Design Techniques for High-Speed Datapaths[1].

Bertrand S. Irissou

University of California at Berkeley
Department of Electrical Engineering and Computer Sciences
Computer Science Division

November 1992

Abstract

This report describes our research on the performance limits of datapaths in MOS technology. By using a combination of single-phase clocking, dynamic logic circuits, limited pipelining and custom layout, we achieve high-speed operation of the datapath and a tremendous performance increase over traditional implementations that use static CMOS circuits and multi-phase clocking. To demonstrate these techniques we have built a 64-bit integer datapath comprising an adder, a three-ported register file and a PLA. The datapath was fabricated in the HP CMOS34 1.2μm process. It has been tested and is fully functional at 180MHz.

[1] This work was funded by the National Science Foundation Infrastructure Grant no. CDA-8722788. Fabrication of the chip was supported in part by a grant from DARPA

Contents

List of Figures

Chapter 1

Project Goals

The last decade has seen the microprocessor become the cornerstone of the computer. While innovative architectures and more advanced fabrication processes account for most of the performance increase obtained during that period, the circuit techniques employed have remained quite conservative. The goal of this project is to understand the practical performance limitations of CMOS circuits and the techniques for achieving high-speed operation for microprocessor datapaths.

1.1 Motivations

Each generation of processors surpasses the previous one in complexity, speed and performance. In that regard, microprocessor design is a technology driver in the same way that dynamic RAM design drives silicon processing technology. Assessing the impact of a specific technique on the performance of a processor is difficult. However, most of the sources of performance improvement fall into two categories: advanced fabrication processes and architectural changes.

The speed of a circuit comes from its ability to quickly charge and discharge electrical nodes. Improved fabrication technology has shrunk the size of devices at a rate of 30% [HP90] every three years. As feature size decreases, the electrical strength of the transistors increases and the local interconnect capacitance decreases. The combination of these two factors has contributed the most to circuit speed increases.

A giant leap was made in the late 1970s when the density of processing technology allowed for the first time the integration of a processor on a single chip. By eliminating most of the off-chip interconnect capacitance, the microprocessor was born and the clock frequency of the system increased significantly from a few megahertz to over 100MHz today.

Architectural changes have also considerably increased the performance of microprocessors. With the first microprocessors, the limited device density only allowed a few functional units on a single chip. Early microprocessors were based on Complex Instruction Set Computer (CISC) designs that maximize the reuse of the functional units for every instruction. Thus, each instruction would typically require 4 to 10 cycles to complete. As the level of integration increased, the number of functional units on a single chip increased and the instruction set evolved to what is known as Reduced Instruction Set Architecture (RISC). RISC implementations assign one functional unit per micro-operation of the instruction. By pipelining the execution of the instruction, the number of cycles per instructions (CPI) was decreased to a little more than one, increasing the performance of the processor. An important side effect of the simplification of the instruction set introduced by RISC architecture is that it also simplified the implementation. This resulted in simpler circuitry with lower clock period. However the data dependency of the instruction stream somewhat limits the usefulness of pipelining. The performance increase of extra pipe stages is quickly offset by the data hazards introduced in the pipe. As such, very deep pipelines are inefficient for datapath implementations where a strong data dependency from one instruction to the next prevails. Although multi-threaded processors are thought to alleviate this problem by interleaving several instruction streams, they still remain a topic for research and are not yet mainstream. More recently, as density increased even further, superscalar architectures have tried to take advantage of the instruction level parallelism of code by executing several instructions in parallel every cycle. This has brought the CPI below one, resulting in even higher processor performance.

However, circuit technology has seen little change since the early days of microprocessor design. As complementary MOS processes became available in the early 80's, a shift from NMOS logic to CMOS logic considerably lowered the power consumption of circuits. Circuits utilizing complementary devices have the advantage of consuming power only when switching. By comparison, NMOS circuits need an additional biasing current that generates a large static power consumption and limits the circuit speed. Also, CMOS circuit designs have been popular for their robustness to noise and process variation. Unfortunately, CMOS circuits trade design safety for speed. In high-speed design, Emitter Coupled Logic has usually been preferred over CMOS logic. But the high power consumption and the lower integration density of bipolar transistors has made ECL unsuitable for single chip implementation of processors. A recent effort to merge bipolar and CMOS technology in a BiCMOS process was thought to offer a good compromise by combining the zero static power consumption of CMOS circuitry with the high switching strength of the bipolar transistor. Unfortunately, the larger number of processing steps involved in the fabrication of BiCMOS circuits makes the process less reliable than a CMOS process. Most microprocessor manufacturers have stayed away from BiCMOS implementations because of the much lower yields. The use of more advanced processes (BiCMOS, GaAs, etc...) for microprocessor design is unlikely in a near future given the economic constraints. We therefore have focused on improving the circuitry.

Clocking methodology is another aspect of microprocessor design that has seen little improvement. Most designs utilize a multi-phase clock distribution scheme. This approach is preferred because faults caused by clock skew can be eliminated by controlling the non-overlap time of the phases. However, multi-phase clocking is ill-suited to high-speed design because the non-overlap time between phases becomes an increasingly important part of the clock cycle that cannot be utilized.

We, and others independently [Dob92], have found that more aggressive circuit techniques and better clocking methodology can significantly increase performance.

1.2 Circuit Considerations

VLSI circuits can be characterized in three ways [Bak90]: the circuit family used, the clocking methodology and the storage element used. The circuit family of choice for microprocessor design has been, without question, complementary MOS gates. Industry circuit designers commonly use complementary CMOS gates and multi-phase clocking with level-sensitive latches, or "TTL style" edge-triggered clocking. These techniques are popular for their robustness to noise and process variations, but are slow compared to dynamic logic. Although dynamic implementations of logic functions require fewer transistors than their fully static counterpart, noise margins are reduced and so they are often avoided. Also, designing with dynamic logic is usually more involved than with static CMOS because special attention must be paid to noise immunity and charge sharing. Still dynamic logic is used when speed and density are of utmost importance. Examples of such structures include the domino and NORA circuit families [GD85].

The clock distribution methodology also plays an important role in design. Because they are virtually guaranteed to be race-free, most VLSI systems adopt a multiple phase clocking methodology. Two-phase designs are the most popular, although three and four phases designs have also been done. A difficulty with clock distribution in multiple phase systems is minimizing the clock skew between phases. Several phases of the same clock are distributed over the entire die and special precautions must be taken to guarantee that no phase arrives early or late on any clock node. Techniques such as load matching at the different nodes of the clock distribution tree and careful layout of the distribution tree are used to minimize skew. Unfortunately, the lack of good static timing analysis tools makes this job quite difficult and tedious, and it is still a source of problems in microprocessor design.

Finally, the storage element used is critical to a design. Even though every project seems to spend a lot of time redesigning the latches and flip-flops to be used, the designs almost inevitably settle on fully static multiple phase or edge-triggered solutions. These conservative approaches are motivated by test necessities such as single stepping of the processor for debugging. Also, boundary and internal scan techniques require most latches to be static in order to scan in and out the state of a processor. In addition, because static latches are more immune to noise, they have been preferred to faster but more noise sensitive designs. A problem with fully static latches is that their propagation time is usually quite large making them undesirable in high-speed design.

1.3 Circuit Alternatives

In an attempt to address these issues, we have found that abandoning these traditional techniques in favor of a somewhat "riskier" design style can lead to significant performance improvements. These techniques include

- Dynamic Logic. Considerable speedup can be obtained by the systematic use of precharged logic. Gate delays are significantly decreased because of the lower switching threshold voltage of the dynamic nodes and the lower capacitance on the nodes.
- Single Phase Clocking. First presented by Svensson et al. [YS89, AS90] , it relies on both the high and the low phase of a single clock to achieve the same effect as two-phase clocking.
- Fast latches. Simple dynamic single phase latches are used to reduce the delay through the latch and the setup and hold time.
- Custom Layout. The combination of dynamic nodes and high-speed switching make capacitive coupling issues a design concern. Custom layout techniques help to limit the effect of coupling and also minimize the parasitic capacitance.

In the next chapter, we will discuss in detail the advantages and limitations of these techniques. Then, we will present a 250MHz implementation of a 64-bit integer datapath using these techniques, addressing the design issues involved with high-speed circuitry.

Chapter 2

Techniques for High-speed CMOS

The factors under the control of circuit level designers that limit the speed of circuits include overhead due to clock distribution and latches, gate delay, capacitive loading due to buses, and the parasitics associated with transistor drains. This chapter describes the techniques we use to overcome these limitations and achieve high-speed operation. They fall into three categories: reduced voltage swing logic, single phase clocking and custom layout.

2.1 Circuit Techniques

2.1.1 CMOS speed limitations

CMOS circuits have been preferred by designers for their robustness to noise and their ease of design. Unfortunately noise immunity is traded for speed. A CMOS gate is composed of two complementary networks: a pull-up network made of p-type transistors and a pull-down network made of n-type transistors. It is the simultaneous presence of the two networks that gives the CMOS gate its noise immunity, since a gate output can switch either high or low. The redundancy of the two complementary networks results in a large number of transistors. Also, substantial sizing of the pull-up network is required to offset the lower mobility of electrons in p-type transistors and match the rise time and fall time on the output, causing more loading on the output.

A crude but useful approximation of the delay through a gate,T_d , is obtained by modeling it as the combination of first a intrinsic fixed delay, T_g, proportional to the size of the transistor of the gate, the logic function of the gate and its switching threshold and secondly, a delay proportional to the capacitive load on the output of the gate, C_{load}, and the driving strength of the gate α:

$$T_d = T_g + \alpha C_{load}$$

For a CMOS gate, the logic threshold is typically set at $\frac{V_{dd}}{2}$.

Instead of using CMOS logic, we speed up the circuit logic by systematically using dynamic or precharged logic in our design. Dynamic logic helps decrease the gate delay by having a lower switching threshold than CMOS gates and having a smaller capacitive load on their inputs.

2.1.2 Dynamic logic

Dynamic logic gates eliminate the redundancy of the complementary pull-up and pull-down networks of CMOS gates. Two types of dynamic logic gates exist: p-type and n-type. In a n-type gate, the pull-up network can be replaced by a PMOS precharge transistor. By default, the gate is charged high and the pull-down network only needs to discharge the output when necessary. Alternatively in an p-type gate, the pull-down network is replaced by an NMOS precharge transistor. Figure 2.1 (C) shows an example of two cascaded precharged n-type inverter and p-type inverter stages.

By using dynamic logic we reduce the switching level of the gate. For an n-type gate, the output of the gate starts changing as soon as the input is one threshold voltage, V_{tn}, above ground. For a p-type

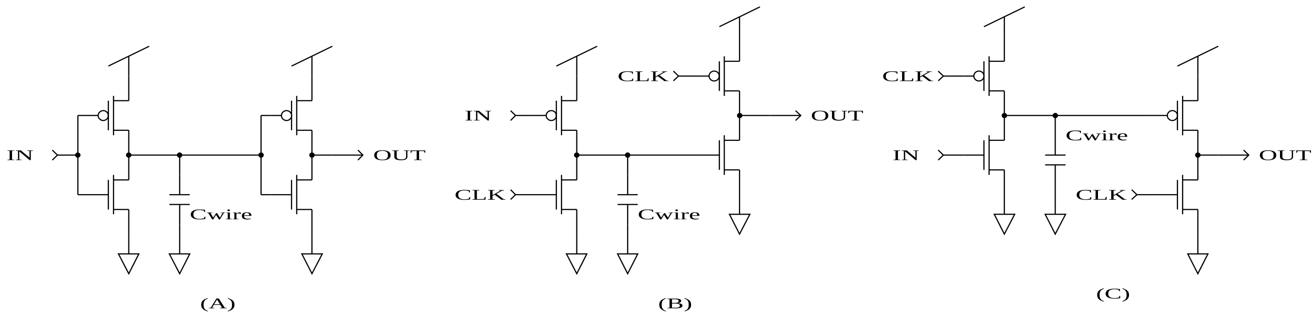


Figure 2.1: Comparison of (A) a pair of CMOS inverters, (B) a n-p precharged inverters and (C) a p-n precharged inverters (C)

gate, the output starts switching as soon as the input falls one threshold voltage V_{tp} below V_{dd}. This helps decrease the intrinsic gate delay T_g. The noise margins of the gate are also reduced because the output of the gate can only switch once after it has precharged. Therefore, careful layout to minimize noise from other signals is required.

Another source of speedup is the lower input capacitance of the gate. Because only the pull-up or the pull-down network is present, the input capacitance of a gate is on average one half the input capacitance of a CMOS gate. This is especially important when fanning out to bigger gates where the αC_{load} term dominates the gate delay.

However, using dynamic logic presents the inconvenience of having to precharge the gates. We solve that problem at the system level by breaking the logic in alternating *n-blocks* and *p-blocks* separated by corresponding n-latches and p-latches. A *p-block* is a set of alternating n-type and p-type gates that precharge on the high phase of the clock and evaluate during the low phase. The block is terminated by a p-latch, transparent during the low phase of the clock and closed during the high phase. Similarly, an *n-block* precharges on the low phase of the clock and evaluates during the high phase. This approach imposes a constraint that systems must be made from an even number of blocks if there is to be any feedback. An example of a such system is shown in the pipeline of Figure 2.2.

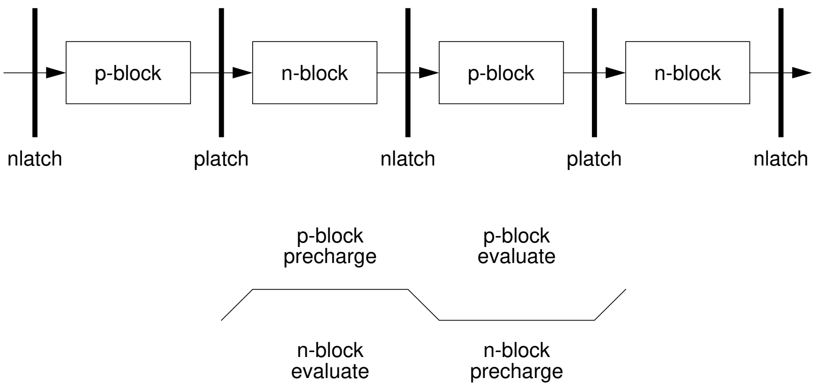


Figure 2.2: Timing of a Precharged Logic Based Pipeline

2.1.3 Reduced Voltage Swing Sense Amplifier

Precharged logic can also speed up the recovery of slowly changing signals from highly capacitive buses or nodes. With a switching threshold of V_{tn} above the ground or V_{tp} below the power rail, they offer an alternative to more complicated sense amplifier designs.

For a bus receiver we use cascaded precharged inverters. In the case of a bus that precharges high, we use an inverter that precharges low. Both the bus and the receiver precharge on the same phase of the clock. On the next phase, the bus driver either begins to pull the bus low or allows it to float high. If the bus is pulled low, the PMOS transistor of the precharged inverter detects the voltage as it falls below the PMOS transistor threshold voltage V_{tp}, allowing the inverter to switch. Because the PMOS transistor operates near its threshold regime (as opposed to being in saturation when used in normal precharged gates), we oversize the transistor to increase its conductance in that region. Since the bus is a already a highly capacitive node, the extra gate capacitance added to the bus has little effect on speed.

