Design and Development of a 120-MHz Bus Interface Block Using Standard Cells and Automatic Place and Route Tools

The RW_IO block runs at 120 MHz and interfaces the master memory controller chip's 60-MHz core with the 120-MHz processor bus drivers. A design approach using standard cells, automatic place and route tools, and a powerful database management and build tool was used to construct the RW_IO block. This approach was chosen over a full custom or data-path solution because of its reduced risk and the flexibility of the design tools.

by Robert E. Ryan

The master memory controller chip is one of three ASICs (application-specific integrated circuits) that make up the memory subsystem of certain HP workstations and business servers. The chip interfaces with the processor via the processor bus, which runs at 120 MHz. It also connects to the other memory subsystem components via the memory bus, which runs at 60 MHz. Most of the internal logic of the chip runs at 60 MHz. The RW_IO block, a portion of the chip's logic that runs at 120 MHz, transfers data between the processor bus drivers and the 60-MHz core logic.

Functional Description

The RW_IO high-speed interface block is designed to transfer data and control between the processor bus running at 120 MHz and the core logic of the master memory controller chip, which runs at 60 MHz. To transfer data from the processor bus efficiently requires twice the bandwidth at the 60-MHz interface. In operation, two cycles of the 120-MHz data and control signals are registered and then the registered information is transferred on the next rising edge of the 60-MHz clock. Fig. 1 is a schematic diagram of the RW_IO input logic.

The data transfer is synchronized by delaying the 60-MHz clock by 4.166 ns with respect to the rising edge of the 120-MHz clock (Fig.2). This synchronization scheme requires that the register-to-register transfer of data from the 120-MHz to the 60-MHz clock domains occur within 4.166 ns worst case. The reverse principle is used for transferring data from the core of the master memory controller chip out onto the processor bus. Data and control information is transferred from the 60-MHz to the 120-MHz domains within the RW_IO block. This information is then multiplexed at the 120-MHz rate and presented to the processor bus drivers.

A special case exists when the processor bus is idle and we wish to present the data directly from the 60-MHz core to the processor bus without registering and multiplexing at the 120-MHz rate. A special path (fast path) exists to handle this case. This path also has the requirement of transferring data from the 60-MHz to the 120-MHz domains within 4.166 ns. Fig. 3 illustrates the timing of the fast path, and Fig. 4 is a schematic diagram of the RW_IO output logic.

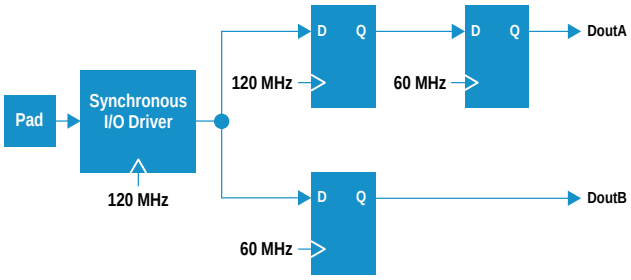

Fig. 1. Input logic of the RW_IO block.

Design Architecture

Five distinct functions are required to transfer data between the processor bus and the core logic. With these five

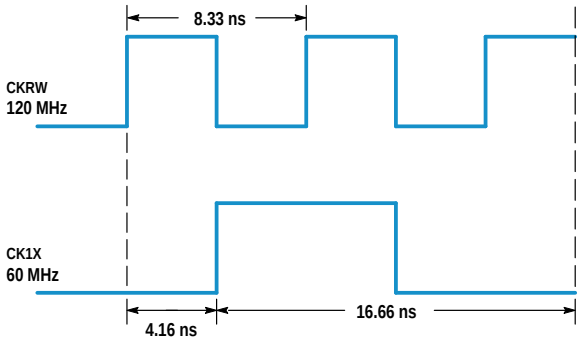

Fig. 2. Relationship of the 120-MHz clock to the 60-MHz clock.

basic functions defined, a bit slice design approach was taken to implement the entire functionality. A bit slice macro was built for each of the five functions using standard cells and Cell3 place and route tools from Cadence Design Systems, Inc. The bit slices were stacked together horizontally to create the complete RW_IO interface block. The following table defines the five bit slice functions and their macro names.

