RPLB Design Project Report

Michael J. Torla
Master of Computer Science Project Report
3 July 1997

64-Bit OS Group
Arizona State University

Chapter 1: Introduction 4
Chapter 2: RPLB Specification 5
Figure 1: RPLB Architecture 5
FIFO 6
Figure 2: FIFO Entry Fields 6
Read/Write Interface 7
Figure 3: Address Map 7
Command Signal Interface 8
Timing Specification 9
Figure 4: RPLB Read/Write Interface Timing (Typical) 10
Figure 5: Command Signal Interface Timing (Typical) 10
Chapter 3: RPLB Design Method 11
Modeling 11
Implementation Selection 12
Synthesis 12
Result Analysis 13
Chapter 4: Analysis of Synthesis Area and Timing Results 14
Analysis of Autologic II Synthesis 14
Analysis of Synopsys FIFO Location Synthesis 14
Figure 6: Results of Synopsys FIFO Location Synthesis Runs 14
Analysis of Synopsys FIFO Synthesis 15
Results of Static Timing Analysis 16
Analysis of Design Limitations 17
Chapter 5: Conclusion 18
Architectural Suggestions 18
Technological Suggestions 19
Future Scaling of Implementations 19
Final Thoughts 20
References 21
Appendix A: Generated Layout 22
Figure 7: Cascade Generated Layout 22
Appendix B: RPLB Synthesizable VHDL Models 23
Figure 8: VHDL Entity Hierarchy 23
Code Block 1: RPLB_CONST.VHD 24
Code Block 2: RPLB.VHD 28
Code Block 3: RWBUFF.VHD 31
Code Block 4: FIFO.VHD 34
Code Block 5: FIFO_LOCATION.VHD 41
Appendix C: RPLB Testbench 43
Code Block 1: TB_PCK.VHD 44
Code Block 2: TB.VHD 50

CHAPTER 1

Introduction

Early on in conceptualization and implementation of Sombrero, the ASU 64-bit OS Group determined that for high performance, some hardware protection mechanisms would be required. The idea conceived of was a device that could monitor memory accesses, and prevent accesses (or the result of accesses) should the location be out of range for a given permission and access type.[6]

This project is intended to prove or disprove the ability to design and implement a hardware Range Protection Lookaside Buffer as a means of protecting address space from non-permitted processes in a single address space operating system implemented on a DEC Alpha processor. In particular, the project goal is to successfully design and synthesize such a block that will be able to provide memory protection within the time constraints of the DEC Alpha pipeline, and to do so ultimately in a 500 MHz, 3.3/2.5 V, 0.35µm circuit.

The design has been described and verified in VHDL (VHSIC Hardware Description Language), using the Mentor design tools available in the ASU VLSI Design Laboratory. Synthesis, the process of converting a language description (in this case VHDL) into a schematic or netlist description, has been performed using Mentor's Autologic II synthesis tool and Synopsys Design Compiler. Finally, timing analysis was performed by the Cascade Design Automation static timing analyzer Tactic.

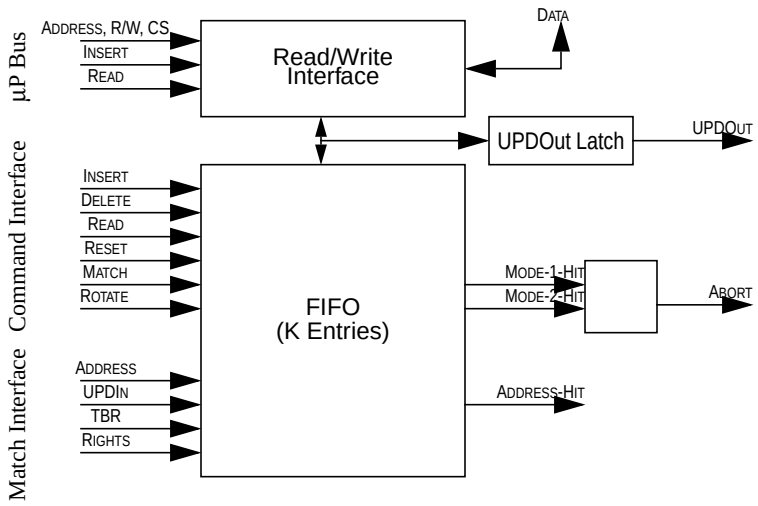
The specifications of the circuit that has been designed and synthesized are located in Chapter 2, "RPLB Specification," on page 5. Chapter 3, "RPLB Design Method," on page 11 describes the design and implementation method used. The results of synthesis, and a discussion of the timing results can be found in Chapter 4, "Analysis of Synthesis Area and Timing Results," on page 14. Final conclusions are tied together in Chapter 5, "Conclusion," on page 18.