An typical example is shown in Figure 2.3 where a capacitive bus changes with a slew rate of 1.3V/ns. The bus is part of an *n-block* and precharges high during the low phase. In this example, the bus is slowly pulled down on the high phase of the clock. Three alternative designs of receiver circuits are shown. Figure 2.3(A) has two inverters with normal ratio of about 2 between the PMOS transistor and the NMOS transistor and has a switching point of $\frac{V_{dd}}{2}$. In Figure 2.3(B) the switching point of the first inverter is increased by making the PMOS transistor much larger than the NMOS transistor and the second inverter has its switching point lowered. Figure 2.3(C) shows the precharged inverter alternative. To keep the comparison fair, the layout of each of the receivers occupies approximately the same physical area. In the HSPICE simulation of Figure 2.3, the input signal, IN, precharges high on the low phase of the clock in 2ns. During that time, the inverter of (C) also precharges. On the high phase of the clock, the input slowly starts changing. The regular inverter of (A) takes 2.19ns to switch. Increasing the switching point the inverter helps and it only takes 1.32ns to switch its output. This technique is limited by the fact that the change in switching voltage is proportional to the square root of the ratio of the devices. Finally, the precharged inverter takes 1.01ns to restore the edge of the original signal.

Because the buses are precharged, we must consider carefully the effects of coupling from other signals. Two sources of parasitic coupling are present in the design. First, a coupling capacitance is present between any two buses running parallel to each other. Fortunately, this is not a problem because the signals on the bus have a limited slew rate of 1.0 to 1.3V/ns. Secondly, smaller 0.8fF to 1.5fF coupling capacitances are present from signals crossing the bus. Those are usually high speed signals with slew rate as high as 7.0V/ns. Again, the coupling is limited by the large capacitive load on the bus (typically over 1.5pF) compared to the much smaller coupling capacitance (10-15fF). Nonetheless, careful layout is of utmost importance to limit the coupling effects.

2.2 Clocking Methodology and Latch Family

2.2.1 True Single Phase Clocking

The performance of a system and its design complexity is tightly coupled to its clocking methodology. The key issue in high-speed design is to use a high percentage of the clock cycle for the evaluation of the logic and to keep the overhead to a minimum. Multi-phase clocking has been successfully used with a variety of logic families. By controlling the non-overlap time between phases, faults resulting of clock skew are eliminated. However, the overhead of generating and distributing several clock signals throughout a chip makes it undesirable for high-speed design. In a two phase system with non-overlapping clock and level-sensitive latches, the minimum cycle time of the system is [Bak90]:

$$t_{cycle,min} = t_{nonoverlap,max} + t_{latch,max} + t_{logic,max} + t_{setup,max} + t_{skew,max}$$

In order to maximize the percentage utilization of the clock cycle, we must decrease all the sources of overhead; that is, all the terms of the equation except for the critical path of the logic, $t_{logic,max}$. First, we can implement logic in the latch, and make the propagation delay through the latch, $t_{latch,max}$ part of the useful part of the cycle. The setup time, $t_{setup,max}$ is not a problem, since in most latch families, this term vanishes (and is often a negative number) as long as $t_{setup,max} < t_{latch,max}$.

A technique that was first introduced as "true single phase clocking" (TSPC) [YS89, AS90] helps eliminate the extra dead time. First, the non-overlap time, $t_{nonoverlap,max}$ is eliminated by using a single

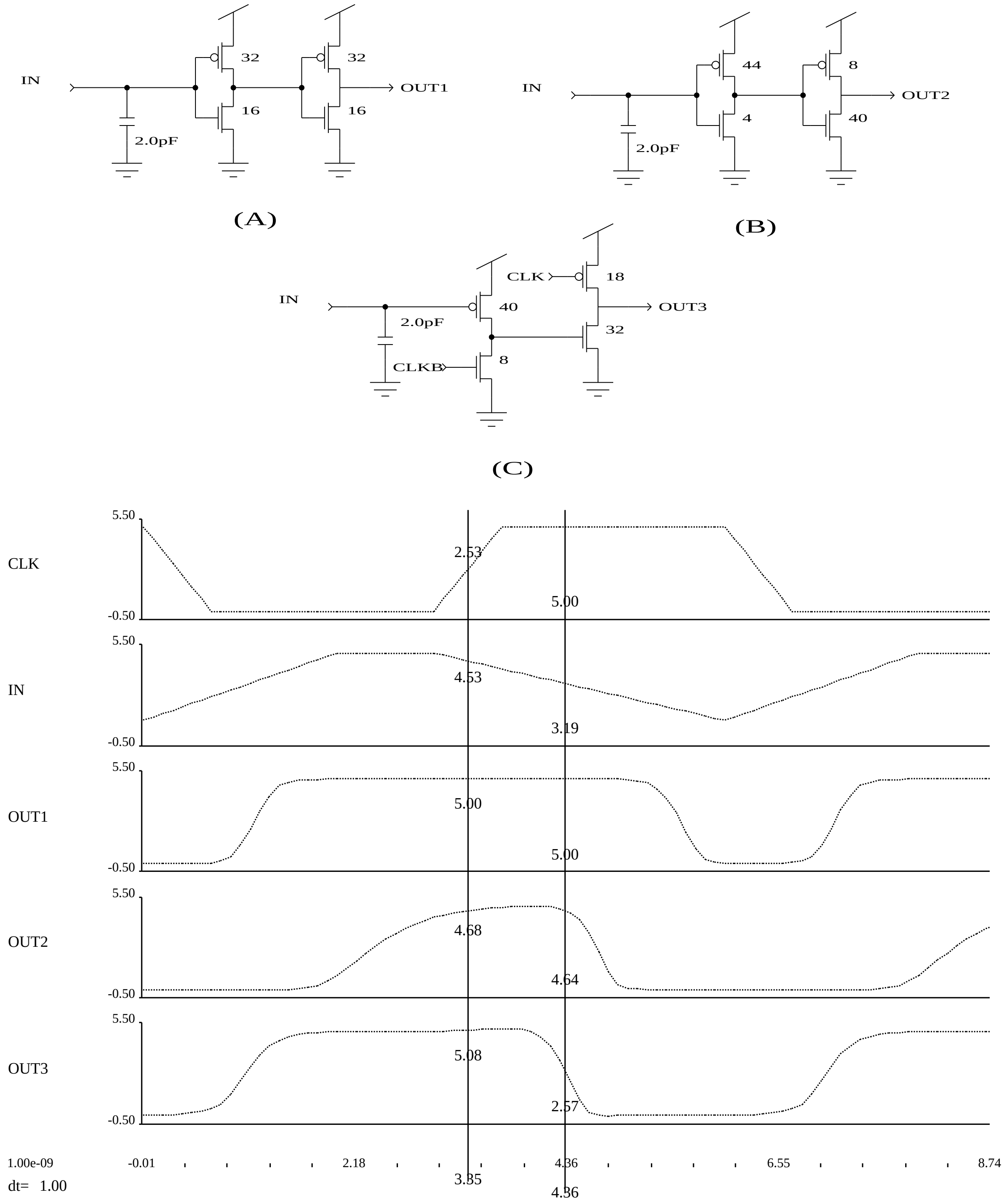


Figure 2.3: Comparison of Inverter, Sized Inverter and Precharged Inverter for Speeding up the Recovery of Buses

clock that is distributed throughout the system. Then, part of the skew due to clock distribution, t_{skew}, is minimized by introducing a family of latches and flip-flops that do not require an inverted clock. TSPC latches have a small number of transistors. They result in fast circuit operation, less latch setup and delay time and less clock load. Although the TSPC methodology can be used to implement single-phase clocking with flip-flops, we prefer using single-phase clocking with latches for two reasons. First, each phase can be used to precharge alternating block of dynamic logic while the other blocks evaluates. In comparison, edge-triggered design are better suited to static logic that does not require precharging. Second, in single-phase clocking certain blocks delays can exceed the half clock period. Because latches are transparent, a block can start evaluating early if the previous one is shorter than half of the clock period. This approach is not possible with edge-triggered designs where flip-flops are hard barriers that cannot be crossed until the following edge.

Most of the remaining clock skew is eliminated by keeping clock distribution simple. We use a single clock node driven by a large driver instead of a more complex distribution tree. This eliminates skew because of mismatched drivers and load in the tree or variation in processing and device strength across the chip.

2.2.2 TSPC Latches

The true single phase clocking methodology requires a latch family able to latch signals on either phase. Figure 2.4 shows the two basic TSPC latches: a p-latch which is transparent when the clock is low and latched when the clock is high and an n-latch which is transparent when the clock is high and latched when the clock is low.

The six transistor implementation does not invert the clock signal and is smaller than its fully static counterpart. When the latch is transparent, it is equivalent to two consecutive inverters. But when it is closed, the pair M1/M3 isolates the input signal from the dynamic memory node formed by the charge stored on the drains of M4/M5.

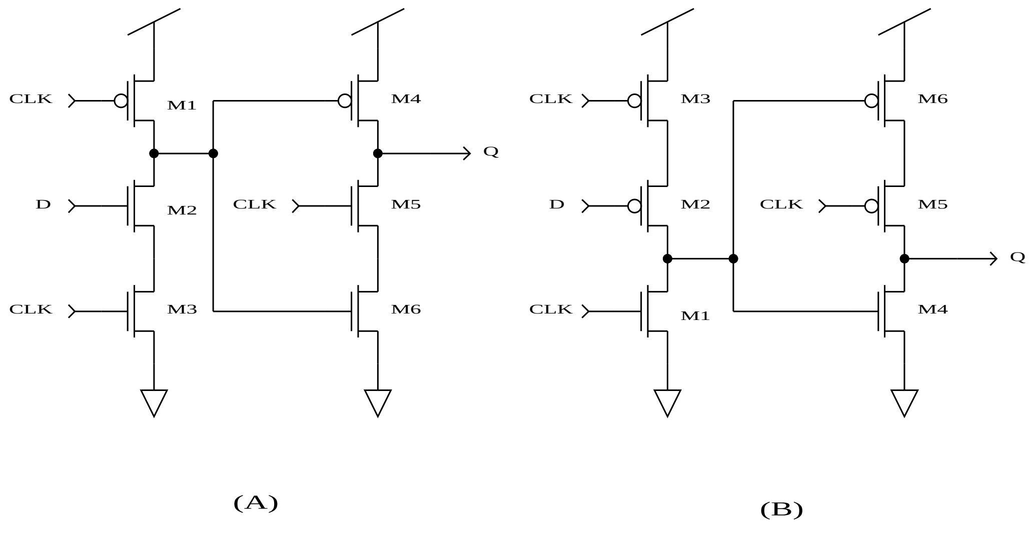


Figure 2.4: TSPC n-type latch (A) and p-type latch (B)

The TSPC latches fit well in a precharged logic scheme. When the last logic gate of a block precharges high before an n-latch (or precharges low before a p-latch), M1/M2/M3 can be eliminated since the precharged value of the last gate will not change the dynamic storage node of the closed latch. The resulting three transistor latch makes the latch overhead minimal. An example of such a combination is shown in Figure 2.5 where and n-type NAND gate is followed by an n-latch.

The proper operation of a system designed around TSPC latches is limited by the edge of the clock. If the clock edge is too slow, two consecutive latches operating on opposite phases will be transparent at

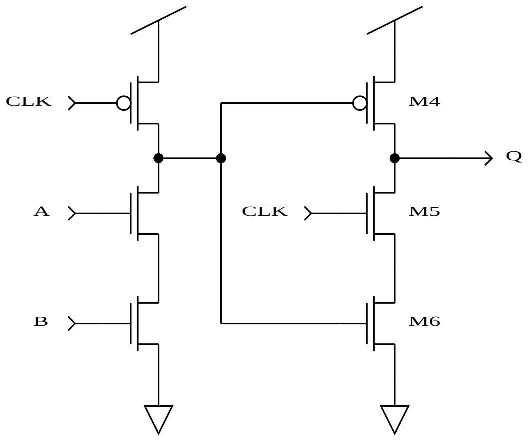


Figure 2.5: A n-type dynamic NAND gate with a TSPC n-latch

the same time. This is a problem common to all latch based designs and special care must be taken in generating a clock signal with fast edges. HSPICE simulation at 5.5V and $0°C$ shows that in the worst case of two latches back to back with no logic in between, the rise and fall time of the clock must be kept below 2.0ns in a $1.2\mu m$ process.

The use of TSPC allows the design of systems with no inverted clocks. While it is possible to build modules in the TSPC methodology using no inverted clocks, it is very difficult in practice to build efficient systems this way. We have relaxed the no clock inversion rule to allow an inverted clock on precharge transistors and power and ground switches, but not on the latches. The inverted clock is not distributed globally, but generated locally when needed. This modification retains the nice properties of TSPC and its resistance to skew while simplifying circuits.

2.3 Layout Considerations

The simultaneous presence of high switching currents, large transistors and dynamic storage nodes is potentially disastrous. To achieve high frequency operation of a circuit, custom layout of the circuits is required to minimize the sources of noise coupling and of parasitic capacitance that can slow down a design.

The power distribution of the chip is of first concern. Because precharged logic switches when an input drops a threshold voltage below power (or above ground), local voltage drop or bounce greater than a MOS threshold voltage on the power or ground nodes can make the precharged logic fail. Large transistor switching can create enough transient current to induce power drops and bounce in inductive and resistive power buses. We address this problem by distributing power and ground in metal-2 perpendicularly to the flow of data in the datapath. Because metal-2 is physically thicker than the metal-1, metal-2 has a sheet resistance about two thirds the resistance of metal-1. This helps minimize the voltage drop created by the high current in the power and ground wires. Also, the power and ground wires of logic blocks in the datapath are not locally connected to their neighbors, permitting local sizing of wires to accommodate blocks with high power needs. This also prevents two consecutive stages of logic from coupling through the power supply. This is especially important for the precharge transistors that can generate large transient currents when they turn on or off.

Minimizing the turn-on and turn-off time of large transistors is also of concern. Very wide transistors present the dual problem of having a large drain capacitance and a long capacitive and resistive gate. The parasitic capacitance of a transistor drain accounts for a large part of the gate delay. By cutting the transistor in half, we can fold it and decrease by about half the parasitic capacitance of the drain, speeding up the switching time of the internal nodes of gates. The RC delay of the gate is also a limitation. Our timing calculations are based on the assumption that the gate turns on uniformly throughout the transistor. However, for the $1.2\mu m$ technology we use, a gate $30\mu m$ wide has an RC delay of 0.2ns from one end of the gate to the other. This results in a progressive switching of the transistor. We estimated that 0.2ns is about the acceptable limit if our timing analysis is to be trusted. Therefore, larger transistors must often be folded multiple times to achieve a reasonable switching time at a cost of extra area.

Figure 2.6 shows an example layout for the bit-cell read bit-line precharge transistors and read sense amplifier. The data flows horizontally in metal-1. Power, ground and clock are distributed vertically in metal-2. A separate power line is used for the folded PMOS precharge transistors of the bit-lines (to the left).