Function	Name
Input only, 2 bits	RWI2BS
Input only, 3 bits	RWI3BS
Bidirectional transfer with fast path	RWIOBS
Bidirectional transfer with fast path and processor bus output gated with data_valid	RWIOVBS
Bidirectional transfer with fast path and multiplexing for address and data	RWADIOBS

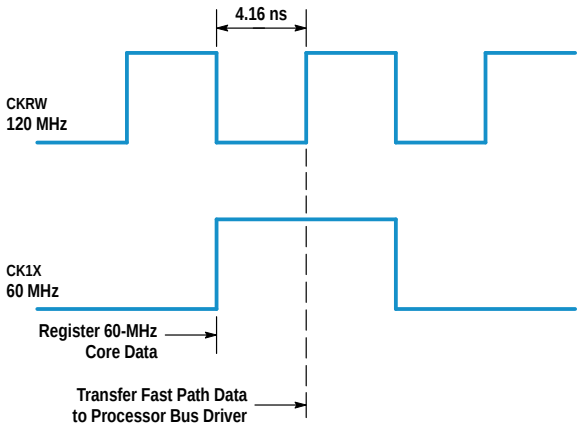

Fig. 3. Fast path timing.

Physical Bit Slice Construction

Because each of the five functions required a limited number of design elements, a semicustom implementation using standard cells was chosen. The most complicated function, RWADIOBS, required 19 cells. The five functions were placed and routed using a proprietary tool called Autopr[1] and Cadence Cell3. Autopr was used to generate the bit slice bounding box and standard cell rows and to place the I/O ports. Special care was taken in cell placement and the routing of the clock signals. Scan control signals and power and ground signals were routed so that connections could be made by abutment when the bit slices were stacked together. Once the construction of the bit slices was complete, Cell3 abstracts were generated for each bit slice using the Cadence Opus framework tools. These abstracts were then used to build up the complete RW_IO functionality.

Physical Block Assembly

The top-level Verilog netlist for the RW_IO block was generated by a script in such a manner that the individual bit slice functions were declared in the order which they physically appear from left to right within the RW_IO block. As part of the netlist generation a stackup file was created which defined the order of the bit slice elements, the pins on each bit slice corresponding to ports on the top and bottom of the block, and the direction of each pin (in or out). A simple awk program was developed that read the stackup file and created two command files for Cell3. The first file directed the placement of the pins on the block to line up with the appropriate bit slice. The second command file directed Cell3 to place the bit slice components in the order defined