CHAPTER 2

RPLB Specification

The Range Protection Lookaside Buffer (RPLB) is a device that is responsible for monitoring memory requests and signalling an abort to the processor upon any invalid request. There are three forms of memory request: instruction, data, and switch; the distinction will become important shortly. For the request in question, the RPLB examines the address, access rights, and value of the Protection Domain generated by the processor, and compares them to values stored in the buffer. If no memory location compares successfully to the memory request, then the RPLB signals the processor to abort the memory access. The RPLB uses a Least-Recently-Used (LRU) algorithm to sort entries. As a result, should a buffer update occur to a full RPLB, the oldest entry is removed.

FIGURE 1 RPLB Architecture



The RPLB consists of two major blocks: the read/write interface through which entries are written to the buffer, and the FIFO. Buffer entries are stored in the FIFO, which also performs the matching function, and implements the least-recently-used algorithm. Additionally, there is a latch *UPDOut* which stores the output of the FIFO when a valid domain switch occurs as described below. Finally, FIFO Outputs MODE-1-HIT and MODE-2-HIT are inputs to a NOR gate that generates ABORT.

FIFO

The FIFO consists of several words, each containing several fields. Some of the fields are multipurpose, and the use of those multipurpose fields are dependent on other fields.

FIGURE 2 FIFO Entry Fields

	Base Address	Address Mask	Protection Domain	Access Rights	Entry Type	Validity
Generic Entry						
Address / Data Entry					0	
Switch Entry		New Protection Domain			1	

Any multipurpose fields are dependent upon the *Entry Type* field. If *Entry Type* contains a 0, then the entry is used for memory-access validation; as long as one or more RPLB entries contain the address being accessed, the access is permitted to complete without an abort occurring. If *Entry Type* contains a 1, then the entry indicates a Domain Switch. Should an access to a Switch Entry occur, then the value contained in *New Protection Domain* is latched in a special RPLB buffer outside the FIFO, and is used by the host processor for further memory accesses as the new protection domain.

Whenever an entry is successfully matched (as defined below), the entry is moved to the most-recently-used position of the FIFO (the top, as it were), and every valid entry is pushed into a less-recently-used position in the FIFO, until an invalid entry is encountered.

FIFO Matching

If the *Entry Type* field contains a 0, then a match is defined as when all of the following are true:

- ADDRESS being accessed is identical to *Base Address* for those bits for which comparison is permitted by *Address Mask*
- Field *Protection Domain* matches the protection domain values available via pins TBR or UPDIN
- The access rights requested are a subset of *Access Rights*
- Command invoked is either MATCH or DELETE.

RPLB Flavors

Type 0 entries are either Address or Data entries. Some implementations of the RPLB may segregate Address entries and Data entries into separate buffers. In that case, some portions of the *Access Rights* field may be unimplemented. In particular, for an Instruction RPLB (IRPLB), the only valid access right is “execute” (and therefore rights checking is not implemented, but left to software), whereas “execute” is disallowed for Data RPLB (DRPLB) entries. In addition, an IRPLB implementation will ignore the TBR input. Switch entries may also be segregated into Switch RPLB units (SRPLB); switch entries require all rights matching, but use no *Address Mask*; the input ADDRESS must match the field *Base Address* exactly. The default or generic RPLB permits all 3 types of access.

If the *Entry Type* Field contains a 1, then a match signifies a switch from one protection domain to another. Because these are only allowed on particular addresses (not for an address range), the *New Protection Domain* field takes the place of *Address Mask*. To be specific, a Switch occurs when all of the following are true:

- The address being accessed is exactly equal to *Base Address*
- *Protection Domain* matches either UPDIN or TBR
- The access has rights that are a subset of *Access Rights*
- Command invoked is MATCH

Within an SRPLB, the switch bit is set for all entries.

Least Recently Used Algorithm

The FIFO keeps valid entries sorted via a Least Recently Used (LRU) Algorithm. When invoked, one entry (either the inserted or matching entry) is moved to the most recently-used location (top) in the FIFO. Any valid entry currently occupying that location is "pushed down" into the next less recently used location. This is applied recursively and simultaneously until either the end of the FIFO is reached, or until a non-valid entry is reached. If end of FIFO is reached, then the entry in the least recently used location (bottom) is overwritten and lost without external notification. If a non-valid entry is reached, it is overwritten, and the LRU algorithm halts.

Note that upon a FIFO Match, the current storage location of the matching entry is marked non-valid simultaneously with the movement of the matching entry to the FIFO top. That is, the storage location is marked invalid, but the valid entry is copied to the FIFO top. Therefore the most the LRU Algorithm will do is rotate among the entries between the top and the first matching entry.

Read/Write Interface

The read/write interface consists of several addressable words that can be read or written as if part of main memory by the host processor. Each major field stored in the FIFO is located at the least-significant end of the 64-bit word. The interface does not recognize address granularity finer than 64-bit-word sized. One input to the Read/Write Interface (to the RPLB, for that matter) is an chip select signal. Decoding address portions above address bit 4 fall outside the scope of the RPLB. Because upper-level addressing is handled by external logic, the RPLB can either sit in the standard address space, or off of a special address bus, as appropriate.

