

Circuit Implementation of a 600MHz Superscalar RISC Microprocessor

M. Matson, D. Bailey, S. Bell, L. Biro, S. Butler, J. Clouser,
J. Farrell, M. Gowan, D. Priore, and K. Wilcox
Compaq Computer Corporation, Shrewsbury, MA

Abstract

The circuit techniques used to implement a 600MHz, out-of-order, superscalar RISC Alpha microprocessor are described. Innovative logic and circuit design created a chip that attains 30+ SpecInt95 and 50+ SpecFP95, and supports a secondary cache bandwidth of 6.4GB/s. Microarchitectural techniques were used to optimize latencies and cycle time, while a variety of static and dynamic design methods balanced critical path delays against power consumption. The chip relies heavily on full custom design and layout to meet speed and area goals. An extensive CAD suite guaranteed the integrity of the design.

1. Introduction

The design of the Alpha 21264 microprocessor [1] was driven by a desire to achieve the highest performance possible in a single chip, 0.35um CMOS microprocessor. This goal was realized by combining low instruction latencies and a high frequency of operation with out-of-order issue techniques. The microprocessor fetches four instructions per cycle and can issue up to six simultaneously. Large 64KB, two-way set associative, primary caches were included for both instructions and data; a high bandwidth secondary cache interface transfers up to 6.4GB/s of data into or from the chip. A phase-locked loop [2] generates the 600MHz internal clock. The increased power that accompanies such high frequencies is managed through reduced VDD, conditional clocking, and other low power techniques.

The Alpha microprocessor road map dictates continual improvements in architecture, circuits, and fabrication technology with each successive generation. In comparison to its predecessors [3-5], the 21264 issues instructions out-of-order, supports more in-flight instructions, executes more instructions in parallel, has much larger primary caches and memory bandwidth, and contains additional integer and floating point function units. Other differences include a phase-locked loop to simplify system design, as well as conditional clocks and a clocking hierarchy to reduce power consumption and permit critical path tradeoffs. Custom circuit design enabled the incorporation of these advances while reducing the cycle time more than possible with the process shrink alone. The new 0.35um (drawn) process provides faster devices and die area for

more features, along with reference planes for better signal integrity and power distribution.

The remainder of this paper will describe the 21264's physical characteristics, design methodology, and major blocks, paying particular attention to the underlying design problems and implementation approaches used to solve them. The paper will conclude with a discussion of how these strategies created a microprocessor with leading edge performance.

2. Physical Characteristics

Characteristics of the CMOS process are summarized in Table 1. The process provides two fine pitch metal layers, two coarse pitch metal layers, and two reference planes. The pitch of the finer layers aids in compacting the layout, while the lower resistance of the coarser layers is beneficial for clocks and long signal wires. The reference planes lower the effective impedance of the power supply and also provide a low inductance return path for clock and signal lines. Moreover, they greatly reduce the capacitive and inductive coupling between the wires, which could otherwise induce reliability failures due to voltage undershoot or overshoot, functional failures caused by excessive noise, and wide variations in path delays due to data dependencies. Gate oxide capacitors, placed near large drivers and underneath upper level metal routing channels, further diminish power supply noise.

Feature size	0.35um
Channel length	0.25um
Gate oxide	6.0nm
VTXn/VTXp	0.35V / -0.35V
Metal 1, 2	5.7kA AlCu, 1.225um pitch
Reference plane 1	14.8kA AlCu, VSS
Metal 3, 4	14.8kA AlCu, 2.80um pitch
Reference plane 2	14.8kA AlCu, VDD

Table 1: CMOS Process Technology

The microprocessor is packaged in a 587 pin ceramic interstitial pin grid array. A CuW heat slug lowers the thermal resistance between the die and detachable heat sink. The package has a 1uF wirebond attached chip capacitor in addition to the distributed on-chip decoupling capacitors.

To ensure a low impedance path for supply current, 198 of the pins are connected to VDD or VSS.

