group. Similar to the approach used in [57], this technique achieves propagation delays inversely proportional to the equivalent MOS width.

Clock synchronization occurs in the next stage of the variable delay cell. Inverters with NMOS enable transistors ensure that the falling edge of one output branch is dependent on the rising edge of the complementary output branch. While this technique does synchronize the two clock signals, it suffers from two drawbacks. First, it generates nonzero delay between the falling edge of one clock and the rising edge of its complement. Next, the circuit can deadlock with both outputs driven low when the output of one branch of the delay cell falls before the output of its complementary branch has been driven high. This scenario is avoided by ensuring that the complementary input of the variable delay cell has nonoverlapping zeroes through different high and low delay data paths through the 2:1 mux (Fig. 19). To ensure that this overlap is maintained through the VDC, careful transistor sizing and proper layout techniques are necessary in the VDC. The final inverter in the variable delay cell is used to buffer the output for the feedback and output paths of the DCO.

D. Re-Aligning Buffer

The re-aligning clock buffer, shown in Fig. 21, shapes the complementary output signals produced by the VDC/clock mux pair, specifically the Out0 and Out1 of the VDC in Fig. 20. The ratios shown represent the multipliers applied to the base transistor width of 2 μm (PMOS multiplier over NMOS multiplier). The PMOS transistors are multiplied by an additional factor to account for the electron mobility differences between these devices. A traditional buffer cannot be used because the amount of overlap will vary depending on the delay setting selected due to the asymmetrical delay control transistors. The longer the delay setting, the more overlap there will be. The technique used here exploits the overlapping nature of the VDC outputs required for the device not to deadlock. A pair of cross-coupled NAND gates is used in the re-aligning buffer to transform the two outputs of the VDC with varying overlap into a pair of clocks with a fixed amount of nonoverlap. Once the signals with constant nonoverlap are created, a specially sized inverter chain is used to re-align the signals to obtain complementary outputs.

E. Simulation Results

The DCO created here is capable of operating in a frequency range between 2.065 and 2.502 GHz when simulated using TSMC's 0.18-μm process. This represents more than 20% flexibility in the frequencies that can be created using the digital control of the variable delay cell. Should a different frequency range be required, it is possible to configure the device with any even number of additional inverters in the feedback path to create a roughly 120-ps extension of the final output clock period per pair of inverters added. Regardless of the configuration (number of inverters added) or the speed setting (delay code sent to the VDC), fully complementary outputs and fast transition times (around 35 ps) are maintained. The duty cycle of the output does not differ from ideal by more than 1% of the clock period, regardless of the settings used. Fig. 22 shows the output of the DCO through the various stages of the re-aligning buffer, with Out0 and Out0B representing the final output clocks. Table II and Fig. 23 show the frequencies and clock periods achievable by the DCO using 0 (Configuration 1), 2 (Configuration 2), and 4 (Configuration 3) feedback inverters. Controller setting is the total multiplier used in the variable delay cell.

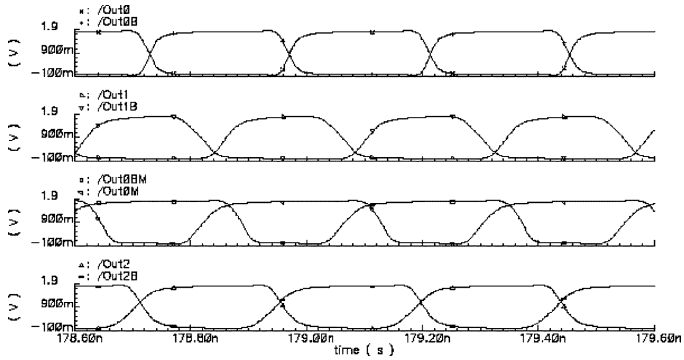

Fig. 22. Simulated DCO operation.

TABLE II
FREQUENCY SPREAD ACHIEVABLE BY THE DCO

Setting	Configuration		Configuration		Configuration	
	T (ps)	F (GHz)	T (ps)	F (GHz)	T (ps)	F (GHz)
2	484.32	2.065	605.61	1.651	725.51	1.378
4	437.72	2.285	557.78	1.793	677.32	1.476
6	423.11	2.363	542.23	1.844	661.51	1.512
8	415.90	2.404	536.13	1.865	655.57	1.525
10	411.01	2.433	531.05	1.883	650.56	1.537
12	408.72	2.447	528.10	1.894	647.29	1.545
14	405.71	2.465	525.94	1.901	645.21	1.550
16	404.98	2.469	525.14	1.904	644.68	1.551
22	401.68	2.490	521.97	1.916	641.10	1.560
30	399.61	2.502	519.95	1.923	639.23	1.564