FIGURE 3 Address Map

Address[4:3]	Field (64 bits)
00	*Base Address*
01	*Address Mask or Protection Domain*
10	*Access Rights, Entry Type, Validity*

For example, to add an Instruction entry to the FIFO, first the processor must write the appropriate *Base Address* to 64-bit location 00. Second, it writes the appropriate *Address Mask* to location 01. The third write is to place the rights (including execute for an Instruction entry), entry (0 because this is not a switch), and the *Validity* bit into location addressed 10. Finally, to complete insertion, the RPLB must receive the INSERT command, which will move the Read/Write Interface contents to the FIFO top.

Command Signal Interface

Besides ADDRESS, DATA, R/W, CS and CLK, the RPLB has six command input wires.

1. RESET: Return all storage to initial state
2. INSERT: Transfer word in RW Buffer into FIFO
3. READ: Transfer word from FIFO to RW Buffer
4. ROTATE: Rotate contents of FIFO
5. DELETE: Delete Matching Word from FIFO
6. MATCH: Perform Match function

The operation of all commands are described in detail below.

RESET

The RESET command places not only the FIFO, but all other storage in the RPLB into its reset state. In particular, all FIFO words are marked Invalid, all fields are set to logic-0 (including the Switch flag in an SRPLB). In the Read Write Interface, all locations are set to logic-0. The UPDOut register is set to contain logic-0s.

Besides being used upon chip initialization, RESET can be asserted whenever a complete RPLB flush is required.

INSERT

The word in the RW Buffer is written into the FIFO. Care must be taken when performing an insert, for the word being inserted will be marked *valid* (and marked a *switch* if an SRPLB) regardless of the state of the contents of the RW Buffer being inserted. That is, the Read/Write Interface will assume that any word inserted into the FIFO is valid. The word will be inserted into the most-recently-used location (top) of the FIFO, and the Least Recently Used algorithm will be invoked.

The INSERT command is the means of adding entries to the FIFO.

READ

The READ command doesn't actually perform a bus-read of the RPLB, but the entry in the least-recently-used FIFO location (bottom) is copied into the RW Buffer. This occurs regardless of the state of the *Valid* bit.

By performing a READ just prior to a INSERT command, an upper-level system can monitor entry deletion.

ROTATE

The ROTATE command simply moves all entries “down” one location in the FIFO. The “bottom” entry is moved to the “top.” ROTATE occurs without regard for the state of any valid bit in the FIFO.

ROTATE is intended to be used in conjunction with READ to ascertain the state of the entire FIFO. Note that use of ROTATE, because it moves all FIFO entries, disturbs the LRU algorithm until to returned to its correct position. The entry is returned to its original position by issuing a subsequent K-1 ROTATE commands, where K is the number of entries in the FIFO.

DELETE

DELETE marks a selected entry in the FIFO invalid. The entry is selected per the description *FIFO Matching* on page 6

DELETE is the means used by the higher-level system to remove individual entries from the RPLB.

MATCH

Match causes two functions to occur. First, if no entry is selected then an ABORT is issued to the Alpha instruction stream. Second, if an entry is selected, then the Least Recently Used algorithm is invoked. The entry is selected per the description *FIFO Matching* on page 6

MATCH is the command most frequently issued to the RPLB.

Timing Specification

The actual VHDL implementation of the RPLB uses the following timing specifications. To a first order, it obeys the timing published for the Alpha 21164 Microprocessor[4].

FIGURE 4 RPLB Read/Write Interface Timing (Typical)