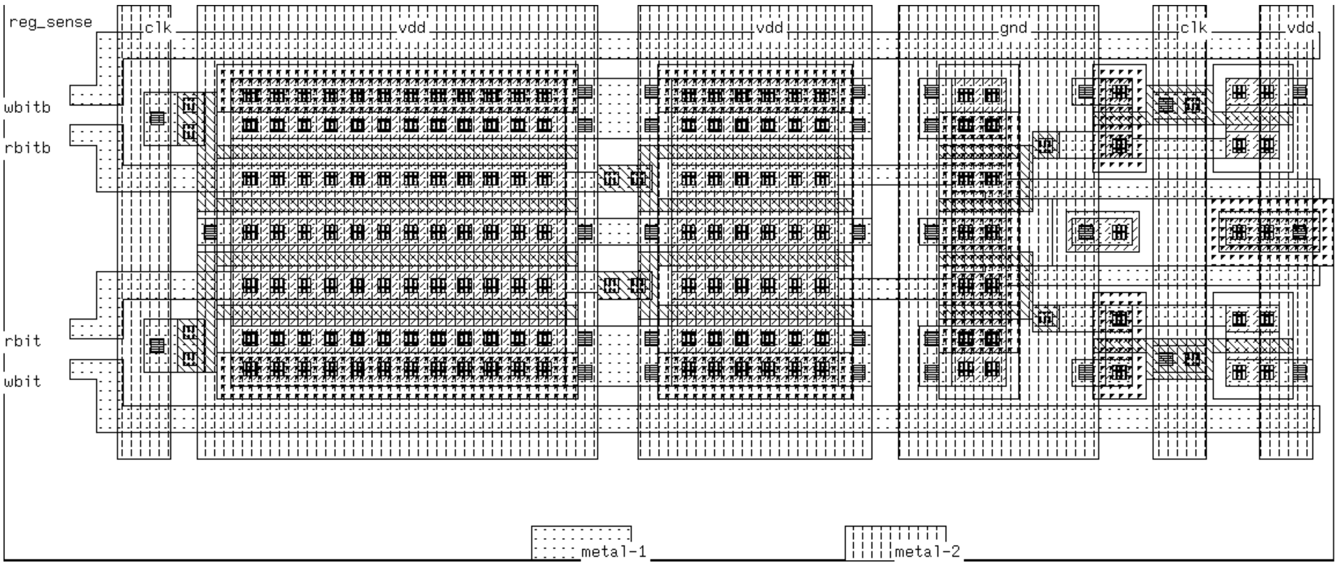


Figure 2.6: An Example of Layout Style

Chapter 3

A 64-bit 250MHz CMOS Integer Datapath

3.1 Motivation

By integrating some of the basic components of microprocessors we intended to demonstrate that the techniques presented are feasible for CPU-size chip design. We also strongly believed that pushing our blocks through the stages of design and fabrication was imperative to our understanding of the issues involved in successfully fabricating high-speed circuits. Clock distribution, power distribution, noise, input and output communication and testing are some of the issues that needed to be solved to make a functional chip.

We have designed a 64-bit integer datapath operating at the target speed of 250MHz in a $1.2\mu m$ CMOS technology with an effective minimum gate length of $1.0\mu m$. The fast datapath consists of two of the basic building blocks of the integer core of a microprocessor: a 64-bit triple-ported register file and a 64-bit integer adder. A complete datapath is more complex than this test datapath, but we believe that the register file and the adder are sufficient to demonstrate that our proposed techniques are viable. Speeding up the carry chain of the 64-bit adder is a challenge for a high-speed design. The register file is also difficult to design for high-speed operation because of the highly capacitive bit-lines. Comparatively, designing a high-speed shifter is not a great challenge: with only one gate delay, a barrel shifter would easily complete in a single cycle. Similarly in a pipelined multiplier, the most difficult part to design is the high-speed adder at the end of the array. The width of the datapath was motivated by the fact that the popular 32-bit architectures are likely to be supplanted by 64-bit architectures.

3.2 Methodology

To meet the tight timing specifications, high-speed designs are mostly full custom. Every circuit must be carefully tuned and precautions must be taken in the layout to avoid electrical problems such as coupling or extraneous RC delays.

We have adopted a design methodology suitable to such full custom circuits that is flexible enough to allow an iterative process. First, the timing specification of the system is defined. Based on the timing requirements and the number of blocks in the system, we allocate the different numbers of pipeline stages necessary for each block. Because blocks are made of alternating p-stages and n-stages, this is a critical phase of the design. For example, designing a two stage critical path as a p-stage/n-stage combination rather than an n-stage/p-stage can have an impact on its speed and its size. A critical path made of n-type gates will often be more compact than one made of p-type gates.

We evaluate the design of each block by entering the schematics in the Workview [VIE89] schematic capture system. Using the netlist generator, we generate SPICE compatible netlists and simulate the critical paths with the circuit simulator HPSICE [Met90]. To obtain an accurate simulation, we add parasitic capacitance that might be present along with the appropriate transistor source and drain capacitance. Then, we select one design for each block, based on speed, ease of pipelining and estimated size.

At this point we create a behavioral model of the block in the C language. This program is used to verify the correctness of the function and later to verify the intermediate results at the latches. It is also used to generate random test vectors for the switch level and the circuit level simulators.

Next, we use the layout editor MAGIC [SMHO86] to enter the leaf cells of the block. Tiling of the leaf cells is done manually in MAGIC. Design rule violations are checked with the MAGIC design rules checker.

The switch level simulator IRSIM is used to check the logical correctness of the extracted layout. IRSIM also gives a first order timing specification that can be used to detect slow or fast paths in the design. Typically, we simulate the block with 100,000 to 500,000 random vectors plus a number of hand picked vectors designed to exercise the critical paths of the design.

Finally, the SPICE-like circuit simulator CAzM [ER90] is used to simulate blocks of up to 20,000 transistors. 200 to 400 vectors are typically simulated to obtain accurate timing information and power estimates at temperature and voltage.

Before assembling the top level of the chip, the blocks are checked for missing well contacts, shorts and proper CIF [WS80] generation. The chip layout is then extracted and an IRSIM simulation at the pads is done to verify its functionality.

3.3 Chip Overview

3.3.1 Description

The basic structure of the test chip is shown on Figure 3.1. The core of the datapath consists of 32 64-bit registers, **regfile**, and a 64-bit adder, **adder**. The core fetches two operands from the register file, adds them together and stores the result back in the register file. Two buses are used for internal communication. The write-back bus, **wbus**, is used to write a result from the adder or the register file back into the register file. A bypass bus, **bbus**, bypasses the register file and forwards results directly into the adder. A register read, add and write back cycle takes four cycles to complete, but the pipelined implementation allows a new operation to start every cycle.

Testing at 250MHz is a difficult task. We opted for a test strategy where the internal datapath is continuously clocked at high-speed and controlled by an internal PLA, **controller**. A test identification number, **tid** and a reset signal, **reset**, are applied from the pins to start the controller. When the program completes, a **done** signal is asserted.

Data communication to the outside world is done at low-speed with two 64-bit shift registers. Data is serially shifted in at 20MHz from the pin of the input shift register, **buffer_in**. Then, it is put at 250MHz on **wbus** to be written in the register file. Similarly, the output shift register **buffer_out** is used to read values out of the core of the datapath. The data on **wbus** is loaded in parallel into **buffer_out** at 250MHz. The data is then shifted serially at 20MHz out of the register. The slow-speed clock of the shift registers is completely asynchronous from the high-speed system clock of the core.

3.3.2 Datapath Timing

As our clocking discipline dictates, each functional block is broken into alternating p-blocks and n-blocks. The timing diagram of Figure 3.2 shows the operation of the core pipeline.

The register file is composed of a p-block decoder and a n-block RAM array and sense logic. First, the register addresses **rA** and **rB** are decoded in the low phase of the clock. The register file RAM array is read during the following high phase and the data out of the register file, **A** and **B**, is latched on the falling edge of the clock. The following low phase is used to drive the data on the bypass bus to the adder. The adder is pipelined in in three n/p/n blocks, **adder1**, **adder2** and **adder3**. It takes three phases to evaluate, but a new add can be started every cycle. Finally, the result from the adder, **S**, is driven on the write-back bus during the low phase and the register file is written during the following high phase.

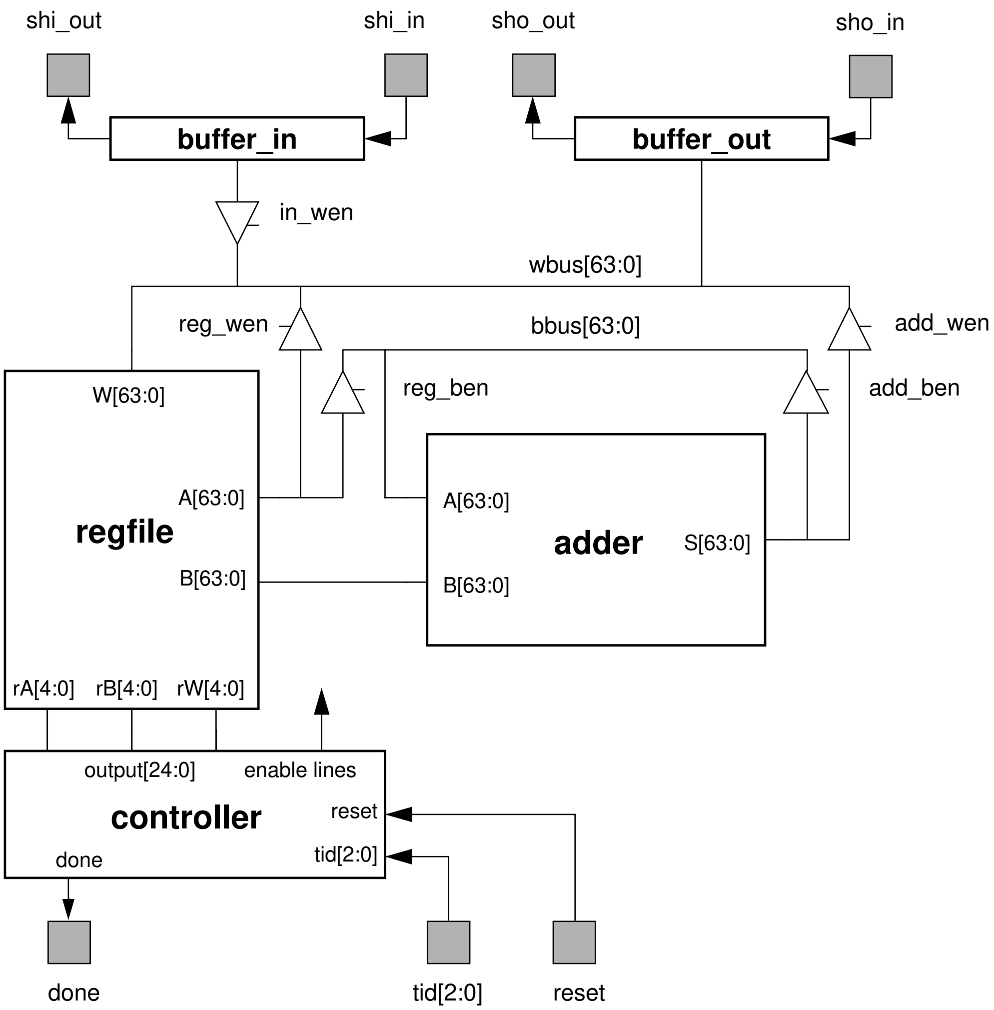


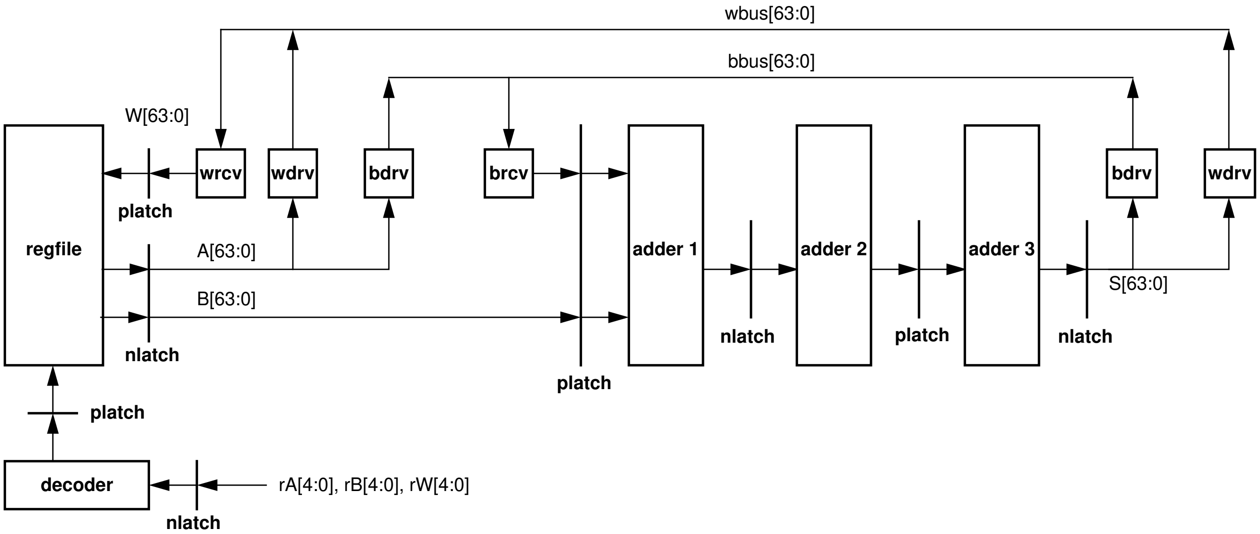
Figure 3.1: Fast Datapath Block Diagram

3.3.3 Floorplan

The fast datapath was implemented using MOSIS SCMOS design rules and fabricated in the HP CMOS34 1.2μm process. Figure 3.3 shows a die photo of the datapath. The die is 6mm on a side. In the center, the adder is $3.0 \times 2.4mm^2$. To its left, the register file measures $3.0 \times 1.2mm^2$ for the RAM array and the sense and write circuitry. The address drivers and decoders are to the top and bottom of the register file and each occupy $0.19 \times 1.32mm^2$. The datapath is compact because every 1-bit cell of the block has the same vertical dimension of 64λ ($38.4\mu m$). The bus logic and shift registers are located to the right of the adder. They increase the datapath length by another $0.42mm$. To the top, a PLA based finite state machine is used to control the operation of the datapath. The controller measures $1.1 \times 1.5mm^2$. Note that all the block measurements include power, ground and clock buses.