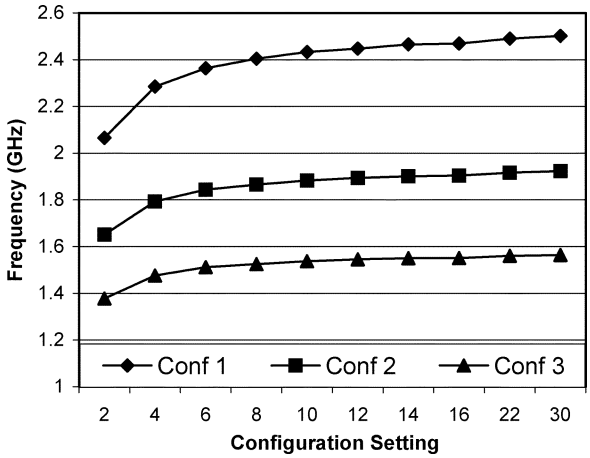

Fig. 23. Graph of DCO's frequency spread.

V. Conclusion

Two primary requirements of GALDS are the ability to control the frequency of the local blocks independently and the ability to robustly communicate between the local blocks running at a wide range of varying frequencies. By combining an interclock domain communication structure, a local clock generator and a global clock generator, we have created the complete framework for GALDS. This system provides high-performance operation while reducing problems with clock skew and excessive heat dissipation. When combined with dynamic voltage scaling, GALDS can also produce very low power circuits while maintaining excellent performance due to the fast frequency changes produced by our all-digital dynamic clock generator. This solution shows that a high-performance GALDS solution can be practically implemented in today's technology without prohibitive design effort and silicon area. While these distinct circuits are here used together as a system, they are versatile enough not to be limited to this architectural scheme. Together, our GALDS solution allows designers to robustly design faster and more power efficient integrated circuits where traditional techniques may fall short.

References

[1] J. Muttersbach, T. Villiger, H. Kaeslin, N. Felber, and W. Fichtner, "Globally-asynchronous locally-synchronous architectures to simplify the design of on-chip systems," in *Proc. 12th Annu. IEEE ASIC/SOC Conf.*, 1999, pp. 317–321.

[2] D. Chapiro, "Globally-asynchronous locally-synchronous systems," Ph.D. dissertation, Stanford Univ., Stanford, CA, Oct. 1996.

[3] A. Chattopadhyay and Z. Zilic, "A globally asynchronous locally dynamic system for ASICs and SoCs," in *Proc. GLSVLSI*, 2003, pp. 176–181.

[4] A. Chattopadhyay, "High-speed structures for dynamically clocked and multi-clock systems," Master's thesis, McGill Univ., Montreal, QC, Canada, Jun. 2003.

[5] A. Chattopadhyay and Z. Zilic, "High speed asynchronous structures for inter-clock domain communication," in *Proc. ICECS*, 2002, pp. 517–520.

[6] I. Brynjolfson, "Dynamic clock management circuits for low power applications," Master's thesis, McGill Univ., Montreal, QC, Canada, Apr. 2001.

[7] A. Das, "On the transistor sizing problem," in *Proc. 13th Int. Conf. VLSI Design*, 2000, pp. 258–261.

[8] R. Singh Bajwa, R. M. Owens, and M. J. Irwin, "Area time trade-offs in micro-grain VLSI array architectures," *IEEE Trans. Comput.*, vol. 43, no. 10, pp. 1121–1128, Oct. 1994.

[9] W. R. Daasch, C. H. Lim, and G. Cai, "Design of VLSI CMOS circuits under thermal constraint," *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 49, no. 8, pp. 589–593, Aug. 2002.

[10] Intel Corp. (2004) Datasheet: Intel Pentium 4 Processor With 512-KB L2 Cache on 0.13 Micron Process and Intel Pentium 4 Processor Extreme Edition Supporting Hyper-Threading Technology. [Online]. Available: http://www.intel.com/design/pentium4/datashts/29864312.pdf

[11] G. Magklis, M. L. Scott, G. Semeraro, D. H. Albonesi, and S. Dropsho, "Profile-based dynamic voltage and frequency scaling for a multiple clock domain microprocessor," in *Proc. 30th Annu. Int. Symp. Computer Architecture*, 2003, pp. 14–25.

[12] R. Y. Chen, N. Vijaykrishnan, and M. J. Irwin, "Clock power issues in system-on-a-chip designs," in *Proc. IEEE Workshop on VLSI*, 1999, pp. 48–53.