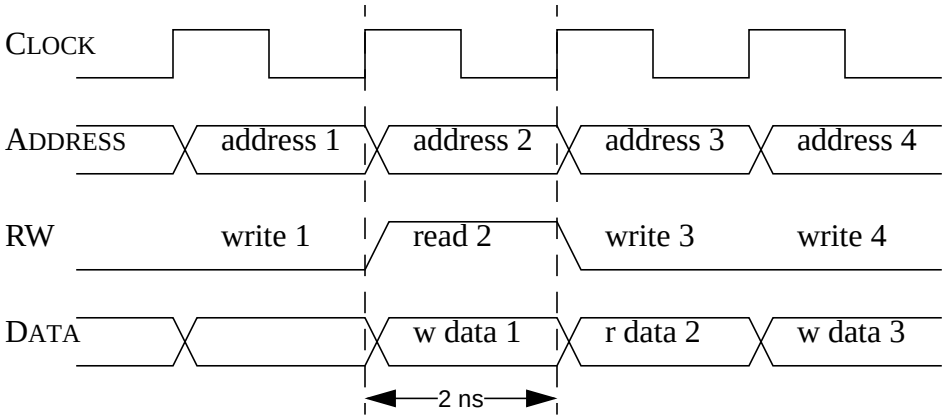
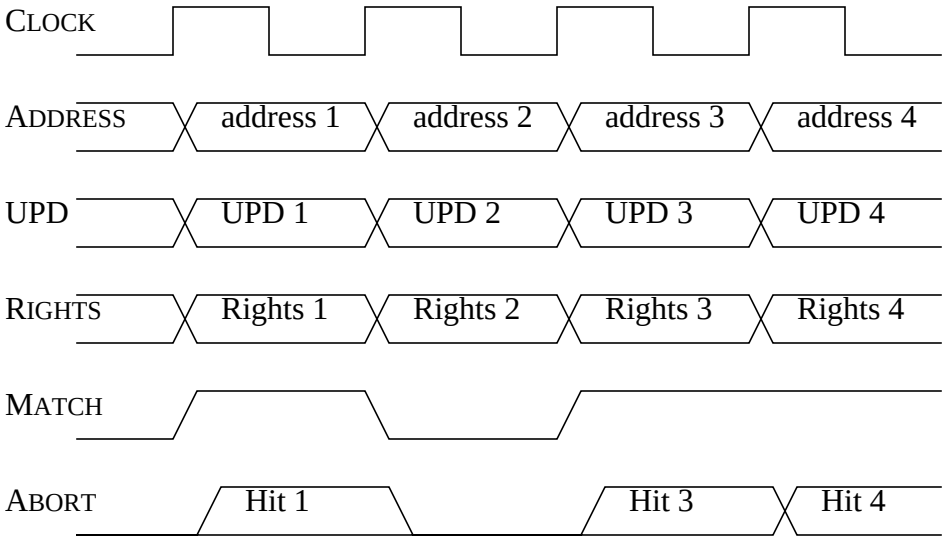


FIGURE 5 Command Signal Interface Timing (Typical)



In order to properly match the RPLB with system timing, external pipeline buffering may be required. While not accounted for nor assumed within the RPLB design, there is sufficient processor pipelining to add such buffering at both input and output sides of the RPLB without instructions completing prior to ABORT generation.

CHAPTER 3

RPLB Design Method

Proof of RPLB concept was achieved by Ganesan, Ramkarthik, and Torla in their implementation of a VHDL RPLB design for CSE 518 in fall 1995. This follow-on MCS project had two goals:

1. Update VHDL model to match latest design requirements
2. Proof of feasibility of using an RPLB structure for performing domain protection in a 500 MHz DEC Alpha system.

The original method envisioned for completing this project involved:

1. Modify original RPLB Model per new requirements, including use of iterated instances in modeling the FIFO. The original FIFO model was built like a state machine; this implementation instantiated N FIFO-words for some generic N.
2. Define and implement particular style and size of RPLB for proof-of-feasibility.
3. Synthesize VHDL model using the new Autologic II tool.
4. Analyze results and compare to requirements, taking into account library discongruities. This was to include static timing analysis and back-annotated functional simulation.

Modeling

There were actually relatively few changes to the RPLB concept between Fall 1995 and Winter 1997. The original model included a feature entitled “Global ATO” which involved complicating the Switch RPLB functionality. The concepts of special purpose RPLB implementations (IRPLB, DRPLB, SRPLB) had not been documented, if even thought of. As a result, some time went into removing now obsolete “Global ATO” functionality. In addition, VHDL Generics IRPLB, DRPLB, and SRPLB were added to existing constants N (address width), U (UPD width), and K (number of entries in the FIFO) to control design model targeting. Also, for consistency, N, U, and K were remodeled to use generics. As a result, a N=48, U=48, K=8 IRPLB and a N=16, U=5, K=8 generic RPLB could be generated from the same model merely by specifying different values for the generics either in Mentor Tool runtime switches, or through the use of configurations.

The original test bench was modified to as a result of the removal of “Global ATO” functionality. The test bench only runs on an N=16, U=5, K=8 generic RPLB.

The other change performed was to remodel the FIFO. Originally, the FIFO was modeled using a state-machine style; all words of the FIFO were implemented as part of the state of one very large state machine. For N=16, U=5, K=8, this resulted in 336 bits of state. While this permitted quick generation of a synthesizable model, the model became unwieldy for the synthesis tool (then Autologic) to operate on. After two weeks of synthesis time on a Sparc 20, Autologic gave no sign of being able to synthesize any of the designs attempted that were larger than the above.

Autologic II, released within the ASU environment after the original CSE 518 project was completed, included new features permitting the instantiation of multiple components using the “For Generate” statement. That is, while Autologic only permitted instantiation of single components manually, Autologic II allowed the designer to write code that would instantiate a “variable” number of components, where that number would be specified at compile time.

As a result, the bulk of the remodelling effort went into

1. Writing a model of a “FIFO Location.” A FIFO Location is the logic that implements the entry storage and the matching function associated with a particular entry
2. Writing a new architecture of FIFO that instantiated K copies of FIFO Location and tied together the hit/miss outputs, as well as completed the logic necessary for the Least Recently Used (LRU) algorithm.