Careful placement of function units minimizes communication delays, as shown in the chip floorplan (Figure 1). The caches are at the bottom of the die, surrounded by memory data pads. Instructions from the instruction cache go to the adjacent prefetcher (Instr Fetch), and are then driven to the floating point and integer instruction queues (FP IBox, Int IBox). The queues are beside their function units (FBox, EBox). Execution results are ultimately passed to a common load/store data bus running across the middle of the die. The data translation buffers (MBox) are immediately between the data bus and data cache. The load and store queues are directly above this bus.

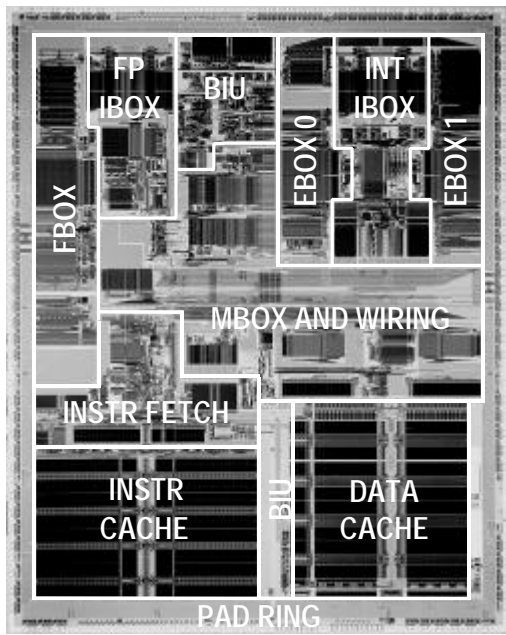


Figure 1: Chip Micrograph

3. Methodology

The design methodology helped engineers create a functional chip in a timely fashion that met performance goals. Certain methodology rules were rigidly enforced. These pertained to functionality issues such as signal degradation due to capacitive coupling and charge redistribution, signal races, clock distribution, and latching. Other rules embodied preferences that would increase productivity. For example, dynamic logic and conditional clocks were generally confined to datapaths, where their speed and area advantages better justified their circuit verification penalty. Designers were strongly encouraged to select latches from a library of standard types. The circuits and layout are predominantly full custom, although

some sections were synthesized from behavioral models where feasible.

Circuits were rigorously verified. Special purpose CAD tools analyzed parasitic capacitance effects, clock networks, noise immunity, and charge sharing. Critical paths were simulated with SPICE. Timing tools identified signal race and path delay failures. Certain enigmatic circuits were verified by hand analysis with SPICE.

4. Major Functional Units

The microprocessor comprises a base clock generator and clock distribution network, instruction cache and prefetcher, branch predictor, register map and instruction issue logic, integer unit, floating point unit, data cache, memory management unit, and system and secondary cache interface. We will discuss the key implementation features in the following subsections.

4.1. Clock Distribution

The Alpha 21264 has a single-node, gridded, two-phase global clock, GCLK, that covers the entire die, similar to the Alpha 21064 [3] and 21164 [4]. It is fundamentally different from previous Alpha microprocessors, however, because it incorporates a hierarchy of clock stages that extend beyond GCLK.

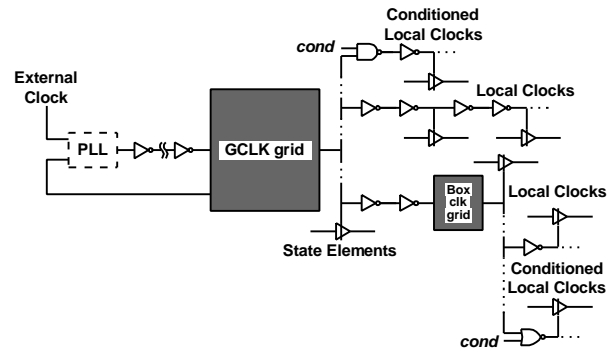


Figure 2: Clock Distribution

The clock distribution network [6] of the Alpha 21264 is shown in Figure 2. Local clocks and local conditional clocks are driven several stages past GCLK. In addition, there are six other major clocks with upper-level metal grids juxtaposed with GCLK but shielded with VDD or VSS from it. These major clocks drive large grids over the instruction fetch, integer, floating point, load/store, and bus interface units, as well as the pads. They also drive local clocks and local conditional clocks.