[13] W. Qing, M. Pedram, and X. Wu, "Clock-gating and its application to low power design of sequential circuits," *IEEE Trans. Circuits Syst. I, Fundam. Theory Applicat.*, vol. 47, no. 3, pp. 414–420, Mar. 2000.

[14] D. Peiliang, Y. Rilong, X. Hongbo, and Y. Chengfang, "Multi-clock driven system: a novel VLSI architecture," in *Proc. 4th Int. Conf. ASIC*, 2001, pp. 555–558.

[15] A. E. Sjogren and C. J. Myers, "Interfacing synchronous and asynchronous modules within a high-speed pipeline," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 8, no. 5, pp. 573–583, Oct. 2000.

[16] S. Hassoun, C. J. Alpert, and M. Thiagarajan, "Optimal buffered routing path constructions for single and multiple clock domain systems," in *Proc. ICCAD*, 2002, pp. 247–253.

[17] S. Hassoun and C. J. Alpert, "Optimal path routing in single- and multiple-clock domain systems," *IEEE Trans. Computer-Aided Design Integr. Circuits Syst.*, vol. 22, no. 11, pp. 1580–1588, Nov. 2003.

[18] J. Schmid and J. Knablein, "Advanced synchronous scan test methodology for multi clock domain ASICs," in *Proc. 17th IEEE VLSI Test Symp.*, 1999, pp. 106–113.

[19] G. Semeraro, G. Magklis, R. Balasubramonian, D. H. Albonesi, S. Dwarkadas, and M. L. Scott, "Energy-efficient processor design using multiple clock domains with dynamic voltage and frequency scaling," in *Proc. 8th Int. Symp. High-Performance Computer Architecture*, 2002, pp. 29–40.

[20] Y. Semiat and R. Ginosar, "Timing measurements of synchronization circuits," in *Proc. 9th Int. Symp. Asynchronous Circuits and Systems*, 2003, pp. 68–77.

[21] N. Chabini, E. M. Aboulhamid, and Y. Savaria, "Determining schedules for reducing power consumption using multiple supply voltages," in *Proc. ICCD*, 2001, pp. 546–552.

[22] Q. Wang and S. Roy, "Power minimization by clock root gating," in *Proc. ASP-DAC*, 2003, pp. 249–254.

[23] L. Bluno, F. Gregoretti, C. Passerone, D. Peretto, and L. M. Reyneri, "Designing low electro magnetic emissions circuits through clock skew optimization," in *Proc. ICECS*, 2002, pp. 417–420.

[24] A. Hemani, T. Meincke, S. Kumar, A. Postula, T. Olsson, P. Nilsson, J. Oberg, P. Ellervee, and D. Lundqvist, "Lowering power consumption in clock by using globally asynchronous locally synchronous design style," in *Proc. Design Automation Conf.*, 1999, pp. 873–878.

[25] S. Moore, G. Taylor, R. Mullins, and P. Robinson, "Point to point GALS interconnect," in *Proc. 8th Int. Symp. Asynchronous Circuits and Systems*, 2002, pp. 69–75.

[26] J. N. Seizovic, "Pipeline synchronization," in *Proc. Int. Symp. Advanced Research in Asynchronous Circuits and Systems*, 1994, pp. 87–96.

[27] B. Brock and K. Rajamani, "Dynamic power management for embedded systems," in *Proc. IEEE Int. SOC Conf.*, 2003, pp. 416–419.

[28] S. M. Martin, K. Flautner, T. Mudge, and D. Blaauw, "Combined dynamic voltage scaling and adaptive body biasing for lower power microprocessors under dynamic workloads," in *Proc. ICCAD*, 2002, pp. 721–725.

[29] K. J. Nowka, G. D. Carpenter, E. W. MacDonald, H. C. Ngo, B. C. Brock, K. I. Ishii, T. Y. Nguyen, and J. L. Burns, "A 32-bit PowerPC system-on-a-chip with support for dynamic voltage scaling and dynamic frequency scaling," *IEEE J. Solid-State Circuits*, vol. 37, no. 11, pp. 1441–1447, Nov. 2002.

[30] S. P. Mohanty, N. Ranganathan, and K. Balakrishnan, "Design of a low power image watermarking encoder using dual voltage and frequency," in *Proc. 18th Int. Conf. VLSI Design*, 2005, pp. 153–158.

[31] N. Ranganathan, N. Vijaykrishnan, and N. Bhavanishankar, "A linear array processor with dynamic frequency clocking for image processing applications," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 8, no. 4, pp. 535–445, Aug. 1998.