This new model was then simulated against the existing test bench. The test bench required one modification; for ease in modeling, the method of invalidation changed between the two different FIFO models. The test bench was modified to take this into account.

Implementation Selection

Following successful simulation of the new RPLB model, an implementation was selected for synthesis and performance evaluation, based on the 21164 specification, and with consultation of the ASU 64-bit OS Group. In order to fit into the existing 21164, 48 bits of Address and 48 bits of UPD were required. The Sombrero team indicated that the highest performance style of RPLB would likely need to be the Instruction flavor, requiring constant single-cycle RPLB monitoring of memory accesses. However, the Instruction flavor of RPLB, while limiting the number of entries, would also implement a relatively small subset of the full match functionality. Therefore, in order to best meld the size advantages of the Instruction flavored RPLB with the general requirements, it was decided to implement an 8-entry generic-style RPLB.

Synthesis

Originally, it was intended that all design work would be completed within the Mentor Tool set available in the ASU VLSI Design Lab. The original method was abandoned due to limitations of the installed version of Autologic II. The synthesis tool was unable to perform a timing analysis on the RPLB FIFO, for reasons still unknown. This pre-

vented performing not only timing optimization, but also prevented generating a timing report.

Therefore, it was decide to use Autologic II to convert the design to a Verilog description, so the designer could synthesize the RPLB using Synopsys Design Compiler. An added advantage to this method was the availability of a 0.35µ library, simplifying final analysis.

Because Autologic II was used as a translator, the design constraints applied to Autologic II were minimal. However, in Synopsys DC, the following constraints were applied:

1. CLK: 2 ns period
2. Output Delay: minimum 0, maximum 1.5ns following rise of CLK.
3. Input Setup: 1/2 clk period (0.5 ns) prior to rise of CLK.
4. Input Hold: 1/20 clk period (0.05 ns) following rise of CLK.

Numerous Synthesis runs were completed upon the FIFO Location portion of the design in order to determine the best synthesis “recipe.” Then, synthesis of the FIFO was completed, using the recipe determined by FIFO Location synthesis.

Result Analysis

Originally, static timing analysis was to be performed within the Mentor Tool set using Quickpath. However, abandonment of original synthesis flow also required abandonment of Quickpath. This function was replaced with the use of the Tactic static timing analyzer available within the Cascade Design Automation “Epoch” Tool set. The primary advantage to the use of Tactic was the availability of a 0.35µ library, simplifying the process of result analysis. Originally, some method of comparing the chosen Mentor library to the Alpha technology would have been required. However, because of the similarities in process, it was assumed that the final synthesis library represented silicon capable of similar performance to that used by Digital Semiconductor in fabrication of their 500 MHz Alpha Processor.

Functional back-annotated simulation was also originally intended. However, because the available licenses of the Synopsys and Epoch tools available do not permit VHDL usage, and because the test bench was generated in VHDL, and because no adequate VHDL to Verilog translation tool was available, this effort had to be abandoned.

Prior to Verilog translation, it was verified that Autologic II would, under similar circumstances, generate a VHDL netlist that simulated correctly against the test bench.

CHAPTER 4

Analysis of Synthesis Area and Timing Results

Analysis of Autologic II Synthesis

Autologic II was able to synthesize a single FIFO Location. It was, however, unable to synthesize the entire FIFO with timing. Autologic II reported the presence of timing loops when performing timing analysis, preventing it from performing timing analysis. When the REP NEIGHBORS -LOOP command was applied in order to identify timing loops, it reported none. Unable to perform timing analysis, Autologic II was deemed unsuitable for generating the design required.

By disabling the timing analysis portion of the synthesis tool, Autologic II was able to generate correct FIFO netlists in VHDL (verified by FIFO Location simulation), and syntactically correct Verilog netlists (verified only by Synopsys synthesis).

Autologic II generated two different models of FIFO Location in building the Verilog model of FIFO. These two models, as well as a model resulting from stand-alone synthesis of FIFO Location were all cross-compared using the Synopsys compare command. All three models of FIFO Location were functionally equivalent.

Analysis of Synopsys FIFO Location Synthesis

Given the constraints specified previously, FIFO Location was synthesized using 18 different recipes. The results are summarized below. The timing result column of the table indicates the amount of time that the reported worst-case path fails by, according to Synopsys Timing Analysis.