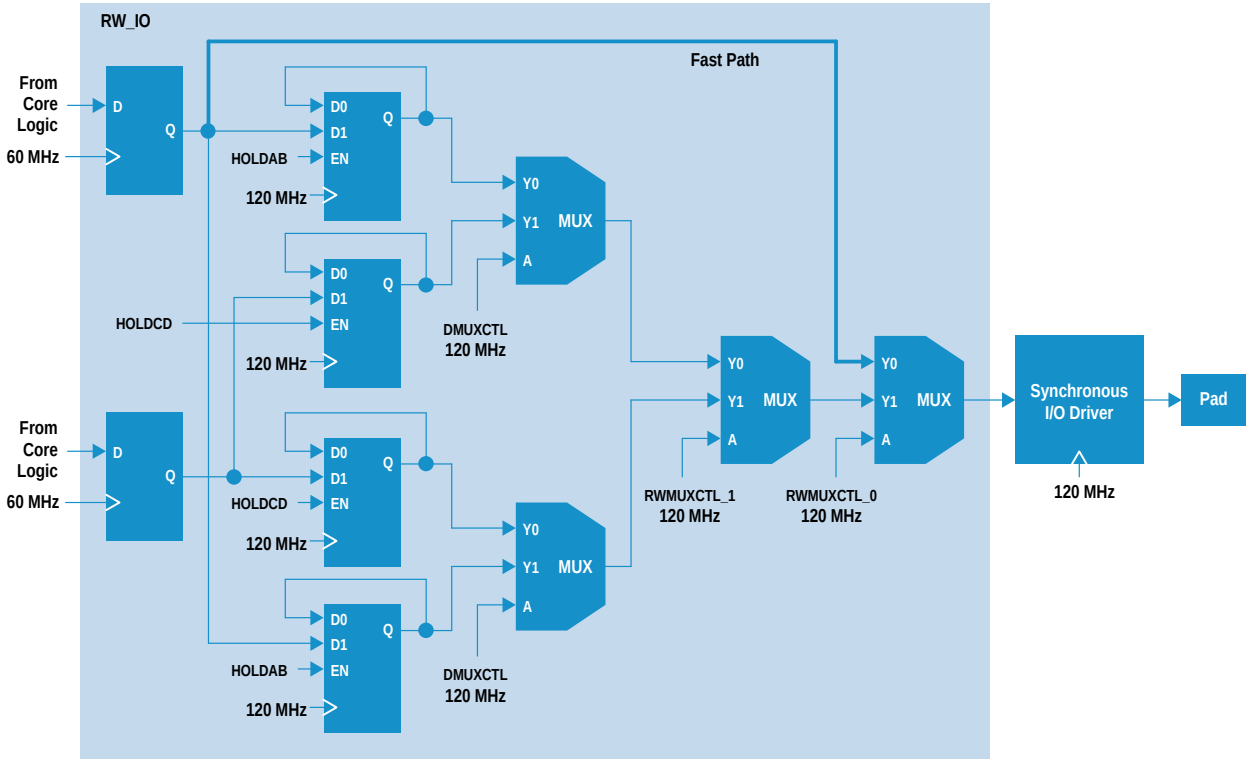

Fig. 4. Output logic of the RW_IO block (RWADIOBS function).

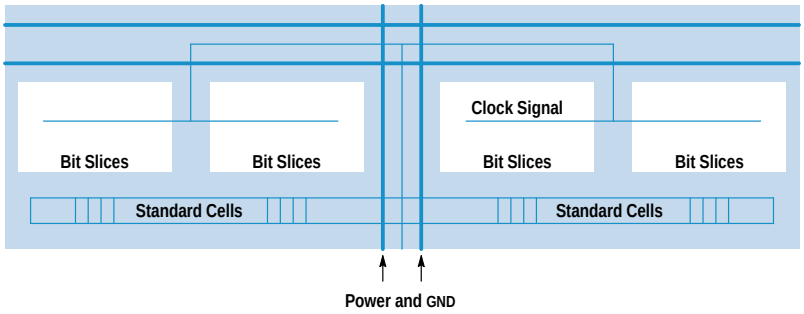

Fig. 5. RW_IO block layout.

by the stackup file. The RW_IO block was separated into four sections to allow the main power, ground, and clock signals to pass through the center of the block and to allow for clock tree taps (see Fig. 5).

The block has a single standard cell row along the bottom where clock buffers and a small amount of combinational logic reside. Cell3's clock tree synthesis capabilities were used to insert buffers in the global data control signals (hold_abL, hold_cdL, mux_ctrl_rw_0, mux_ctrl_rw_1, mux_ctrl_rdata, return_byte_swap). The routing of clock, V_{dd}, and GND signals was customized using specific Cell3 commands.

Characterization

In the early stages of development each bit slice function was characterized separately using Aida's Timver static timing analysis tool, with lumped capacitance and distributed RC delay information back-annotated from Cell3. Timing libraries were generated automatically by Timver for each function. A total of eight Timver runs were required to characterize all the functional paths in each bit slice. The results of the eight analysis runs were merged into a single timing model of the bit slice. A new Verilog netlist was generated from the physical Cell3 database that included the buffers inserted by Cell3. The libraries and the new Verilog netlist were used for the initial master memory controller chip top-level timing analysis. Final top-level timing analysis used a Verilog netlist back-annotated from Cell3 and full distributed RC back-annotation for the block, instead of the Timver-generated timing libraries. This reduced the potential for error during final timing analysis.