[32] V. Krishna, N. Ranganathan, and N. Vijaykrishnan, "Energy efficient datapath synthesis using dynamic frequency clocking and multiple voltages," in *Proc. 12th Int. Conf. VLSI Design*, 1999, pp. 440–445.

[33] C. Lichtenau, M. I. Ringler, T. Pfluger, S. Geissler, R. Hilgendorf, J. Heaslip, U. Weiss, P. Sandon, N. Rohrer, E. Cohen, and M. Canada, "PowerTune: advanced frequency and power scaling on 64b PowerPC microprocessor," in *IEEE ISSCC Dig. Tech. Papers*, vol. 1, 2004, pp. 356–357.
[34] Intel Corp. (2004) Wireless Intel SpeedStep Power Manager White Paper. [Online]. Available: http://www.intel.com/design/pca/applicationsprocessors/whitepapers/300577.htm
[35] W. Kim, J. Kim, and L. M. Sang, "Dynamic voltage scaling algorithm for fixed-priority real-time systems using work-demand analysis," in *Proc. ISLPED*, 2003, pp. 396–401.
[36] T. Pering, T. Burd, and R. Brodersen, "The simulation and evaluation of dynamic voltage scaling algorithms," in *Proc. ISLPED*, 1998, pp. 76–81.
[37] A. Qadi, S. Goddard, and S. Farritor, "A dynamic voltage scaling algorithm for sporadic tasks," in *Proc. 24th IEEE RTSS*, 2003, pp. 52–62.
[38] R. M. Secareanu, D. Albonesi, and E. G. Friedman, "A dynamic reconfigurable clock generator," in *Proc. 14th Annu. IEEE Int. ASIC/SOC Conf.*, 2001, pp. 330–333.
[39] J. Zhou and H. Chen, "A 1 GHz 1.8 V monolithic CMOS PLL with improved locking," in *Proc. MWSCAS*, vol. 1, 2001, pp. 458–461.
[40] A. Chakraborty and M. R. Greenstreet, "A minimal source-synchronous interface," in *Proc. 15th Annu. IEEE Int. ASIC/SOC Conf.*, 2002, pp. 443–447.
[41] A. Chakraborty and M. R. Greenstreet, "Efficient self-timed interfaces for crossing clock domains," in *Proc. 9th Int. Symp. Asynchronous Circuits and Systems*, 2003, pp. 78–88.
[42] T. Chelcea and S. M. Nowick, "A low-latency asynchronous FIFO's using token rings," in *Proc. 6th ASYNC*, 2000, pp. 210–220.
[43] T. Chelcea and S. M. Nowick, "Robust interfaces for mixed-timing systems with application to latency-insensitive protocols," in *Proc. Design Automation Conf.*, 2001, pp. 21–26.
[44] W. J. Bainbridge and S. B. Furber, "Delay insensitive system-on-chip interconnect using 1-of-4 data encoding," in *Proc. 7th ASYNC*, 2001, pp. 118–126.
[45] N. C. Paver and D. A. Edwards, "Is asynchronous logic good for low-power?," in *IEE Colloq. Low Power Analogue and Digital VLSI: ASICS*, 1995, pp. 4/1–4/5.
[46] M. Singh and S. M. Nowick, "MOUSETRAP: ultra-high-speed transition-signaling asynchronous pipelines," in *Proc. ICCD*, 2001, pp. 9–17.
[47] S. Schuster, W. Reohr, P. Cook, D. Heidel, M. Immediato, and K. Jenkins, "Asynchronous interlocked pipelined CMOS circuits operating at 3.3–4.5 GHz," in *IEEE ISSCC Dig. Tech. Papers*, 2000, pp. 292–293.
[48] V. I. Varshavsky and V. B. Marakhovsky, "One-two-one track asynchronous FIFO," in *Proc. APCCAS*, 1998, pp. 743–746.
[49] I. Sutherland and S. Fairbanks, "GasP: a minimal FIFO control," in *Proc. 7th ASYNC*, 2001, pp. 46–53.
[50] S. Laberge and R. Negulescu, "An asynchronous FIFO with fights: case study in speed optimization," in *Proc. 7th ICECS*, vol. 2, 2000, pp. 755–758.
[51] M. Shams, J. C. Ebergen, and M. I. Elmasry, "Modeling and comparing CMOS implementations of the C-element," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 6, no. 4, pp. 563–567, Dec. 1998.
[52] D. L. Oliveira, M. Strum, W. J. Chau, and W. C. Cunha, "Synthesis of multi bursts-mode controllers using generalized C-elements," in *Proc. IX Workshop Iberchip*, 2003.
[53] K. Y. Yun and R. P. Donohue, "Pausible clocking: a first step toward heterogeneous systems," in *Proc. ICCD*, 1996, pp. 118–123.
[54] K. Y. Yun and A. E. Dooply, "Pausible clocking-based heterogeneous systems," *IEEE Trans. Very Large Scale Integr. (VLSI) Systems*, vol. 7, no. 4, pp. 482–488, Dec. 1999.
[55] R. Ginosar, "Fourteen ways to fool your synchronizer," in *Proc. 9th ASYNC*, 2003, pp. 89–96.
[56] C. Mead and L. Conway, *Introduction to VLSI Systems*. Reading, MA: Addison-Wesley, 1980, pp. 218–262.
[57] R. Watn, T. Njolstad, F. Berntsen, and J. F. Lonnum, "Independent clocks for peripheral modules in system-on-chip design," in *Proc. IEEE Int. SOC Conf.*, 2003, pp. 25–28.
[58] M. De Clercq and R. Negulescu, "1.1-GDI/s transmission between pausible clock domains," in *Proc. ISCAS*, vol. 2, 2002, pp. 768–771.
[59] S. Yunhua, S. Shimin, L. Yue, and J. Lijiu, "Implementation of a 6.5 MHz 34-B NCO," in *IEEE ISSCC Dig. Tech. Papers*, 1995, pp. 205–207.
[60] I. Janiszewski, B. Hoppe, and H. Meuth, "Numerically controlled oscillators with hybrid function generators," *IEEE Trans. Ultrason., Ferroelectr., Freq. Contr.*, vol. 49, no. 7, pp. 995–1004, Jul. 2002.
[61] R. Ertl and J. Baier, "Increasing the frequency resolution of NCO-systems using a circuit based on a digital adder," *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 43, no. 3, pp. 266–269, Mar. 1996.
[62] T. M. Almeida and M. S. Piedade, "High performance analog and digital PLL design," in *Proc. ISCAS*, vol. 4, 1999, pp. 394–397.
[63] C.-H. To, C.-F. Chan, and O. C.-S. Choy, "A simple CMOS digital controlled oscillator with high resolution and linearity," in *Proc. ISCAS*, vol. 2, 1998, pp. 371–373.
[64] J.-S. Chiang and K.-Y. Chen, "The design of an all-digital phase-locked loop with small DCO hardware and fast phase lock," *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 46, no. 7, pp. 945–950, Jul. 1999.