3.3.4 Pin Count and Packaging

Five pads are used to control the datapath: a 3-bit test identification number **tid[2:0]**, a reset signal **reset** and a completion signal **done**. Four pads are used to shift data in and out of the two shift registers. The input shift register pads are **shi_in** and **shi_out** and the output register pads are **sho_in** and **sho_out**. The high-speed clock requires three pads: one for the input signal and two ground pads



read address, rA, rB	eval	pc	eval	pc	eval	pc	eval	pc	eval	pc	eval
read decoder	pc	eval	pc	eval	pc	eval	pc	eval	pc	eval	pc
read row select	stable	eval	stable	eval	stable	eval	stable	eval	stable	eval	stable
read bitline	eval	pc	eval	pc	eval	pc	eval	pc	eval	pc	eval
read sense amp	eval	pc	eval	pc	eval	pc	eval	pc	eval	pc	eval
data out, A, B	eval	stable	eval	stable	eval	stable	eval	stable	eval	stable	eval
wbus	pc	eval	pc	eval	pc	eval	pc	eval	pc	eval	pc
adder1	eval	pc	eval	pc	eval	pc	eval	pc	eval	pc	eval
adder2	pc	eval	pc	eval	pc	eval	pc	eval	pc	eval	pc
adder3	eval	pc	eval	pc	eval	pc	eval	pc	eval	pc	eval
sum, S	eval	stable	eval	stable	eval	stable	eval	stable	eval	stable	eval
bbus	pc	eval	pc	eval	pc	eval	pc	eval	pc	eval	pc
write address, rW	eval	pc	eval	pc	eval	pc	eval	pc	eval	pc	eval
write decoder	pc	eval	pc	eval	pc	eval	pc	eval	pc	eval	pc
write row select	stable	eval	stable	eval	stable	eval	stable	eval	stable	eval	stable
write bitline	eval	pc	eval	pc	eval	pc	eval	pc	eval	pc	eval

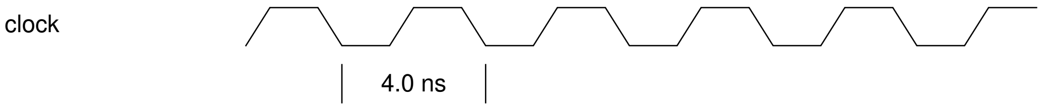


Figure 3.2: Datapath Core Block Diagram

for shielding. Similarly, a clock output signal used to monitor the internal clock requires three additional pads.

The package available from MOSIS is an 84 pins ceramic PGA with no dedicated power or ground low-inductance planes. Therefore, the remaining pads have be used for power and ground in order to minimize the inductance on these nodes. The average inductance per pin was estimated to be 10 to 15nH (MOSIS estimate) for the package and an additional 5nH for a 5mm average bonding wire from the chip to the package. Because the single clock scheme results in large instantaneous current when the clock is switching, a large number of pads was required to reduce the power and ground inductance and bounce. To a first approximation, the combined inductance of n parallel inductors is:

$$L_{eff} = \frac{1}{\sum_{i=1}^{n} 1/L_i} \tag{3.1}$$

For n inductors of equal value L, this simplifies to:

$$L_{eff} = \frac{L}{n} \tag{3.2}$$

This does not take into account mutual inductance of two bonding wires next to each other which make the actual parallel inductance larger than predicted by this equation. With the goal of obtaining power and ground inductances less than 1nH, over 20 pads of each type are required. Table 3.1 shows a summary of the pads used for the fast datapath.

Pad Name	Type	Count
Vdd	power	26
GND	ground	27
reset	in	1
tid	in	3
shift, shiftb	in	2
done	out	1
shi_out, sho_out	out	2
shi_in, sho_in	in	2
clkin	in	3
clkout	out	3
Total		70

Table 3.1: Pad Count

Figure 3.3: Fast Datapath Die Photograph

3.4 The Adder

The 64-bit adder is pipelined over three phases, each stage separated by alternating p or n latches. The latency through the adder is one and a half cycles, but a new addition can be started every cycle. The low-level circuits for the adder are fairly straightforward and comprises a Kogge-Stone binary lookahead tree [HCL87]. However, a unique 2 to 1 reduction of the carry chain results in a significant reduction in area without affecting the speed. The adder was designed to run at 250MHz at $70°C$ and $Vdd = 4.5V$ in a $1.2\mu m$ process. It occupies a total area of $5103 \times 3954\lambda^2$.

3.4.1 Logic Overview

Looking at building the fastest possible adder, several implementations were considered before making a design decision. In particular, an early evaluation of circuits showed that a carry select implementation was competitive for 32 bits, but was slower for 64 bits than the implementation finally chosen: a Kogge-Stone binary lookahead tree with a 2 to 1 reduction of the carry chain.

The adder presents the difficult task of having to propagate a carry all the way across 64 bits in the worst case, both logically and physically. By logically, we mean that a minimum number of stages are required to implement the carry propagation and by physically, that the carry will have to electrically be forwarded all the way across the 64-bit datapath, resulting in a large wiring capacitance.

The Despain-deDood Carry Chain Reduction

The 2 to 1 reduction in the carry chain was invented by Professor Alvin M. Despain and Paul deDood at U.C. Berkeley [DdD89]. The reduction starts by rewriting the carry term of a Manchester carry chain of equation 3.3 as equation 3.6:

$$\begin{aligned} C_{2n} &= G_{2n} + (P_{2n}.C_{(2n-1)}) && (3.3)\\ &= (G_{2n} + P_{2n}).(G_{2n} + C_{(2n-1)}) && (3.4)\\ &= ((A_{2n}.B_{2n}) + (A_{2n} + B_{2n})).(G_{2n} + C_{(2n-1)}) && (3.5)\\ &= P_{2n}.(G_{2n} + C_{(2n-1)}) && (3.6) \end{aligned}$$

By doing so on every even term, the standard carry chain of Figure 3.4(A) can be modified to look like the carry chain of Figure 3.4(B) without any change in the logic. We notice that the carry chain now has adjacent OR functions and adjacent AND functions rather than alternating ANDs and ORs. We can combine two consecutive ANDs or ORs into one gate by creating the P'_n and G'_n terms:

$$\begin{aligned} P'_n &= P_{(2n+1)}.P_{2n} && (3.7)\\ G'_n &= G_{2n} + G_{(2n-1)} && (3.8) \end{aligned}$$

The reduced carry chain looks like Figure 3.4 and is half the length of the original chain. The carry chain terms are now C'_n and not the original C_n and the original carry signal is reconstructed once the carry chain has evaluated as:

$$\begin{aligned} C_{2n+1} &= G_{(2n+1)} + C'_n && (3.9)\\ C_{2n} &= P_{2n}.(G'_n + C'_{(n-1)}) && (3.10) \end{aligned}$$

The Kogge-Stone Lookahead Tree

Even though the Despain-deDood reduction scheme cuts the carry chain in half, a regular Manchester carry chain is still too slow for 32 bits. Instead, a Kogge-Stone binary lookahead tree is used to evaluate the carry chain in $O(log(n))$ stages, that is five stages for 32 bits.

Figure 3.5 shows the block diagram for a two-bit slice of the adder. The adder comprises three blocks of 3, 4 and 3 stages respectively. The first block of the adder is an n-block and has only three gate delays in addition to the latch. The **g** and **p** gates create the generate signal $G_{2n} = A_{2n}.B_{2n}$ and

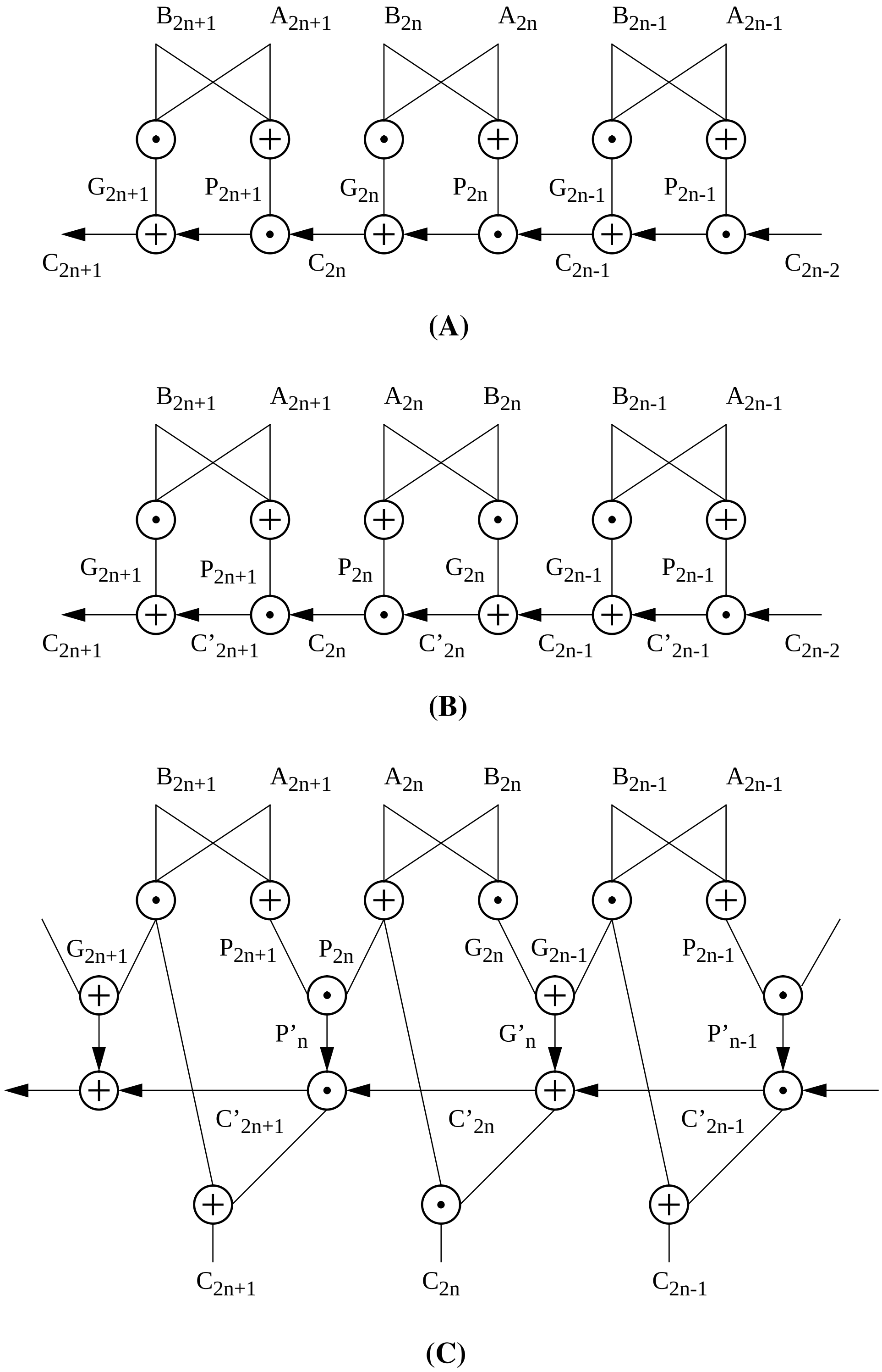
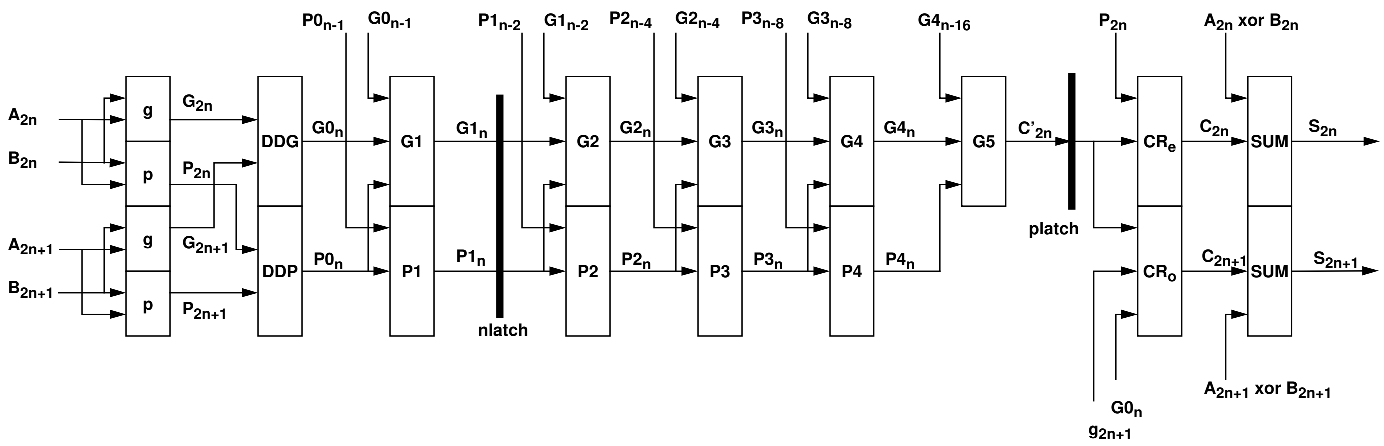


Figure 3.4: Despain-deDood Carry Reduction

the propagate signal $P_{2n} = B_{2n} + A_{2n}$. Then, the propagate and generate signals of two consecutive bits are combined in the carry reduction blocks **DDG** and **DDP** producing $G0_n = G_{2n} + G_{(2n-1)}$ and $P0_n = P_{2n}.P_{(2n+1)}$ respectively. Also, the first stage of the Kogge-Stone lookahead binary tree, **G1** and **P1** completes within this block.

The second block is a p-block and comprises stages 2 to 5 of the binary lookahead tree. Each stage of the tree consists of a pair of propagate/generate blocks. Figure 3.5 shows only an intermediate bit-slice, where all the stages are needed. For low order bits, the corresponding carry is propagated in fewer than 5 stages and the unnecessary P and G stages are replaced by buffer stages.

In the last block of the adder, the real carry C_n is reconstituted in the $\mathbf{CR_e}$ and $\mathbf{CR_o}$ blocks. P_{2n} and $G_{(2n+1)}$ are forwarded and pipelined from the first block. Finally, the **SUM** block calculates the result of the addition with an exclusive OR between C_n and $A_n \oplus B_n$, also forwarded from the first block.



p,g Propagate and generate block
DDP, DDG Despain/deDood carry chain reduction
P1, G1 1st stage of Kogge-Stone tree
P2, G2 2nd stage of Kogge-Stone tree
P3, G3 3rd stage of Kogge-Stone tree
P4, G4 4th stage of Kogge-Stone tree
P5, G5 5th stage of Kogge-Stone tree
CRo, CRe Re-generate the real carry Cn from C'n
SUM Sum block

Figure 3.5: Adder Block Diagram

The carry reconstitute stage of the carry chain reduction introduces one additional level of logic compared to a six-stage binary lookahead tree. Yet, it is faster than other designs and also presents a saving in area. The carry reduction scheme introduces the need for a forwarding bus to save the propagate term of the odd bits and the generate term of even bits. These signals are needed when reconstituting the real carry term in the last block of the adder and must be latched at the end of the first and of the second block. Table 3.4.1 shows the area saving when the carry reduction technique is used. Given the cell pitch of the datapath, a bypass bus increases the cell area by 12% and four additional latches are needed. While a six-stage carry lookahead tree does not require the four latches and the extra bus, the 64-bit carry lookahead tree requires twice as much logic as the 32-bit version. This represent an estimated 21% increase in area over an implementation that uses carry reduction.

3.4.2 Circuit Overview

The schematics of the adder in Figures 3.7, 3.8 and 3.9 show the logic for one two-bit slice of the adder. All transistors have a minimum length of 2 lambda and their width is indicated in lambdas. Each block

	Carry Reduction (λ)	KS Logic (λ)	Extra Latches (λ)	Other (λ)	Total Width (λ)
With Carry Reduction	330	1800	190	1634	3954
No Carry Reduction	0	3150	0	1634	4784

Table 3.2: Adder Area Comparison

consists of alternating precharged n-stages and p-stages, terminated by a n-latch for the n-blocks **adder1** and **adder3** and a p-latch for the p-block **adder2**.