FIGURE 6 Results of Synopsys FIFO Location Synthesis Runs

Flatten Options	Structure Options	Compile Options	Worst Path Timing Violation	Area Reported (μm^2)
low, single-output, no-phase	off	medium-effort	4.5 ns	195072
medium, single-output, phase	off	medium-effort	4.4 ns	194180
medium, multiple-out, no-phase	off	medium-effort	4.5 ns	195072
medium, multiple-output, phase	off	medium-effort	4.4 ns	194180
off	no-boolean, no-timing	medium-effort	2.4 ns	158762
off	boolean, no-timing	medium-effort	2.4 ns	158762

FIGURE 6 Results of Synopsys FIFO Location Synthesis Runs

Flatten Options	Structure Options	Compile Options	Worst Path Timing Violation	Area Reported (μm^2)
off	no-boolean, timing	medium-effort	2.4 ns	158156
off	boolean, timing	medium-effort	2.4 ns	158156
medium, single-output, phase	no-boolean, timing	medium-effort	2.4 ns	158665
low, single-output, no-phase	off	high-effort	4.5 ns	195072
medium, single-output, phase	off	high-effort	4.4 ns	194180
medium, multiple-out, no-phase	off	high-effort	4.5 ns	195072
medium, multiple-output, phase	off	high-effort	4.4 ns	194180
off	no-boolean, no-timing	high-effort	2.4 ns	159103
off	boolean, no-timing	high-effort	2.4 ns	159103
off	no-boolean, timing	high-effort	2.3 ns	158145
off	boolean, timing	high-effort	2.3 ns	158145
medium, single-output, phase	no-boolean, timing	high-effort	2.2 ns	159355

Additional Synopsys synthesis runs were attempted, examples include trying to encourage use of otherwise restricted scan cells, or encouraging the removing of hierarchy prior to synthesis. None of these attempts ever improved upon the synthesis run reported at the bottom of figure 6.

Failing Timing Paths, Results of Best Recipe

Most of the failing paths either involve the *valid* bit within FIFO Location, or they are on the path from TBR to one of the HIT pins. One additional failing path was reported from UPDIN to MODE-1-HIT. All *valid*-associated failures violated timing by more than 2 ns; the others violated by roughly 1.5 ns.

Analysis of Synopsys FIFO Synthesis

Methods learned from synthesis of one FIFO Location were applied to synthesis of the FIFO as a whole. Several different hierarchical approaches were applied:

1. including plain instantiation of the FIFO Location cell 8 times (with dont_touch set, disallowing FIFO synthesis from performing *any* changes to FIFO Location)
2. Synthesizing with boundary optimization (encouraging Synopsys to make minor changes) and uniquification (allowing Synopsys to tailor individual FIFO Location modules to particular circumstances)
3. Synthesizing with hierarchy flattening (removing all hierarchy and allowing Synopsys to operate on the design as one flat netlist).

While synthesis of FIFO Location with the most complicated synthesis options always took on the order of several hours, actual much simpler synthesis runs of the much larger FIFO took days, limiting the number of runs permitted.

Synopsys produced the best results by synthesizing using uniquification and minor boundary optimizations.

FIFO Area Results

The area reported for all synthesis runs of the entire FIFO is less than the sum of the area required for 8 FIFO Location blocks. This is particularly noteworthy because of the combinatorial logic implemented in the FIFO block for support of the Least-Recently-Used Algorithm. The worst FIFO area observed was approximately 1.27mm^2. The Alpha, manufactured to a 0.35mm process, is reported to require 208mm^2 [1]; the synthesized RPLB would be a very small addition to an Alpha chip. The results of place and route indicated that for the entire FIFO, Synopsys was wrong in its area estimates. For the best-performing synthesis of the FIFO, area reported was approximately 1.1 mm^2. For the same netlist, Cascade Place and Route generated a cell with area of approximately 2.5 mm^2.

FIFO Timing Results

The synthesis runs indicated worst-case paths failing by more than twice any FIFO Location synthesis run. The worst path in the best synthesis run ran from FIFO Location instantiation 4 to FIFO_BUS, the common bus in the design. It failed timing by just over 8 ns. This failure is consistent with the logic required to implement the LRU algorithm. In other words, despite the ability to update flags in approximately 4 ns, the propagation delay associated with determining which entries are moved (and moving them) requires much more time than the resolution of one FIFO Location. The movement of entries is the cost of using the LRU Algorithm.

Results of Static Timing Analysis

After using Cascade Design Automation's Epoch tools to generate an initial timing-driven foreplay and layout, their Tactic Static Timing Analyzer was used to perform final timing analysis. Tactic reported worst case paths in FIFO Location failing by roughly 1.2 ns rather than the 2.4 ns reported by Synopsys. Tactic also failed to report Synopsys' worst case path; forcing Tactic to analyze that particular path indicated a similar reduction in timing violations.

Upon performing timing analysis of the entire FIFO, Tactic reported similar results as that to Synopsys, if on different paths. In particular, for the best synthesis run, Synopsys reported it's worst path to be from the CLK pin of flip-flop 35 of FIFO Location 4 to the common FIFO_BUS, missing timing by 8.053 ns. Tactic reported this path to require 7.109 ns, but only reported this path when asked explicitly. Tactic reported it's worst path to be from UPDIN[10] to the TEST pin of flip-flop 149 (*valid* flag) of FIFO Location 8, requiring 8.751 ns. Synopsys reported this to require 8.338 ns (failing by 5.525 ns), but only reported this path when asked explicitly.