Metal Migration Analysis and Power Bus Sizing

The RW_IO block consists of 87 bit slices, 68 of which are the RWADIOBS design. Because the RWADIOBS is the most complex macro and makes up most of the RW_IO block, it was assumed that all 87 bit slices were of this type for the metal migration analysis. The first step in the analysis was to determine the current required by each element in the RWADIOBS macro. This was done by first calculating the amount of capacitance being switched by each input of every cell and then the capacitance being switched by each output, which was the sum of the device output drivers and the wire load (back-annotated from the layout). Once this was done, the equation $I_{ave} = CVf$ was used to determine the current switching of each element. The current requirements of the elements were summed to determine the total for the RWADIOBS block. This value was used to determine the total current required by the RW_IO block and to size the power and ground buses appropriately.

Verification and Extraction

The fully placed and routed block-level database was brought into the Cadence Opus framework where a GDSII and CDL† netlist of the complete RW_IO block was generated. This data was then transferred to Mentor Graphics'* CheckMate tool where layout-versus-schematic, design rule, and electrical rule checks were performed. Checkmate was instructed to flatten the database to the standard cell level and then perform a complete distributed RC extract. This data along with the back-annotated netlist was then used for final full-chip static timing analysis.

Build and Design Data Management

The entire design process, from the construction of the bit slices to final analysis of the RW_IO block was managed by the Atria ClearCase tools. ClearCase is designed for managing large software development projects. The ClearCase tools provided a development environment with configuration management, revision control, and build management. ClearMake, ClearCase's make-compatible build tool, was used to build all the parts of the RW_IO block automatically.

† GDSII and CDL are industry-standard formats for data interchange.

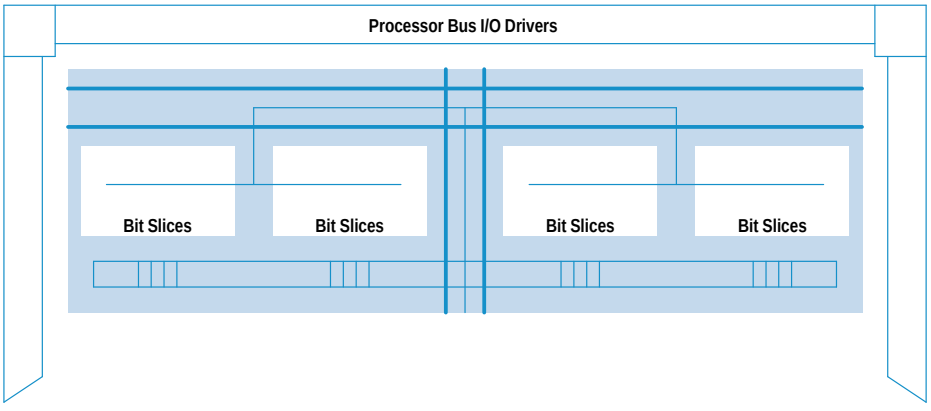

Fig. 6. First version of the RW_IO block.

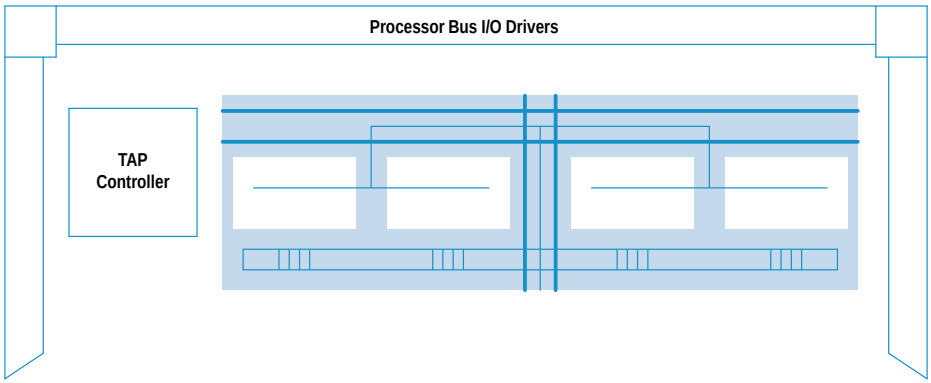

Fig. 7. RW_IO block redesigned for area conservation and inclusion of little endian functionality.