The two design problems of the adder, reflected in the large transistor sizes, are the large interconnect wires and the slow p-stages. The wiring capacitance in the adder grows significantly in the later stages of the lookahead tree. In the last stage of the tree, the signal $G_{(n-16)}$ coming from the output of the **G4** stage has to travel half way across the datapath (1.6 mm) before reaching the input of the **G5** stage. To get a short gate delay through that stage, we increase the size of the **G4** stage. Unfortunately, that increases the fanout of the previous stages that in turn also have to be large. To make matters worse, **P3** and **G3** must be p-stages to comply with the alternating p-stage/n-stage design rule. They are very large to offset the weaker drive of the p-type transistors. The alternative to a p-stage would have been to use domino logic where the p-stages are replaced by n-stages with inverters on their inputs. However, the speed gained by the stronger drive of an n-stage does not sufficiently offset the inverter delay introduced between stages.

The speed of the adder is limited by the critical path in the second block. This occurs when all the propagate blocks and the generate stages are turned on successively by an incoming $P_{(n-2)}$ signal and all the generate blocks are off: when one of the input of the adder is all 1's while the second input is exactly 1. The HSPICE simulation of Figure 3.6 shows the propagate and generate signals rippling in the tree. During the high phase of the clock, the tree is precharged and P0<1>⊥ and G0<1>⊥ are stable by the beginning of the falling edge. The tree starts evaluating 0.60ns after the falling edge as P1<3> and G1<3> switch. C'<31> is stable 1.48ns after the falling edge of the clock and the latch output C'<31>⊥ is stable 1.88ns after the falling edge.

3.4.3 Power Consumption

The power consumption of the adder comes mostly from the precharge logic and therefore is very dependent on data. CAzM simulations with random data report an average power consumption of 2.87 Watts.

Part of the power consumption is actually burned during precharge fights. By trading speed and area for power consumption, the design eliminates the ground switch for n-type stages and the power switch for p-type stages that disconnect the gate from its power source during precharge. Instead the circuit relies on the previous stage to precharge to end its own precharge fight. This creates a precharge rippling effect and a gate cannot be fully precharged until all the previous gates have precharged. Making the precharge transistor large enough helps in the event of a fight, because it will quickly overcome the pull-up (or pull-down) network even if the previous gate has not completely precharged. This happens at the cost of an increase in power consumption.