While actual layout did not permit the design to work at speed, the improvements indicate that the Synopsys wire load model, which assumes capacitance on every net based

on fanout and average wire length for a design of a particular cell area, was pessimistic for this particular design.

Analysis of Design Limitations

It should be noted that the design project was essentially attempting, to use an analogy, to build a world-class racing bicycle using heavy-duty mountain bike components. There are three significant reasons why the RPLB as designed here was handicapped relative to the Alpha.

1. Process technology: the design was synthesized to a 0.35 µm 3-layer metal library produced by a semiconductor manufacturing company claiming 57 ps typical gate delay in that process. AT&T Microelectronics claims a 52 ps typical gate delay for their 0.35 µm process.[7] No published claims for the speed of the DEC Alpha process were found, however it would be reasonable to assume their process is marginally faster than even the AT&T process. It is known from published works[1] that the Alpha process included an additional layer of metal to that permitted in the Synopsys library.
2. Library limitations: works published by Digital Semiconductor [1] indicate the use of advanced dynamic libraries for critical portions of the Alpha design. The library used by Synopsys was a typical ASIC/Standard Cell library. Because of the wide audience using such libraries, these elements are typically designed more conservatively than full custom libraries used by a select design team.
3. Methodological limitations: works published by Digital Semiconductor [1, 2] indicate the use of full-custom design techniques, including the use of “extensive RC delay simulation” and “layout-extracted resistance and capacitance data”. These techniques are far beyond the sophistication or the scope of any logic synthesis tool.

Cited works published by Digital Semiconductor indicate some use of logic synthesis tools in development of the Alpha 21164. However, use of logic synthesis was reserved for blocks of random logic, and was carefully analyzed and modified by engineers as required. [3] Furthermore, this same reference discusses the development of an issue scoreboard, requiring some of the same comparison functions required in the RPLB. The issue scoreboard that was produced included such advanced techniques as the use of domino logic and the implementation of AND functions using transmission gates.

Digital Semiconductor’s circuit design and layout techniques are far beyond the scope of this project, so naturally they would produce results superior to those produced here.

CHAPTER 5

Conclusion

To put it simply, synthesis failed. More specifically, it failed to produce, using a conservative design technique in a similar process, a Range Protection Lookaside Buffer that could meet the timing requirements of a state-of-the-art 500 MHz Alpha Microprocessor. This should not be a surprise.

What is encouraging is how close the design came to working. Timing analysis indicates that the design, in its worst incarnation, would operate at 120 MHz. The FIFO Location itself would operate at 250 MHz. These results are encouraging enough to warrant future investigation.

Architectural Suggestions

The design is expected, upon a hit, to produce the appropriate signalling within one clock cycle of receiving data. Upon a hit, it is furthermore expected to rearrange the entries stored such that the entry hit is moved up to the *top* of the FIFO from wherever in the FIFO it sat within one clock cycle. The clock cycle in this design is 2 ns. This timing requirement of one clock cycle is somewhat arbitrary, yet a requirement of the use of the Least-Recently-Used (LRU) Algorithm.

Several improvements could result from switching to a Not-Last-Used (NLU) algorithm. First and foremost, the interaction between individual FIFO Location instances would cease, as all that is required to be done on a hit is to update a Last-Used-Entry flag rather than moving several entries around in the FIFO. As a result, the only timing paths would be within a single FIFO Location. Because the LRU Algorithm requires hits to occur immediately after data setup, there is no sense in minimizing matching logic paths at the expense of entry logic paths. However, without the LRU Algorithm, RPLB speed could be marginally improved by forcing the match function to be integrated with the stored entry as close to the ABORT pin as possible. Such an approach might increase the RPLB's operating speed by 10%.

In addition, removal of the LRU algorithm would permit pipelining within the RPLB; it would not be necessary for the entire state of all entries to be known at the end of every clock cycle, it would be sufficient to allow a hit to mark an entry *not last used* in a pipelined fashion. By inserting a stage of flip-flops at the latest possible point in the logic tree generating HIT, the design could be forced to meet timing. However, this would require the pipelining to be allowed to flush prior to entry deletion, and the entry deletion

algorithm would become more complex. This would likely allow the RPLB to operate at 500 MHz, perhaps even an RPLB synthesized in the same fashion as the way this experiment proceeded.

Such pipelining would require further study of the Alpha execution pipelines; it is believed currently that a multi-cycle RPLB would produce ABORT signals with sufficient speed to operate properly.

It should be noted that the Alpha Instruction and Data Translation buffers use the less active Not-Last-Used algorithm.

The synthesis portion of the project synthesized a Generic-flavor RPLB for a 48 bit address and 48-bit protection domain. Yet discussions amongst the 64-bit OS Group about the target system indicate the desirability for special-purpose RPLB implementations; that is, the other 3 flavors mentioned. All other RPLB flavors offer simplifications over the Generic RPLB. In addition, assuming that some minimum size block larger than the byte is the minimum granularity permitted for domains, one could reduce the number of address pins. Such an improvement is expected to help, but not solve the timing problems observed.