Atanu Chattopadhyay received the the B.Eng. degree in computer engineering and the M.Eng. degree in electrical engineering (microelectronics and computer systems) from McGill University, Montreal, QC, Canada, in 2000 and 2003, respectively. He is currently pursuing the Ph.D. degree at McGill University, researching clock generation and distribution for ASICs and FPGAs with a doctoral bursary from the FQRNT. He has also been a teacher's assistant and a course instructor for various classes at McGill University.

Zeljko Zilic (M'97) received the Dipl. Ing. degree in computer engineering from the University of Zagreb, Zagreb, Croatia, in 1989, the M.Sc. degree in computer and electrical engineering from the University of Toronto, Toronto, ON, Canada, in 1993, and the Ph.D. degree from the University of Toronto in 1997.

From 1990 to 1992, he was a Faculty Lecturer at the University of Zagreb and Research Engineer at the Electrical and Computer Engineering and Computer Science Departments of the University of Toronto as well as for the Computer Systems Research Institute in Toronto. From 1997 to 1998, he was a Member of the Technical Staff at the FPGA Division of the Microelectronics Group of Lucent Technologies. Currently, he is an Associate Professor at McGill University, Montreal, QC, Canada, and the Director of McGill's Microelectronics and Computer Systems Laboratory. He holds four patents in the area of clock and power management and has co-authored the book *Verification by Error Modeling* (New York: Springer, 2003).

Prof. Zilic is a recipient of a Chercheur Strategique research chair from province of Quebec. He has received the Myril B. Reed Best Paper Award at the IEEE International Midwest Symposium on Circuits and Systems in 2001 and received the 2005 Design and Verification Conference (DVCon05) Best Paper Award. He has served as a member of the Technical Program Committees of the ACM International Symposium on FPGAs, the IEEE International Test Conference, the Midwest Circuits and Systems Symposium and the Electronic Circuits and Systems Conference. He is also on the editorial boards of the *Journal of Multiple-Valued Logic and Soft Computing* and the *International Journal of Software and Information Technologies*.