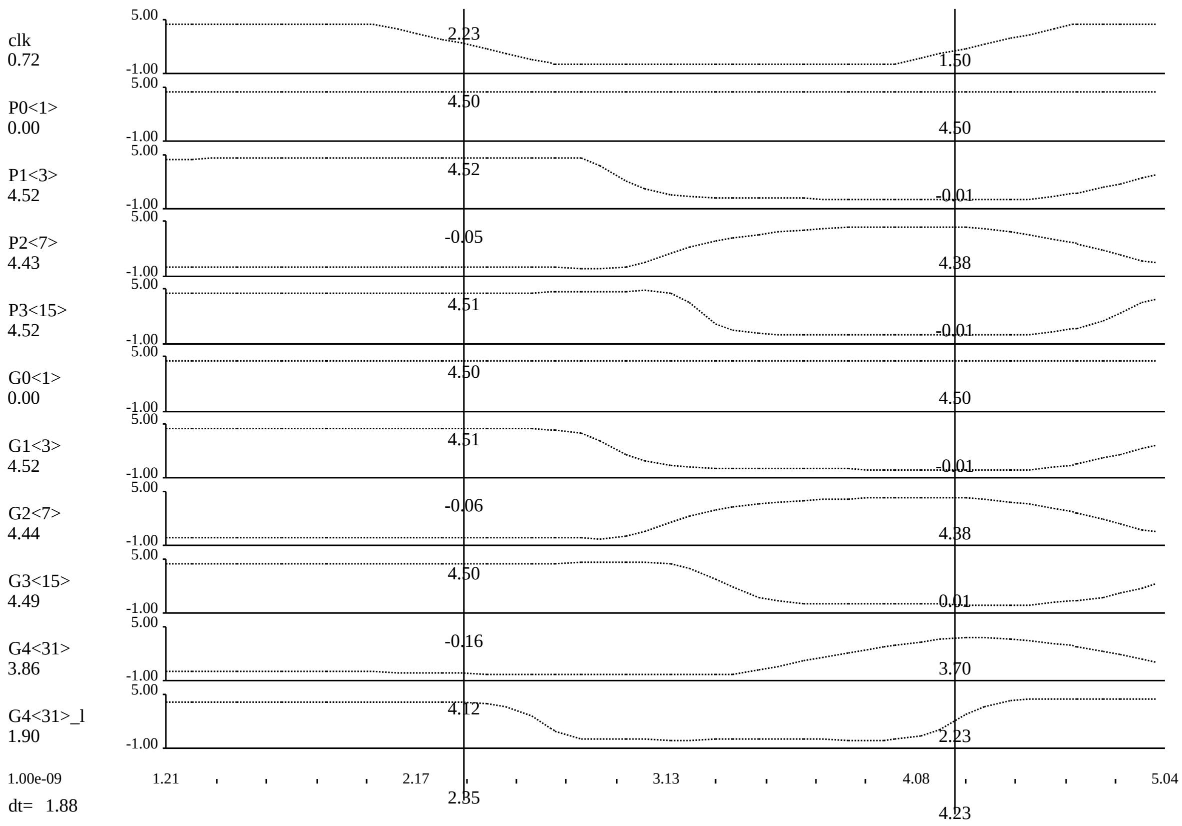


Figure 3.6: Adder Critical Path Simulation

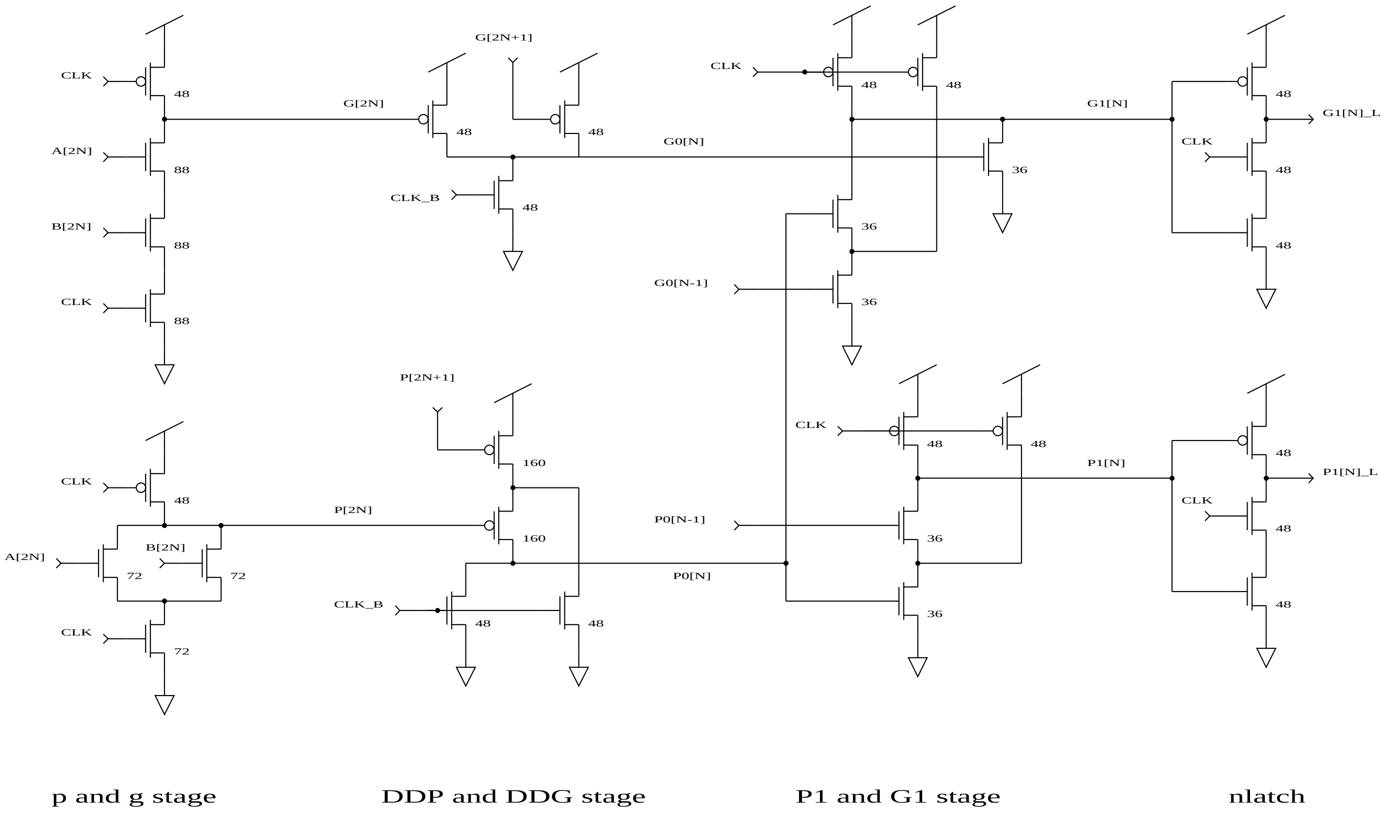


Figure 3.7: Adder Block 1

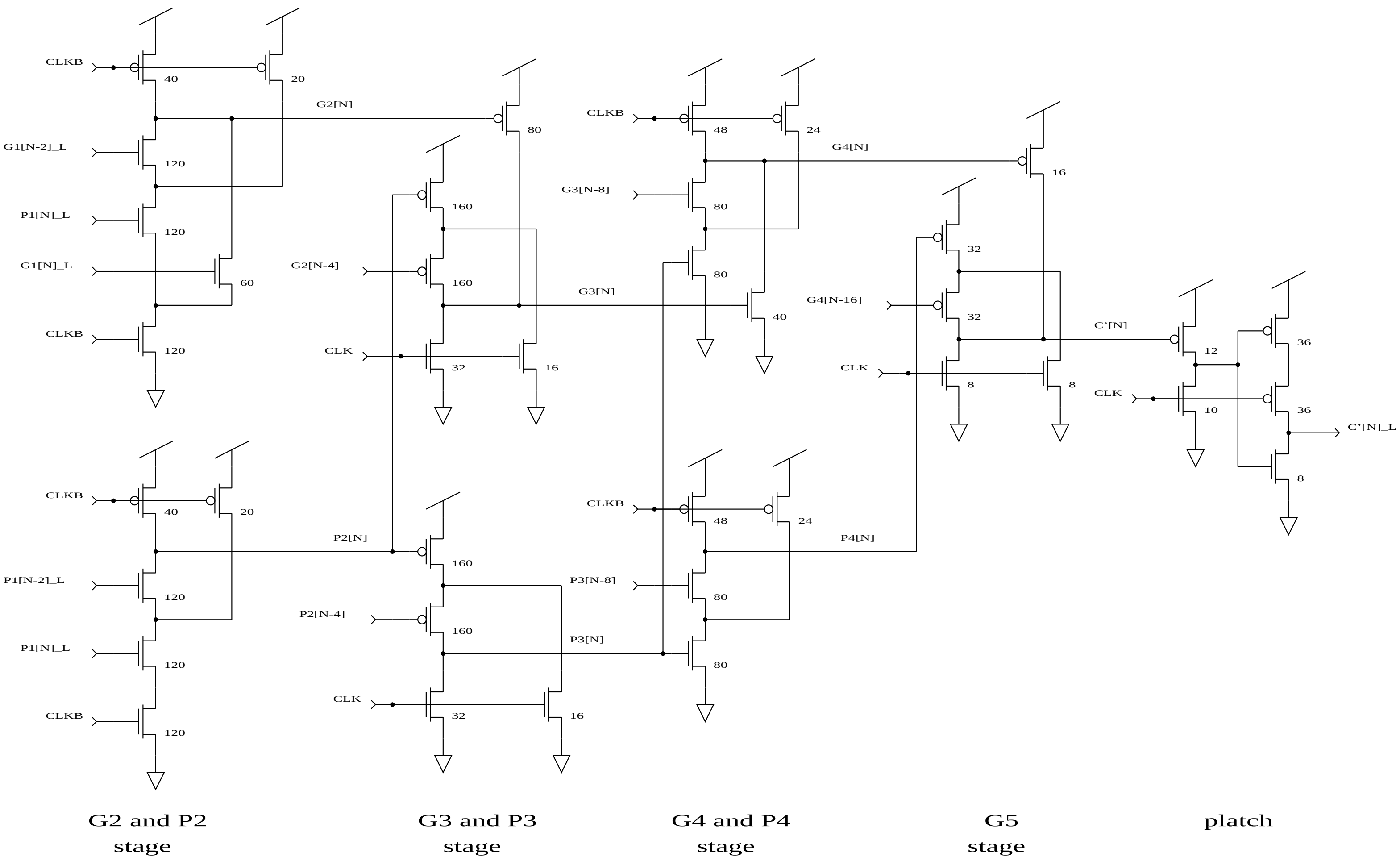


Figure 3.8: Adder Block 2

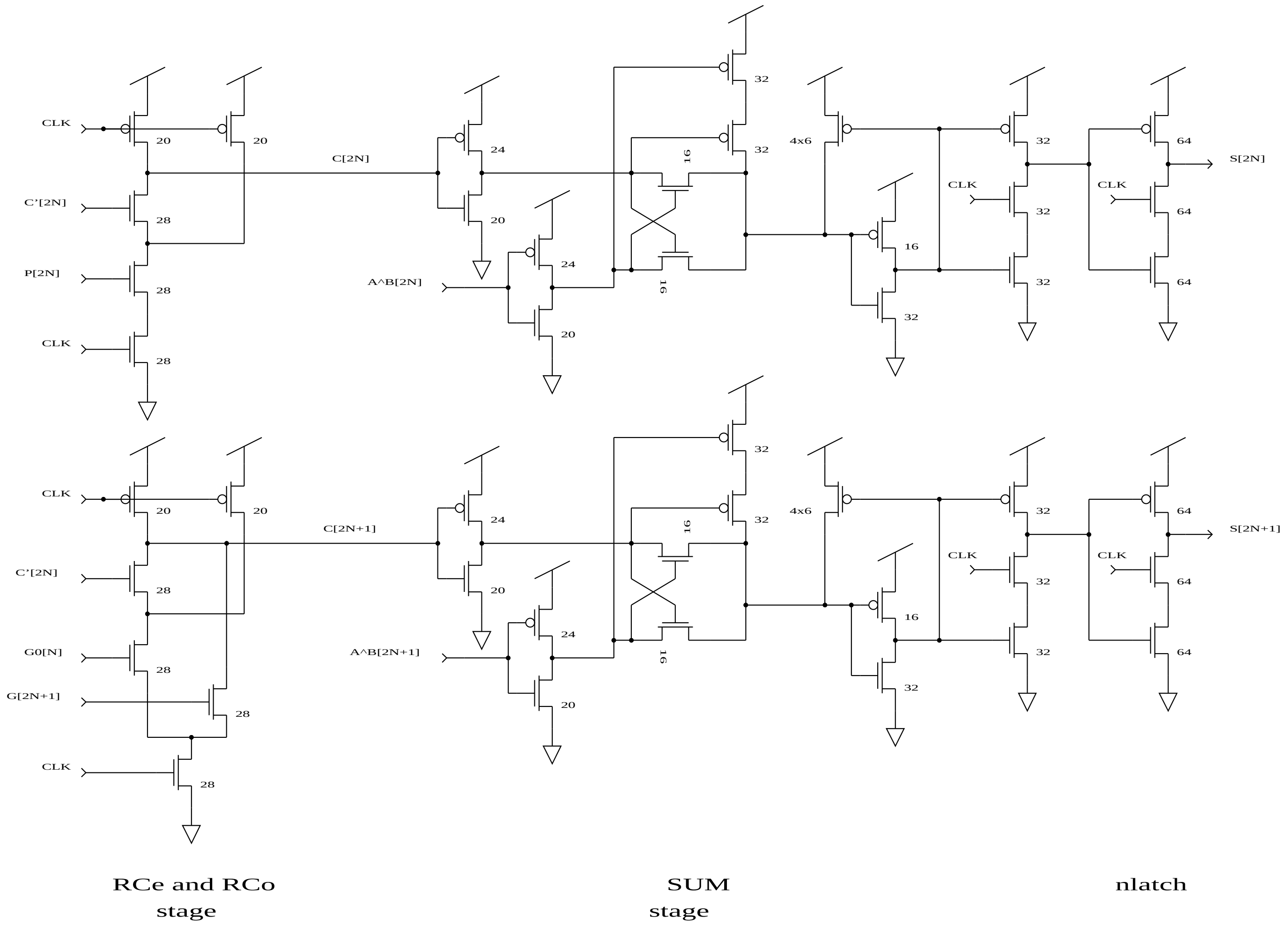


Figure 3.9: Adder Block 3

3.5 The Register File

The register file consists of a static array of 32 64-bit registers, dynamic decoders and read/write circuitry. It is designed to run synchronously at 250MHz at $70°C$ and $Vdd = 4.5V$ in a $1.2\mu m$ process. The register file can sustain up to two reads and one write every cycle, with a one cycle latency for read. The register file is not write-through, which means that the value of a read is undefined if the same register is also written during the same cycle. In the final layout, register 0 is hardwired to the integer value 0 and cannot be written.

Figure 3.10 shows the floorplan of the register file. At the center, a 2Kb RAM array organized in 32 rows of 64 columns stores the register values. Each bit-cell is $64 \times 41\lambda^2$, or $4096 \times 1312\lambda^2$ for the whole array. To the right, two single-sided sense amplifiers are used to read the read bit-lines of the RAM array. The write circuitry is adjacent to the sense amplifiers. It differentially drives the write bit-lines of the array when writing a register. Both the sense amplifiers and the write drivers are pitch-matched to the bit-cell. Each set of two sense amplifiers and write drivers fits in $64 \times 724\lambda^2$, or $4096 \times 764\lambda^2$ for 64 bits.

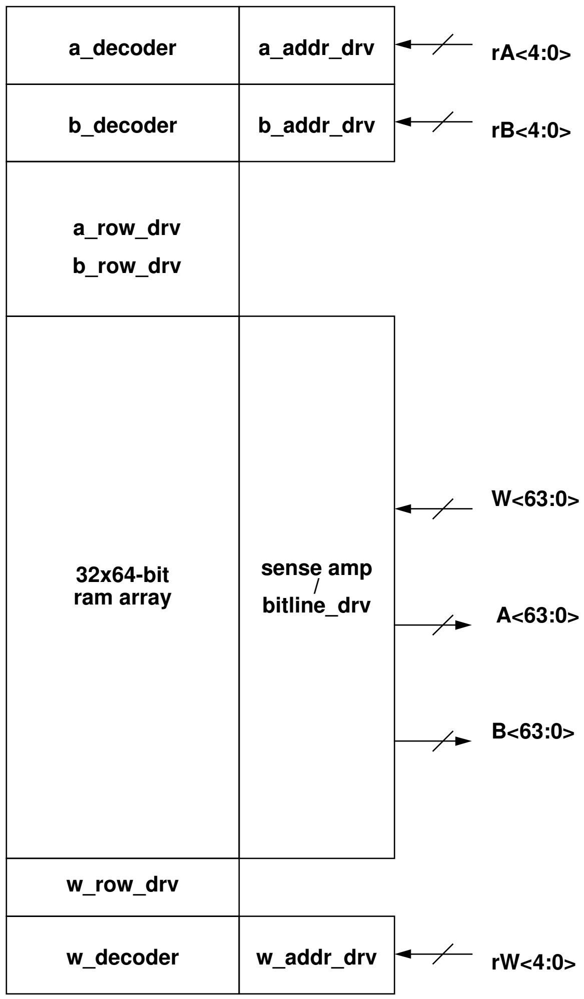


Figure 3.10: Register File Floorplan

Because the register file operates at high frequency, the decoders cannot be multiplexed. Therefore, each of the three read and write ports has its own 5-bit address drivers, 32-bit for decoder and 32-row select drivers. The logic for the two read ports is located on top of the RAM array and the logic for the write port is at the bottom. Each decoder/driver pair is $824 \times 1310\lambda^2$ and the address driver is an extra $295 \times 590\lambda^2$. Overall, the register file occupies an area of $7845 \times 2032\lambda^2$, including the power, ground and clock buses.

3.5.1 Block Diagram Overview

Figure 3.11 shows a detailed block diagram of the register file for a single bit. Its operation is broken down over a p-phase and an n-phase. The first phase is used to decode the address. The address of the the register accessed must be stable by the beginning of the low phase of the clock. During the p-phase, the **decoder** block selects one of the 32 row select lines and the inputs to **row_drv** are latched on the rising edge of the clock to be stable during the high phase.

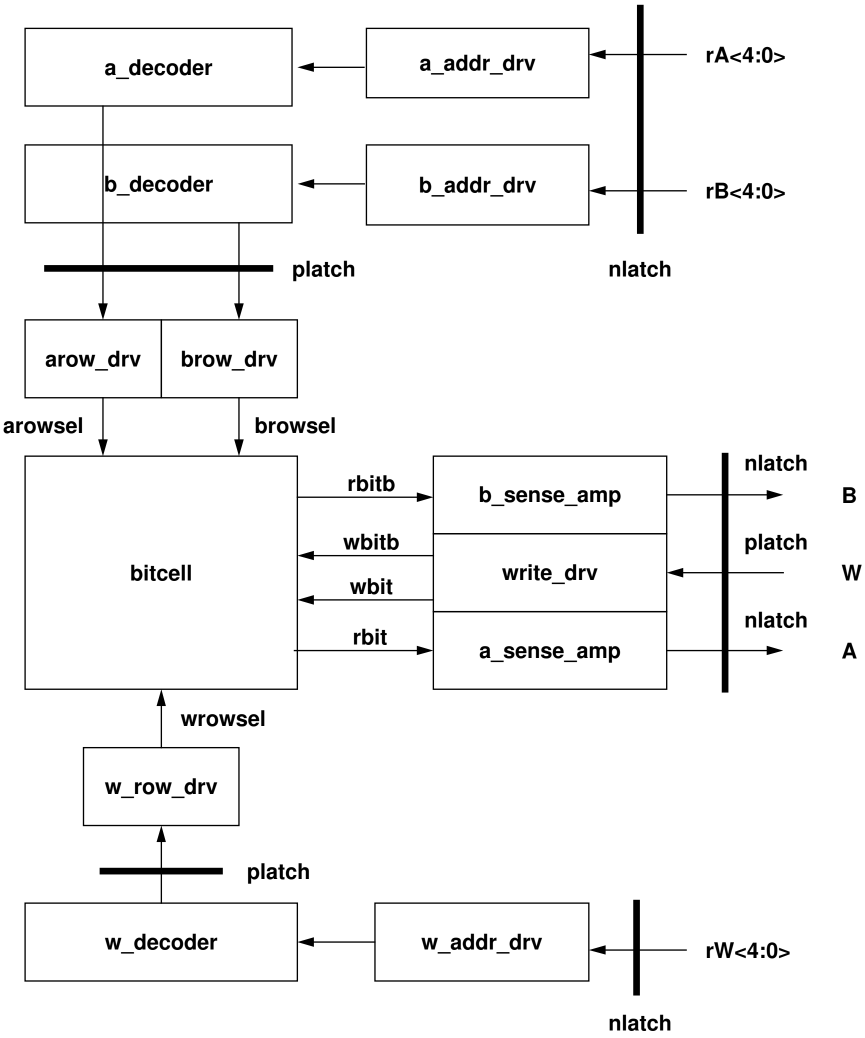


Figure 3.11: Register File Block Diagram

For a read operation, the n-phase is used by the storage element **bit-cell** to drive one or both of the two complementary precharged bit-lines, **rbit** and **rbitb**, depending on the row select lines enabled. Because the bit-lines are precharged, the bit-cell only needs to discharge the bit-lines when necessary. Then, the read bit-lines are sensed by the single-sided sense amplifier, **sense_amp** and the data read is latched on the falling edge of the clock.

For a write operation, the data to be written must be ready by the rising edge of the clock. During the high phase of the clock, the write driver **write_drv** differentially drives the complementary write bit-lines, **wbit** and **wbitb**, and the value is stored in **bit-cell** by the falling edge of the clock.

3.5.2 Circuit overview

Decoder Tree

The circuitry of the address drivers and the decoders [LS87] is shown on Figure 3.12. This partial schematic shows the address driver for one address bit. It also shows the branch of the decoder tree for bit 0. The rest of the decoder tree can be easily completed by symmetry.

The address driver consists of transistors M1/M3/M5 and M2/M4/M6. Five drivers are used to address the 32 registers. During the high phase of the clock, the M1/M2 pair discharges the differential address lines. As the address lines are precharged low, the pull-up PMOS transistors M8/M9/M10/M11/M12 completely precharge high the intermediate nodes of the decoder tree. The dynamic tree requires precharging to prevent charge sharing during the evaluation. Because the address lines are precharged, the address ADDR<n> must arrive before the falling edge of the clock to allow the inputs of M3 and M4 to be stable on the falling edge of the clock. The addresses ADDR<n> are latched on the falling edge of the clock.

During the high phase of the clock, the PMOS transistors M5 and M6 turn on, pulling up one of the differential address line A<n> or A<n>_B. The addresses turn on only one path to ground in the decoder tree through one set of pass transistors. The speed of the decoder resides in the exponentially increasing pull-down NMOS chain constituted by M13, M14, M15, M16 and M17. By doubling the size of each transistor closer to ground, the delay through a pull-down chain of n transistors is only $O(n)$ and not $O(n^2)$ as would be the case for identical size devices. Also, the large PMOS transistor M18 at the end of the chain behaves as a sensing device that turns on as soon as the output of the decoder is a PMOS threshold below Vdd. Finally, the decoder output is latched on the rising edge of the clock. Because the decoder outputs are precharged low during the low phase of the clock, the row select signals RSEL<n> of the RAM array are also low and there is no risk of reading or writing the wrong register if the decoder output is not completely stable by the rising edge of the clock.

Each decoded bit is made of 5 $\frac{16}{2}\lambda$ NMOS pass transistors and 5 $\frac{16}{2}\lambda$ PMOS pull-ups. The larger size pass transistors and pull-ups are simply obtained by connecting in parallel 2, 4, 8 or 16 $\frac{16}{2}\lambda$ NMOS or PMOS transistors at the different stages of the chain. The resulting structure is rectangular and lays out very nicely because of the regularity.

Figure 3.12 shows a CAzM simulation of the decoder at $70°C$ and $Vdd = 4.5V$. ADDR<0> changes during the high phase of the clock at 6.25ns, changing the decoder address from 1 to 0. The output of the decoder RSEL<0> turns on 1.69ns after the falling edge of the clock and the latched output ROW_SELB<0>_L is valid after 1.75ns. The row select signal ROW_SEL<0> is valid after 2.12ns, stealing 0.12ns from the next phase. It is important that no overlap of the decoder row select signals occur or erroneous data will written in the RAM bit-cell. As we can see, the row select turns off 1.18ns after the falling edge of the clock, leaving 0.94ns between the two enabled row select signals. Because the decoder tree is precharged and any bit selected has to be pulled down through 5 identical pass transistors, the evaluation time is the same for every bit.

The Read Circuitry

The read circuitry of the register file is shown on Figure 3.13. The basic RAM bit-cell is made of the static cross coupled inverters M1/M2/M4/M5. The two pass transistors M3 and M6 provide two single ended read ports to the bit-cell. The access devices and the cross coupled inverter are carefully sized to preserve the cell contents on a single-sided read. To speed up the reading of the cell, the read bit-lines RBIT and RBITB are precharged high by M7 and M8 so that the bit-cell only has to pull-down one of the bit-lines through its NMOS transistors M2 and M5. This considerably reduces the size of the pull-up PMOS transistors, M1 and M4, whose sole purpose is to restore the internal logic levels of the bit-cell and to prevent data corruption during reads.