Technological Suggestions

Using full custom design techniques, the resultant RPLB would be much faster. Semicustom approaches, like used in this RPLB design project, have never been suitable for producing leading-edge performance circuitry. Clearly the results of this project indicate that semicustom techniques are able to produce circuitry that only several years ago was beyond the reach of full-custom techniques, but this improvement is mostly due to silicon fabrication improvements, which would scale the speed of full-custom and semicustom circuitry equally. A full-custom design style is beyond the resources of all except the most committed (and well funded) silicon design houses. Such obvious improvements available to full-custom techniques include precharged wire-nor comparison. In addition, the Alpha 21164 design team used library elements such as level-sensitive transmission gate latches, with an estimated speed improvement of 10 percent over their own Alpha 21064 design. [3]

Although not known for sure, it is hypothesized that the Digital Semiconductor process used to fabricate the 500 MHz Alpha microprocessor is superior to the process described in the Synopsys synthesis library. To a first order, they are equivalent; however, the first order is just the drawn polysilicon width. It is known that the Digital process offers an additional layer of metal interconnect. It is also known that there are ASIC vendors claiming somewhat faster gate delays than claimed for the library used in this synthesis project. Some performance improvement could be gained by using a library for the DEC Alpha process.

Future Scaling of Implementations

Current 64-bit processors actually implement fewer than 64 bits of address; that is why the RPLB was only synthesized with 48 bits of ADDRESS. It is envisioned that address

buses will continue to grow in width. Based on early synthesis work in Autologic II, it is not believed that the width of the ADDRESS bus presents much of a speed impact to the RPLB. Therefore, these results scale to larger implementations. Certainly, additional ADDRESS bits will require additional logic, that logic is likely to somewhat decrease RPLB performance. Such performance decreases are expected to be minor.

Final Thoughts

It is unsettling to spend several months working on a project that because of scope looks like a failure, yet yields such promising results. A working RPLB design was not produced. Yet the lessons learned indicate that a high performance Range Protection Lookaside Buffer is within the grasp of capable full-custom designers. While the ability to produce a full-speed RPLB is not guaranteed, these results indicate that such a design is likely enough to warrant further study using more advanced techniques.

References

[1] Gronowski, P. E., Bannon, P. J., Bertone, M. S., Blake-Campos, R. P., Bouchard, G. A., Bowhill, W. J., Carlson, D. A., Castelino, R. W., Donchin, D. R., Fromm, R. M., Gowan, M. K., Jain, A. K., Loughlin, B. J., Mehta, S., Meyer, J. E., Mueller, R. O., Olesin, A., Pham, T. N., Preston, R. P., and Rubinfeld, P. I. “A 433 MHz 64b Quad-Issue RISC Microprocessor”, *ISSCC Digest of Technical Papers*, 1996, pages 222-223.

[2] Bowhill, W. J., Allmon, R. L., Bell, S. L., Cooper, E. M., Donchin, D. R., Edmonson, J. H., Fischer, T. C., Gronowski, P. E., Jain, A., K., Kroesen, P. L., Loughlin, B. J., Preston, R. P., Rubinfeld, P. I., Smith, M. J., Thierauf, S. C., Wolrich, G. M. “A 300 MHz 64b Quad-Issue CMOS RISC Microprocessor”, *ISSCC Digest of Technical Papers*, 1995, pages 182-183.

[3] Bowhill, W. J., Bell, S. L., Benschneider, B. J., Black, A. J., Britton, S. M., Castelino, R. W., Donchin, D. R., Edmonson, J. H., Fair, H. R., Gronowski, P. E., Jain, A., K., Kroesen, P. L., Lamere, P. E., Loughlin, B. J., Mehta, S., Preston, R. P., Santhanam, S., Shedd, T. A., Smith, M. J., Thierauf, S. C. “Circuit Implementation of a 300-MHz 64-bit Second-generation CMOS Alpha CPU”, Digital Technical Journal, Volume 7, Number 1, Special Edition 1995.

[4] Digital Semiconductor 21164 (366 MHz through 433 MHz) Alpha Microprocessor Hardware Reference Manual, Digital Equipment Corporation, March 1996.

[5] Ganesan, R., Sukavanam, V., and Torla, M., “Range Protection Lookaside Buffer, VHDL Synthesis Report”, Term Report CSE 518 VLSI Synthesis Using HDLs, December 1995.

[6] Skousen, A. C., and Miller, D. S., “Resource Access and Protection in Sombrero: Protection Model, Software Protection Data Structures and Hardware Range Protection Lookaside Buffer”, *ASU 64-bit OS Group WP 2*, Computer Science and Engineering Department, Arizona State University, TR-95-013, May 1996.

[7] “AT&T Microelectronics introduces 0.35 micron system-ASIC products”, AT&T News Release, http://www.att.com/press/1195/951102.mea.html, 2 November 1995.