The reasons for implementing this clocking hierarchy are to improve performance and to save power. Multi-level buffering in the clock path allows circuit designers to remedy critical paths and race problems by adjusting the number of buffers between driving and receiving state elements and thus "borrow" time from temporally adjacent

clock cycles. The clock hierarchy saves power through clock driver placement and conditional clocks. The local clock drivers can be placed near their loads, reducing the routing capacitance and permitting smaller drivers. Conditional clocking reduces peak power when execution exclusivity is exploited.

4.2. Instruction Cache

A 64KB cache supplies instructions to the out-of-order issue logic. To satisfy the consumption rate of the superscalar function units, a block of four 42 bit predecoded instructions must be fetched every cycle. However, the fetch unit requires a cycle to analyze the results of the previous fetch for any stalls, traps, or branches before updating the program counter (PC). This restriction led to the dilemma of needing to fetch every cycle but not having a valid PC until the end of the fetch cycle. Moreover, performance modeling showed that the fetch stream needed to be more efficient (as measured by the mispredict rate) than in previous Alpha implementations, in order to mitigate the penalty of issuing invalid instructions into the deeper, out-of-order pipeline.

A line prediction mechanism was implemented to meet the above architectural requirements and the 600MHz frequency goal. The instruction cache address of the next fetch block is stored within each fetch block. Due to the performance requirements this mechanism needed to address the cache directly. However, the fetch unit also needed access to the cache for traps, stalls, mispredicts, and fills. The circuit shown in Figure 3 was used to mux between the three possible addresses (Line Prediction, PC, and Fill) and to latch the result, without incurring any additional gate delays to accommodate the fetch unit intrusions. The select wires are exclusive, so only one address port, all of which are predecoded in order to reduce array access time, can write the latch used to hold the final fetch address. The fetch unit cannot determine if the current fetch needs to be stalled until it is too late to stop the next fetch and all of the addressing paths. Therefore, the stall signal is fed into the cache directly, stopping the port selects from selecting and the fetch address latches from reloading. This allows the cache to enter a stalled state immediately.

The 64KB cache was implemented as eight 8KB banks, in order to reduce array access times and to accommodate redundancy. An indirect benefit of this was that the replicated address logic permits multiple bank accesses to distinct addresses. By allowing a fetch and a fill simultaneously to differing banks, the need for a stream buffer was eliminated. Incoming fills, both resulting from demand misses and prefetches, are written directly into the cache. Performance modeling found this to have only a slight effect (either positive or negative) on overall performance, depending highly on the program trace being tested. The circuit implications of removing the stream

buffer were tremendous, allowing for faster cache fetches and greatly simplified control.

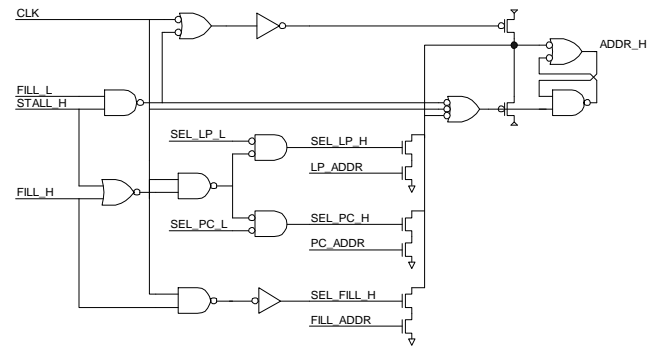


Figure 3: Instruction Cache Address

4.3. Register Map and Instruction Issue Logic

The register map and instruction issue logic (IBox) govern instruction execution in the Alpha 21264. The pipeline has one cycle for mapping instruction registers from architectural space to physical space, and then an additional cycle to issue the instructions from the instruction queue [7]. There are separate mappers and queues for integer and floating point instructions, which eases the routing difficulty between the queues and the associated execution units. Instructions are received by each mapper from the instruction fetch unit, and then the register mapping operation is performed in parallel with the instruction decode operation. The decoded instruction and the mapped physical register numbers are forwarded to the corresponding instruction queue. An enqueued instruction is able to issue immediately to an execution unit if all of its operands and an execution unit are available. Once the instruction is complete, a separate unit retires it. The architectural state of the microprocessor is committed in order; consequently the retire unit must keep track of all instructions that are in flight and only retire instructions that have had all older instructions retired as well.