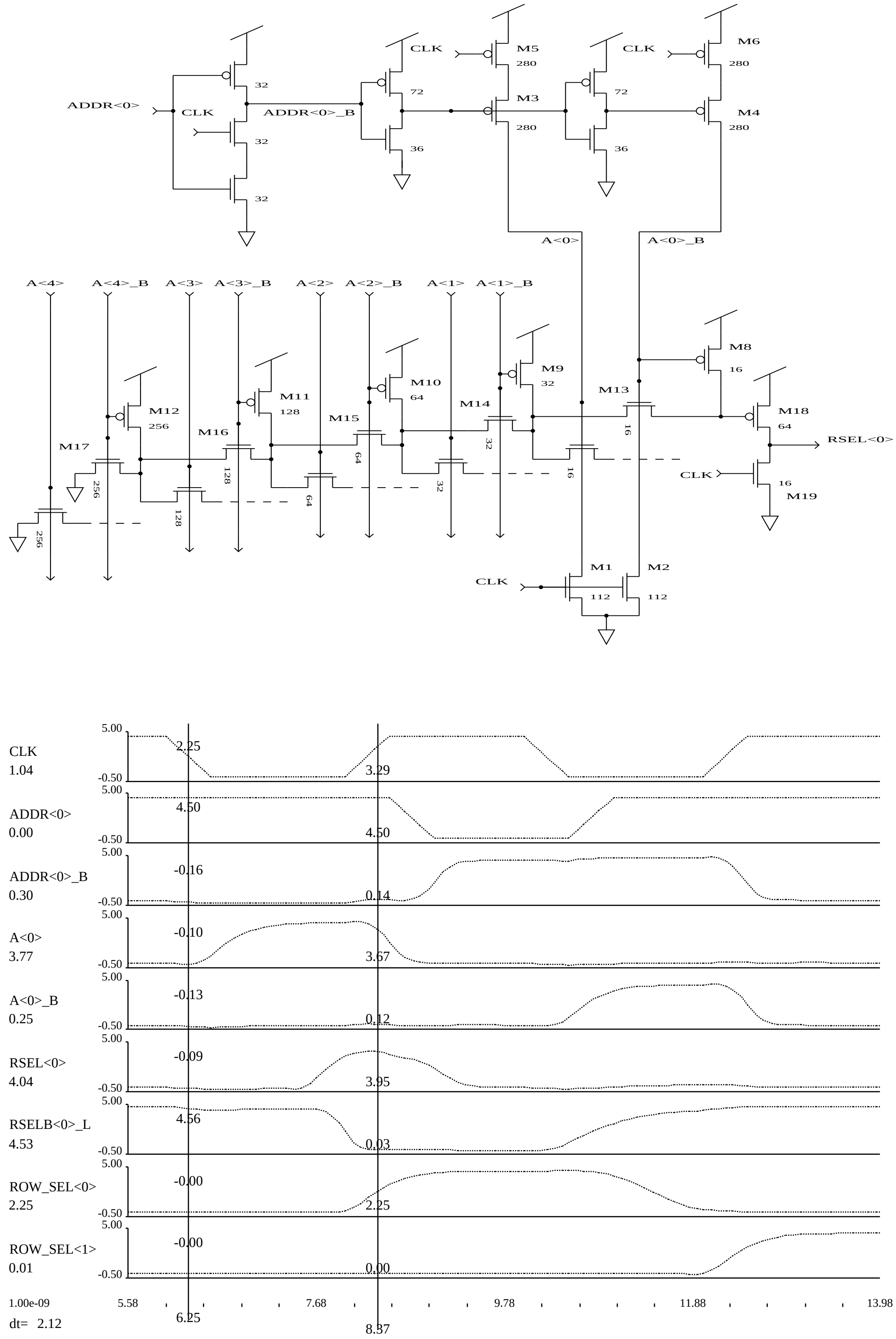


Figure 3.12: Register File Decoder Circuit and Simulation

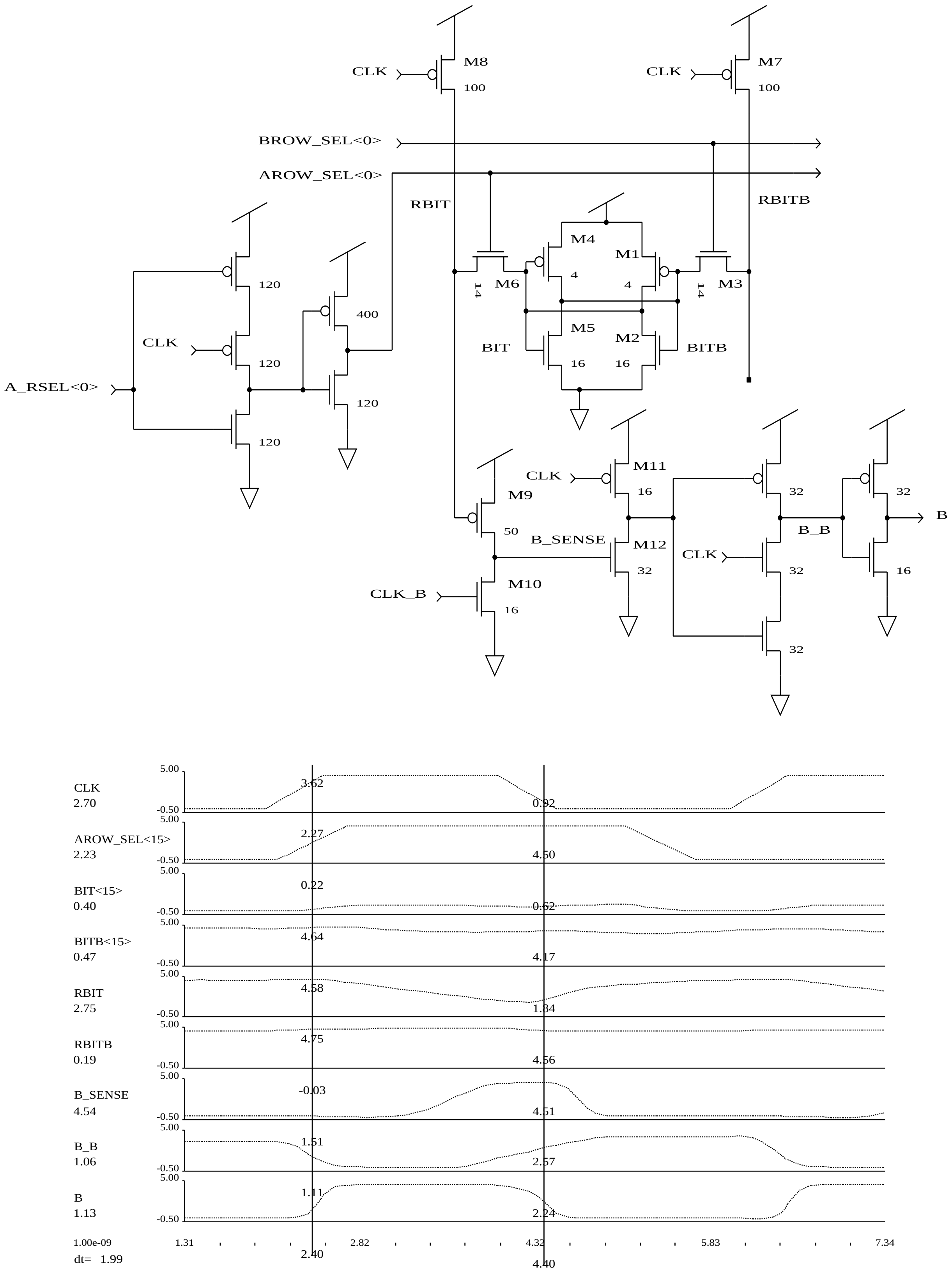


Figure 3.13: Register File Read Circuitry and Simulation

The sense circuitry consists of the two precharged inverters M9/M10 and M11/M12. They precharge on the low phase of the clock. During the high phase, the read bit-line is pulled down and its edge is sped up by the two inverters acting as single-sided sense amplifier. The data sensed is latched on the falling edge of the clock.

The simulation on Figure 3.13 shows a single-sided read on a bit-cell. Because the bit-lines `RBIT` and `RBITB` are both precharged high and are complementary to one another, only one bit-line is pulled down at any one time, to at most 1.54V. By not pulling all the way down to 0V, we significantly reduce the power consumption of the register file. The sense amplifier turns on as soon as `RBIT` is below 3.0V and the value on the bit-line is fully sensed 1.21ns after `ROW_SEL<0>` turns on or on the rising edge of the clock, whichever comes first. The data is latched after 1.92ns. The critical path occurs when reading a zero on port B where an inverter is needed after the n-latch. The data after the inverter is valid after 2.14ns, once again stealing 0.14ns from the next cycle.

Because the ratio of the NMOS access transistors to the cross-coupled inverters of the bit-cell was sized carefully, the effect of the fight between the precharged bit-line is limited on single sided read. The limited effect can be observed on `BIT` and `BITB`. `BITB` is lowered to 3.82V and `BIT` increases to 0.43V during the read, but the cell does not switch.

The Write Circuitry

To write a bit-cell, the write bit-lines `WBIT` and `WBITB` must be driven differentially with the `WROW_SEL` line enabled as is shown on Figure 3.14. The differential write driver is made of the transistors pairs M8/M10 and M7/M9 and the ground switch M11. The write data `W` must be valid before the end of the low phase so that inputs to M9 and M10 of the bit-line drivers are stable by the beginning of the high phase. It is important that the data is latched on the rising edge of the clock to prevent the gate of M9 and M10 from accidentally changing when driving the bit-lines and writing erroneous data into the bit-cell. The write bit-lines are precharged high during the low phase and the differential driver only pulls down one during the high phase. By the end of the high phase, the new data has been written in the bit-cell through the two access devices M3 and M6.

The simulation on Figure 3.14 shows two successive write cycles into the same bit-cell, first with a 1 and then with a 0. The differential write bit-lines `WBIT` and `WBITB` take the entire low phase to precharge. Then, they are driven 0.21ns after the rising edge of the clock and the internal storage nodes of the cell, `BIT<15>` and `BITB<15>`, are written 0.67ns after the rising edge.

Power Consumption

The register file consumes an average power of 1.82 Watts as given by the circuit simulator CAzM 1.82W at $27°C$ and $Vdd = 5.0V$for 200 random test vectors.

The power consumption has two origins: the precharge and discharge of the bit-lines and the clock node every cycle. The read bit-lines do not switch to the rail voltage, but only by 3.5V. They account for about 250mW. On the other hand, the write bit-lines switch a full V_{dd} and account for 500mW. Finally the clock nodes account for about 700mW. The remainder is mostly in the decoders and address line drivers.

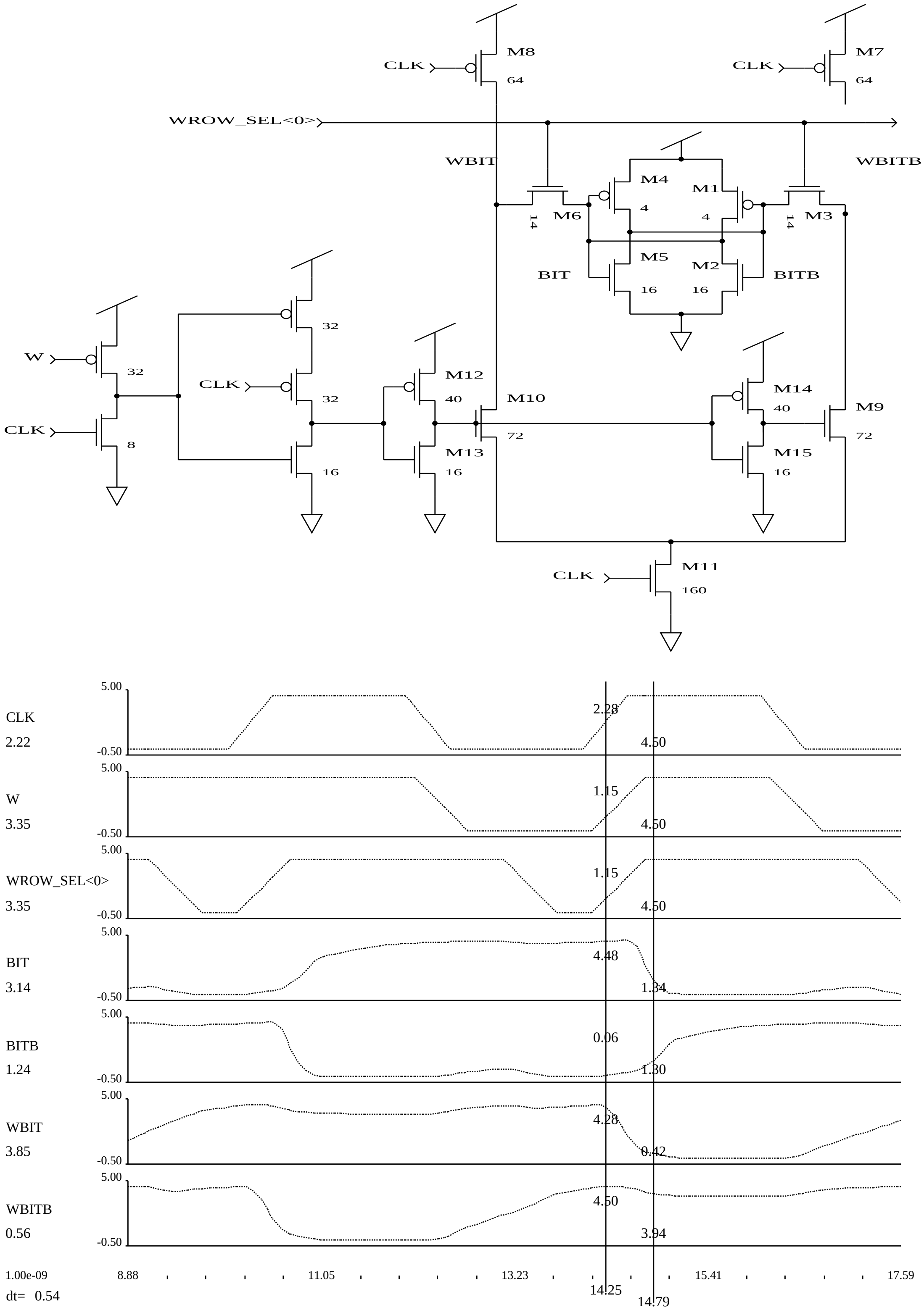


Figure 3.14: Register File Write Circuit and Simulation

3.6 Clock Distribution

The clock driver of the fast datapath is designed to drive a load of 350pF with a rise or fall time of 0.7ns at $27°C$ and $Vdd = 5.0V$. It minimizes clock skew by driving a single internal clock node from both sides of the datapath. The skew is also reduced by minimizing the internal resistance of the clock node, but at an increased cost in power consumption and area.

3.6.1 Description

The challenge in distributing the clock is to minimize the clock skew in the system. A brute force approach was chosen to generate the internal clock signal, `CLK`. Figure 3.15 shows how we implemented a chain of ratioed drivers to built up to the required final driver strength. Contrary to a distribution tree where the later stages of the driver are left unconnected, we connect together all the outputs of each stage to create equipotential nodes. The final stage of the driver is the internal clock signal, `CLK`. By making `CLK` as close a possible to an equipotential node, we eliminate the skew due loading mismatches on the branches of the tree.

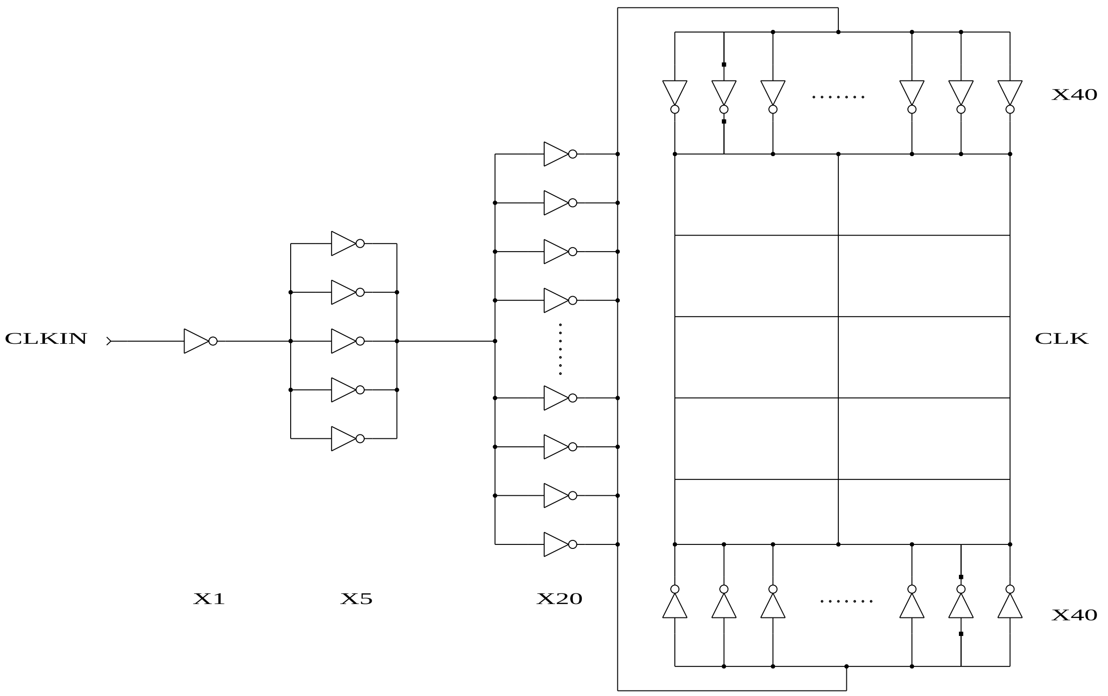


Figure 3.15: Clock Driver Block Diagram

The single phase design used in the fast datapath also simplifies the clock distribution. Instead of having to route one clock wire for each phase of the clock to all the blocks of the system, the single clock signal `CLK` needs to be distributed to all the blocks of the chip. An inverted clock is generated locally when needed to control precharged logic.