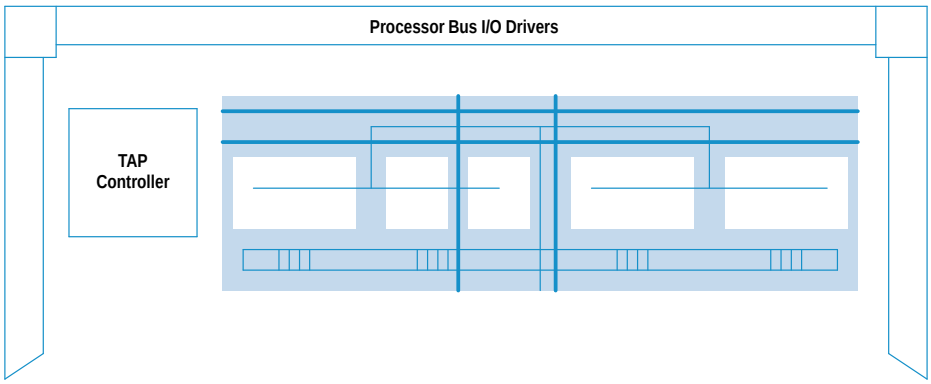

Fig. 8. RW_IO block with relocated power rails.

Makefiles were generated to optimize the overall construction process and to allow for efficient rebuilding if necessary. ClearMake's parallel distributed building capabilities greatly improved the efficiency of the build process by parallelizing the construction of the independent sections and distributing the build processes across the networked machines. During all build steps ClearMake performs auditing of the build processes. ClearMake keeps track of all the files read and written during the build and creates an audit trail called a configuration record which is kept with the built object (derived object). The configuration records are used by future builds to determine if the object is out of date and needs to be rebuilt. ClearMake examines the versions of all the elements referenced in the configuration record to determine if a rebuild is necessary. It does not rely on the "date-time-modified" of the object. The configuration records can be examined by the developer and used as a source of history information regarding any derived object.

Summary

During the course of the master memory controller chip design the RW_IO block went through many changes. The initial specification required that the block be long and narrow and cover the entire side of the chip that interfaces with the processor bus (see Fig. 6).

Later in the project, area became critical, and it was necessary to reduce the size of the block. An additional requirement on the master memory controller chip was that it had to support little endian byte order (least-significant byte is byte 0). The structure of the RW_IO block was ideally suited to implement the little endian functionality which required byte swapping the data returned from memory before delivering it to the processor bus (see Fig. 7).

The original specifications called for the main power and ground buses to pass through the center of the RW_IO block. Toward the end of the chip development the power buses changed and the V_{DD} bus had to be moved. This again required reconstructing the RW_IO block (see Fig. 8).

The use of standard cells and automatic place and route tools for the construction of the RW_IO block proved to be very advantageous. Because of the speed and flexibility of the tools the design team was not hindered when making changes to the high-speed interface block as they might have been if a full custom approach had been pursued. The most complex change to the block, which required rebuilding all of the bit slices and resizing and reconstructing the block, took only three days to complete, at which time the block was fully verified.

The design approach of using standard cells and semicustom layout techniques was favored over a data-path or full custom implementation because of its reduced risk. However, GDT schematics were generated for documentation purposes as well as a fallback position if a data-path solution had been required. The combination of standard cells, custom Cell3 layout, and accurate static timing analysis proved to be an effective solution for the RW_IO high-speed interface block.

Reference

1. S. Clayton, et al, "The Automation of Standard Cell Block Design for High-Performance Structured Custom Integrated Circuits," *Proceedings of the 1993 HP Design Technology Conference.*