To achieve very high performance for cycle time and instructions per cycle, tradeoffs were made in the architecture, circuits, and floorplan to meet the overall target. Two examples of the types of decisions made illustrate the custom design methodology on the 21264: the mapper to queue interface and the instruction decode logic.

The mapper is designed as a flat array of physical registers, whose output is a set of 12 registers: two sources and one destination for each of four instructions in the fetch block. Ideally, this decoded array of registers would be located directly next to the decoded register scoreboard array in the queue for easy one-to-one information transfer. However, the floorplan did not allow an area as large as the combined mapper and queue to exist without significantly impeding the communication between the queue and the execution units. Compounding the problem is the need to

keep the pipeline as short as possible to minimize the latency of a redirect in the instruction stream. The compromise, as shown in Figure 4, is a bus of twelve small-signal, differential register buses that encode the value of the physical register number. This arrangement allows the final mapping operation and the encoding to occur in the same phase, with a sense amplifier receiving the encoded register in the queue at the beginning of the next phase. The sense amplifier then drives into a NOR-type decoder that drives register values into the scoreboard array.

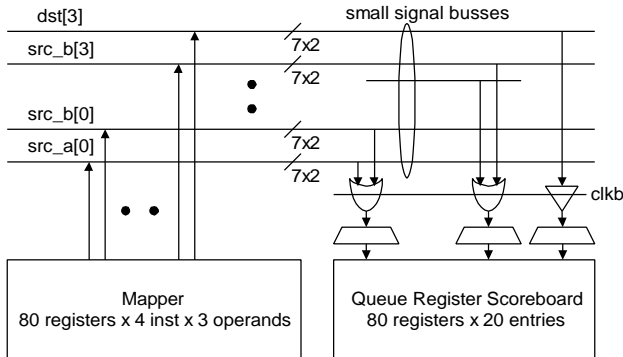


Figure 4: Register Mapper

The instruction decode operation that occurs in parallel with the mapping operation is performed by a PLA AND plane into random logic. This one cycle operation allows one phase for the AND plane, with its benefits of easy programming and logic density, and one phase for the random logic, which is fast and provides versatility for logic modifications to accommodate changes required by downstream logic. Each instruction in the fetch block has a dedicated decode logic block. Careful architectural decisions were made to ensure a minimum amount of logic was required across a fetch block, as this would require either a serial path after each instruction was decoded, or separate, parallel decoders that examined four instructions at once.

4.4. Integer Execution Unit

The integer execution unit (EBox) is divided into two clusters. Each cluster is broken up into an upper and lower sub-cluster. The IBox can issue up to one instruction to each sub-cluster from the instruction queue every cycle. The lower sub-clusters contain a logic unit and a general purpose adder that also performs address calculations for loads and stores. The upper sub-clusters contain a logic unit, an adder, a shifter, and an integer multiplier or a motion-video-instruction unit (MVI). All functional units are single cycle operations except for the multiplier and MVI.

Each cluster contains an 80 entry register file with four read ports and six write ports. The two register files are kept coherent; cross-cluster updates incur a one cycle

penalty. The EBox determines when to forward instruction results, comparing six register file writes (two local destination registers, two cross-cluster destination registers, and two load result registers) to the four source operand registers. The 24 results of these CAMs are broadcast to the sub-clusters, sixteen to the lower sub-cluster and eight to the upper sub-cluster. These signals are then combined with the functional unit conditional clock to generate a pulsed bypass signal that drives into the top of a two-high nMOS pulldown stack onto the low swing, differential operand bus.

The result bypass mechanism may be seen in Figure 5. The front end of each functional unit contains a sense amplifier that receives the operand bus. The firing of the sense amplifiers is controlled by conditional clocks, so that only one functional unit can sense the data on the bus. The conditional clock that fires the sense amplifier also enables an nMOS pulldown on a dynamic node that controls the precharging of the operand bus. Once the data on the bus is captured by a functional unit, plus some amount of delay to guarantee the hold time requirements of the sense amplifier, the data is precharged away.

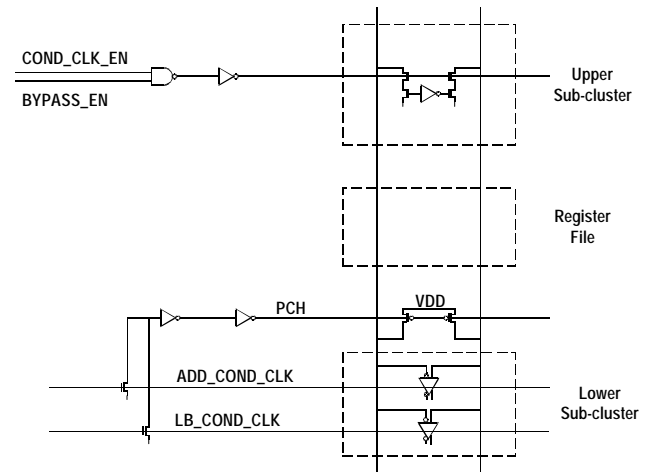


Figure 5: Bypassing from an Upper Functional Unit to a Lower Functional Unit

4.5. Floating Point Unit

The floating point unit (Fbox) comprises a register file and interface logic, an add/sub/convert pipeline, a multiplier pipeline, an iterative divider, and an iterative square root. These sections can simultaneously receive and execute multiply, add/sub/convert, and up to two load/store instructions from the processor's floating point instruction queue. Data can be transferred directly between integer and floating point register files. Moreover, both divide and square root instructions can execute concurrently with the foregoing instructions. The multiplier has its own final adder that includes rounding circuitry, while the add/sub/convert, divider, and square root sections share a

common final adder. The FBox implements both IEEE and VAX data formats and rounding modes.

Circuit speed was the primary motivation in the design of the floating point unit. The 1.7ns cycle time and four cycle add/sub/convert and multiply latencies allow only 50 gate delays for these operations. Consequently techniques such as latchless pipelining, amplification of low swing signals, "one hot" dynamic logic, and functional merging were essential. However these techniques were applied judiciously in order to manage power consumption. Low power static circuits were used where possible; a reduced supply voltage and conditional clocking were also beneficial. The conditional clocks led to race issues that were carefully controlled with circuitry to match the delays of complementary clocks, a special race analysis tool, and by limiting clock skew to 60ps.

The FBox divider contains an example of these circuit techniques. This divider uses a reciprocal and quotient approximation method. At each iteration a new quotient digit is determined by multiplying the current partial remainder by the divisor's approximate reciprocal. The product of this new digit and the divisor becomes a correction term which is subtracted from the current remainder to form the next one. Each iteration is accomplished in one cycle using a modified Wallace tree multiplier that generates the product in sum and carry form in one phase and a recoder that transforms the product's MSB's in the next. The recoder directly converts the upper bits of the multiplier's carry-save vectors to a radix-4 representation, avoiding the need for a costly carry propagate addition. High speed is achieved by encoding the signed digit set with "one hot" dynamic logic and by merging the addition and recode into one logic stage. Consequently the divider is able to generate eight quotient bits per cycle.

4.6. Data Cache

The chip contains a 64KB, two-way set associative, virtually indexed, physically tagged, ECC protected, write-back data cache which can satisfy up to two independent quadword loads per cycle. The cache is banked, but there are no bank conflict restrictions relating to loads. Instead, the cache makes use of a bit line static-load scheme which allows each array to be accessed up to twice per cycle despite the use of a conventional single ported 6T SRAM cell (Figure 6). When not busy with loads, the cache can write up to one octaword per cycle. In this case, the stale octaword is ejected atomically with the write operation in the same cycle to facilitate victim write-back in the case of fill operations and ECC processing in the case of byte-masked store operations. The cache cannot mix load and store operations within a given cycle, but there are no such restrictions from cycle to cycle. At the 600MHz chip operating frequency, the cache can support simultaneous peak read and write bandwidths of 9.6GB/s during store and

fill processing. Peak load bandwidth is 9.6GB/s after muxing based on set selection.

The bit line static-load scheme effectively doubles the bandwidth of the cache by eliminating the bit line precharge phase of the more traditional design. Each bit line is driven high by a static pMOS load which is deactivated only to write the array. This pMOS load must be carefully sized. Larger pMOS loads result in faster bit lines but with smaller signals for sensing. Each array can be accessed up to once per clock phase. During write operations, which take an entire cycle, the first phase is used to read the stale data and the second to write the array. Because this write may be followed by an unrelated read in the very next phase, care was taken to insure an adequate bit line precharge following this operation. The cache consists of eight banks, each clocked conditionally on a per phase basis to save power. When processing loads, the cache is presented with up to two independent indices per cycle. In one case, the decode proceeds without delay to access the specified array in the first available phase. Data from this operation is then piped by a phase awaiting set select muxing. In the second case, the decoded index is piped by a phase before the array is accessed. Data from this operation is available just in time for set select muxing. The two cases appear identical to the rest of the chip, supporting three cycle latency for integer operations (i.e. issue load, wait, wait, issue consumer) and four-cycle latency for floating point operations. Store data to the cache is buffered at the top in a queue before being committed. Load data can be bypassed from this queue under control of the memory management unit.

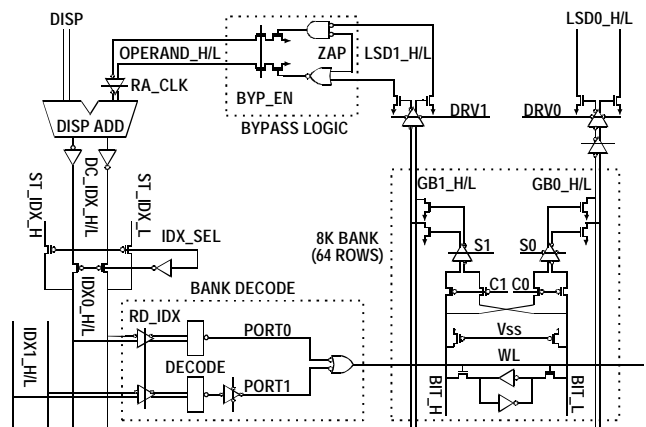


Figure 6: Data Cache

4.7. Memory Management Unit

The memory management unit (MBox) consists of two load/store pipelines, two data translation buffers, two reorder queues, and control for the store queue. A load/store pipeline resides under each of the two integer execution unit clusters. Each pipeline has a separate data translation buffer to map addresses from virtual to physical

space. Each buffer has 128 entries; coherency is maintained by writing new mappings into each buffer simultaneously. Load/Store instructions access the data cache in parallel with their address translations. An eight entry miss address file processes cache misses. As long as a subsequent load/store instruction misses at the same cache block as a previous miss, the two instructions will merge into a single entry.

Although load/store instructions are received out-of-order, they are requeued in program order by separate load and store reorder queues. Store instructions are kept in their queue until they can be safely retired, and are then written into the data cache. Load instructions are allowed to accept data from the store data array based on a partial match of virtual addresses; this avoids the potentially long latency for store data being written into the data cache. Latency is further reduced by decoupling the address and data portions of load instructions. Load instructions can access the data cache tags while the cache data is filled in response to a previous miss. This hides some of the cache miss latency.

The load and store reorder queues ensure correct memory operation by accounting for load/store dependencies. Each queue entry tracks a load/store instruction's physical address, in-flight number, and status bits to detect any deviation from in-order programming. A dynamic global trap wire spanning both queues is asserted when an instruction needs to be killed and reissued. This is a critical task at such high clock frequencies. To combine the necessary number of logical minterms, phase stealing is employed (Figure 7). A delayed clock fires edge triggered sense amplifiers that provide both latching and clock-like data pulses that discharge the trap wire. This gives sufficient time to evaluate the trap equation while still meeting the IBox's setup time requirement. This technique improved circuit speed but made data races more likely. The races were blocked by opposite phase flow-through latches on the sense amplifier input data, or by anding the sense amplifier outputs with clock-like data from domino logic.

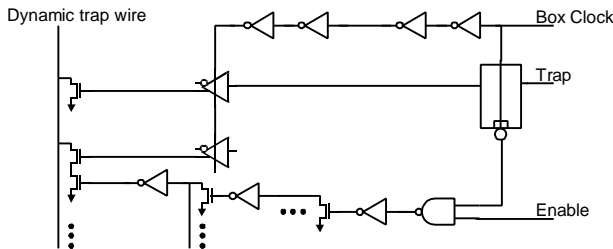


Figure 7: Memory Management Traps

4.8. System and Secondary Cache Interface

The system and secondary cache interface (CBox) transfers data into and out of the microprocessor. It

controls a private off-chip, second-level, direct mapped cache, which can range in size from 1 to 16MB. The cache interface supports many configurations, ranging from low cost synchronous SRAMs to high speed dual-data source synchronous SRAMs with peak transfer rates up to 6.4GB/s. System commands, addresses, and data are transferred to and from memory controllers and data switches through a separate port, achieving transfer rates of up to 3.2GB/s. The CBox includes address and data datapaths, a complex control section, and the I/O pad ring. The address datapath consists of the I/O write address merging logic and 4 entry queue, the address bypass logic, an 8 entry victim address file, an 8 entry system probe queue, the secondary cache tag compare logic, and the system addresses and command generation and decode logic. The data datapath comprises a 144 bit wide, 12 entry, dual write port, single read port RAM array which stores victim data from the primary data cache as well as I/O write data. In order to meet timing and area constraints, the address and data datapaths were implemented using full custom circuit design and layout methodologies.

The majority of the CBox area is dedicated to large and complex control structures. The CBox control logic is composed of several independent command pipelines and arbiters, along with many random logic control structures. The major challenge in implementing the CBox control logic was in structuring and optimizing the logic to meet the cycle time and die area goals. Early feasibility work demonstrated that relying exclusively on logic and layout synthesis would not satisfy these objectives. Consequently a limited, partitioned synthesis methodology was used. Many logic sections were generated using an in-house logic synthesis tool, and the results were incorporated into hand-drawn schematics. These schematics were then manually laid out, with the aid of in-house developed layout synthesis tools. With this methodology, logic changes generally affected only a small portion of the schematic and were easily incorporated into the schematics and layout. Also, timing analysis issues could be corrected locally without resynthesizing large sections.

The I/O pad ring features a 144 bit (128 bit data and 16 bit ECC) secondary cache data interface and a 72 bit (64 bit data and 8 bit ECC) system data interface. These data pads are located on the sides of the chip. The cache and system data I/O pads are interleaved to allow for the efficient muxing of incoming data onto the load/store bus. The cache tag and index pads are located on the top of the chip, and two 16 bit time-multiplexed unidirectional system address busses are located on the bottom of the die. All of the cache and system pads use source synchronous clocking, and they operate at a maximum clock frequency equal to two-thirds of the CPU clock rate. In addition, the pad ring contains a number of miscellaneous I/O, including a fully IEEE 1149.1 compliant JTAG interface, interrupts, and support for an on-chip phase-locked loop.

Figure 8 shows an example of the typical bidirectional cells used for both the system and secondary cache interfaces. Each group of bidirectional data bits has a unidirectional output and a unidirectional input source synchronous clock. Output data is first buffered in the output stage silo, which provides data steering for dual data support. Output transfers are updated on both rising and falling edges of INTFCLK. This data is then applied to the bidirectional data pin (FDATA). In parallel with the data transfer, clock FCLKOUT is generated. This clock is skew aligned to the data pins.

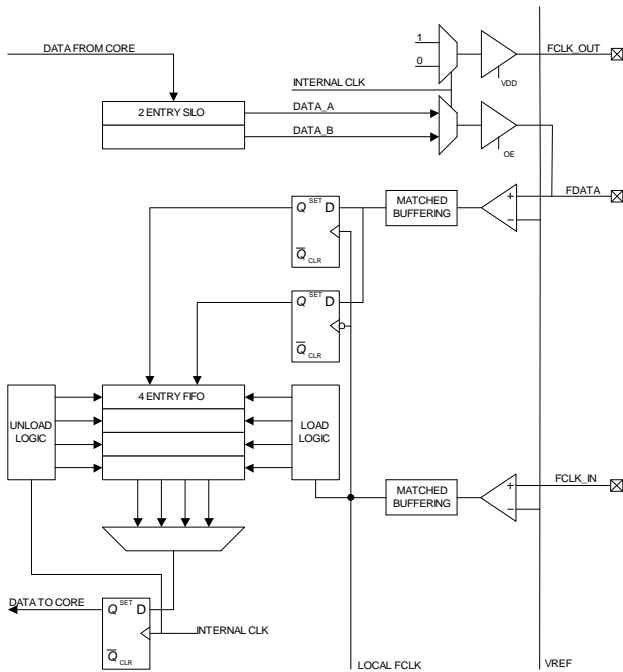


Figure 8: Bidirectional Data Interface

When operating as a receiver, external logic provides both data (FDATA) and a skew aligned clock (FCLKIN). The Alpha 21264 buffers the received clock to provide adequate drive for latching the received input data. Data is similarly buffered to control skew alignment. Edge triggered latches are then used to sample pin data. A 4 entry FIFO transfers data from the source synchronous clocked domain to the internal CPU core clock domain. Load and Unload logic properly positions this sampling window.

5. Conclusion

The implementation of the Alpha 21264 required careful orchestration of the competing demands posed by architectural, circuit, floorplanning, and process considerations. Simultaneous consideration of these constraints led to a design that was optimized across many domains, a design that incorporates out-of-order issue techniques and a large memory bandwidth coupled with

leading edge clock frequencies. Aggressive performance goals were achieved through innovative static and dynamic circuit techniques, a heavy emphasis on full custom design and layout, and a willingness to modify the microarchitecture for the benefit of circuit speed. CAD tools that rigorously ensured the reliable operation of this circuitry were equally important.

Acknowledgments

The authors gratefully acknowledge the contributions of the entire 21264 team, all of whom helped ensure the success of this project.

References

- [1] Gieseke, B., et al., "A 600MHz Superscalar RISC Microprocessor with Out-Of-Order Execution," ISSCC Digest Technical Papers, Feb., 1997, pp. 176-177.
- [2] von Kaenel, V., et al., "A 600MHz CMOS PLL Microprocessor Clock Generator with a 1.2GHz VCO," ISSCC Digest of Technical Papers, Feb., 1998, pp. 396-397.
- [3] Dobberpuhl, D., et al., "A 200MHz 64b Dual-Issue CMOS Microprocessor," IEEE Journal of Solid-State Circuits, Vol. 27, No. 11, Nov., 1992.
- [4] Bowhill, W., et al., "A 300MHz 64b Quad-Issue CMOS Microprocessor," ISSCC Digest of Technical Papers, pp. 182-183, Feb., 1995.
- [5] Gronowski, P. et al., "A 433MHz 64b Quad-Issue RISC Microprocessor," ISSCC Digest of Technical Papers, pp. 222-223, Feb., 1996.
- [6] Fair, H., and Bailey, D., "Clocking Design and Analysis for a 600MHz Alpha Microprocessor," ISSCC Digest of Technical Papers, Feb., 1998, pp. 398-399.
- [7] Fischer, T., and Leibholz, D., "Design Tradeoffs in Stall-Control Circuits for 600MHz Instruction Queues," ISSCC Digest of Technical Papers, Feb., 1998, pp. 232-233.