Nonetheless, the combination of a high clock load and large switching current still makes the design of the clock driver difficult. Although the single-phase latches used in the design present less clock load than a normal multi-phase latch, the precharge transistors of the dynamic logic increase greatly the clock load of the system. In the fast datapath, the total clock load that needs to be switched twice every cycle is about 350pF. At 250MHz, it creates an instantaneous switching current of approximately 3.8A. Therefore, the clock node resistance must be minimized to eliminate the unwanted RC delay with the clock load. As we decrease the clock wire resistance by making the physical node wider, we also add a distributed clock load due to the metal of the wire. The additional distributed capacitance is offset by making the driver larger and the RC skew is traded for increased power consumption.

Ultimately, some clock skew is present because of the finite propagation velocity of electrical signals over a silicon dioxide dielectric of 15cm/ns. The last stage of the clock driver was evenly split to the left and to the right of the datapath. The previous drivers are located on the top center of the datapath so that the delay to each half of the last stage is identical. However, because the final driver is distributed in the pads, the inputs to the last stage can be as far apart from each other as 4.0mm. Also, the logic connected to the any output of the clock driver can be as far as 3.0mm away. Hence, the total skew due to the limited propagation velocity over silicon dioxide is $\frac{0.70}{15}ns = 47ps$.

3.6.2 Bypass Capacitors

The circuit of Figure 3.16 shows the actual model used for simulating the clock driver. Because of the large instantaneous current, it is important to model the effect of the package and bonding inductance on power and ground. The increasingly larger driver sizes is reflected in the transistor size of the driver. With a $\frac{40000}{2}\lambda$ PMOS pull-up and a $\frac{18000}{2}\lambda$ NMOS pull-down, the last stage of the clock driver has an equivalent resistance of under 2Ω.

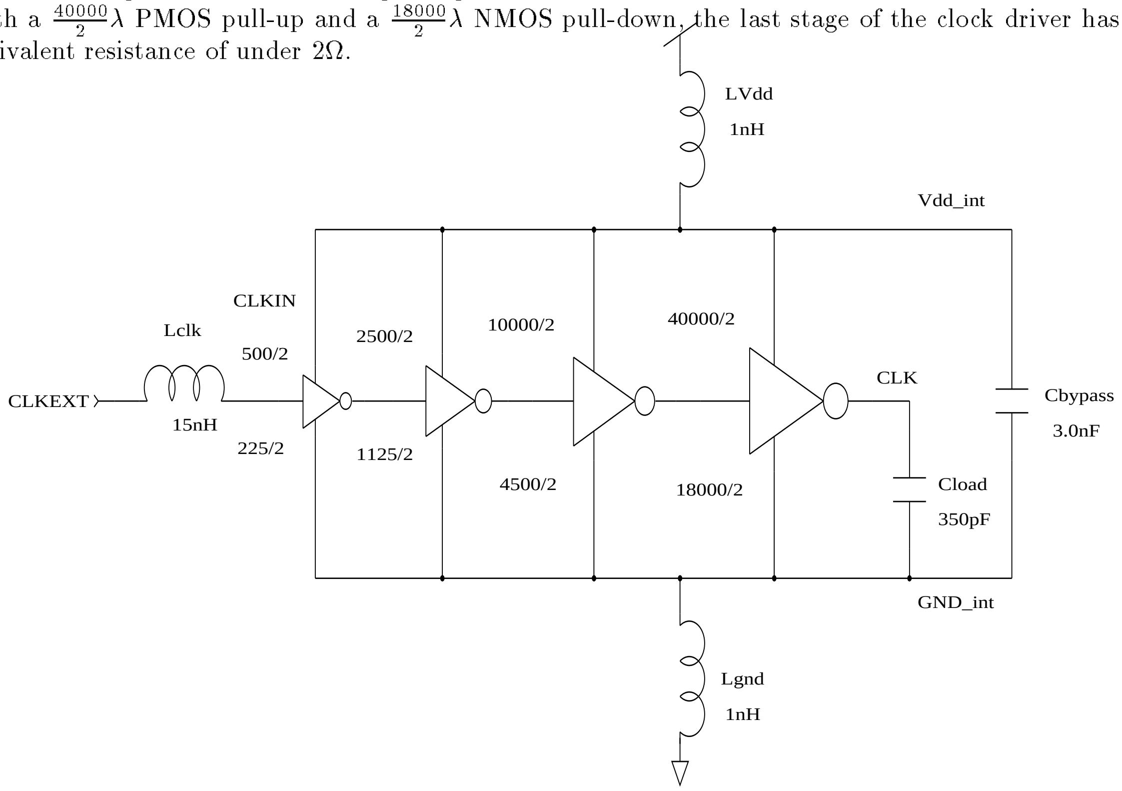


Figure 3.16: Clock Driver Circuit Diagram

Package parasitics inductance can result in non-functional chip when operating at high speed. Large currents switching through inductive pins induce large voltage bounces on the internal supply and ground nodes that can create latchup and other undesired effects. In order to switch 350pf of capacitance in 0.7ns, the clock driver requires a peak current of 3.8 Amps. Even though multiple pads and bonding wires reduce the pin and bonding inductance to below 1nH for V_{dd} and ground, the 1nH inductance will induce a voltage of about $L\frac{\partial i}{\partial t} = 5.4V$.

The simulation of Figure 3.17(A) was done for a single ended 0-2V 250MHz external clock input, with a 2.3V offset and a rise and fall time of 1.5ns. The signals represented are voltages for the successive clock driver stages. The last three signals are the current through the power parasitic inductor (Ivdd), through the ground parasitic inductor (Ignd) and the internal power voltage (VDDint). The HSPICE simulation at $27°C$ and $Vdd = 5.0V$shows clearly that the current going through the power and ground pins is limited to 1.65A by the parasitic inductors Lvdd and Lgnd. This results in large bounces on the internal clock nodes, especially the **CLK** node. The internal power node also bounces between 10 and 1.4 volts.

The solution to this problem is to add on-chip bypass capacitors to limit the large current transient through the parasitic inductors. Large on-chip capacitors can be made by using the gate-oxide capacitance of MOS transistors. The capacitors are built by connecting the sources and drains of NMOS transistors to ground and their gates to the power rail or connecting those of PMOS transistors to power and their gates to ground. The capacitors can fit under power rails and do not requires any extra space. Typical sizes used were $235 \times 80\lambda^2$, mostly limited by the maximum distance to well contacts and polysilicon resistance of the gate. Density of up to 0.9nF/mm was obtained.

To a first approximation, limiting the power and ground rail bounce to 10% or 0.5V requires a bypass capacitor of:

$$C_{bypass} = \frac{5 - 0.5}{0.5} C_{load} = 3.2nF$$

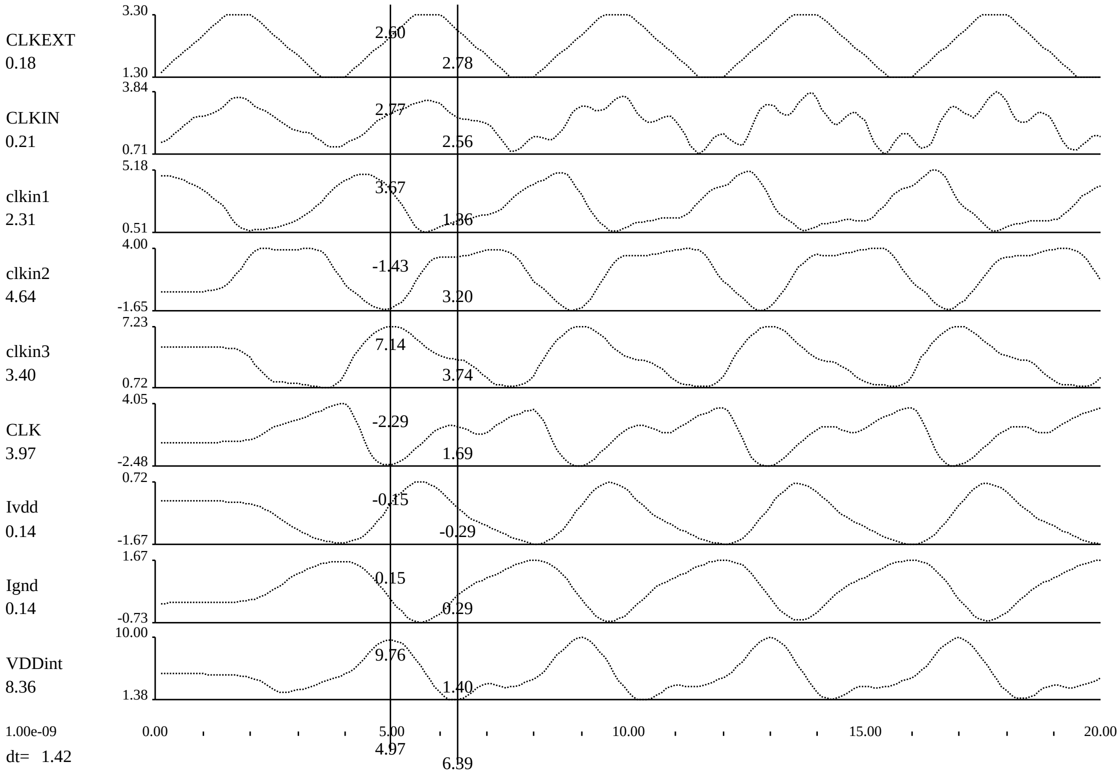
At no additional area cost, we were able to fit 3.1nF under the power and ground rail. The simulation of Figure 3.17(B) shows that the **CLK** node is now clean and its bounce is limited to -0.30V and +0.10V. A side effect of the bypass capacitor is the low frequency introduced on the power node. The $L_{vdd/ground}C_{bypass}$ pair creates a low resonance oscillation with period

$$p_{osc} = 2\pi\sqrt{(L_{vdd} + L_{ground})C_{bypass}} = 2\pi\sqrt{2nH \times 3.1nF} = 16ns.$$

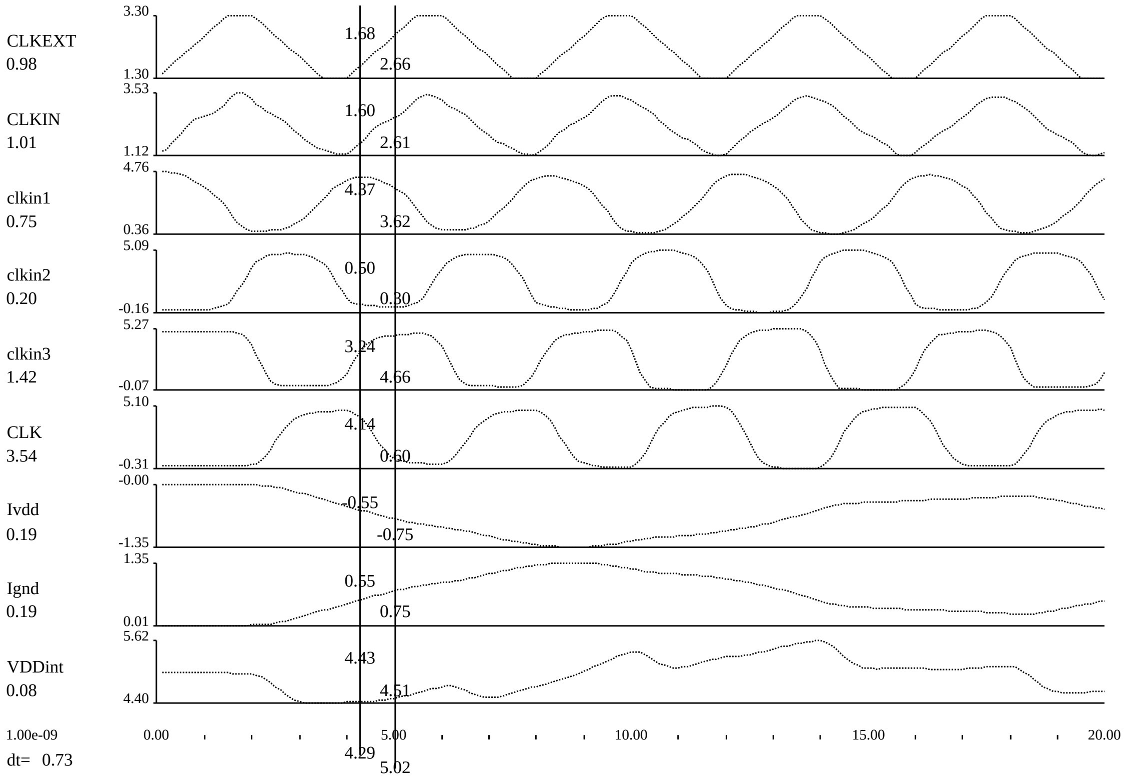
The current through L_{gnd} and L_{vdd} and the internal power rail voltage $VDDint$ clearly show the slow oscillation between 4.40 and 5.60V. In reality, those effects should be dampened by the resistance not taken into account in the simulation and should not affect the proper operation of the circuit.

3.6.3 Power Consumption

The power consumption of the clock driver turns out to be a significant part of the chip power consumption. At $27°C$ and $Vdd = 5.0V$, the unloaded clock driver consumes 3.61 Watts. That is twice the power consumption of the register file and one and a half times that of the adder. The loaded clock drivers burns 5.86 Watts. It is interesting to notice that 2/3 of the power consumption of the clock driver is used just to build up to the required driving strength. More aggressive sizing of the clock driver could have been done in the first stages to cut down power consumption, but that would have resulted in slower rising and falling edges in the last two stages, which was not acceptable.



(A)



(B)

Figure 3.17: Clock Driver Simulation

3.7 The PLA based Controller

A 250MHz PLA based controller was designed to control the operation of the datapath. The PLA was simulated for up to 16 inputs, 64 minterms and 32 outputs, although fewer are used in the fast datapath. Eight different test programs are built in to exercise all the data signal paths of the chip.

3.7.1 Why a PLA?

Testing and monitoring the operation of the fast datapath at high speed was not going to be an easy task. First, 20 control signals needed to change every cycle, including the register addresses and the bus enable signals. A sufficiently fast tester was not available to control these signals externally so we had to resort to an on-chip solution.

The difficulty in designing a controller came from the tradeoff between the flexibility of programming the controller and the ease of design. Two approaches were considered. First we thought of using a RAM based controller where control signal sequences would be stored at slower speed from a tester. Then, the program stored would execute at high speed and the results could be output at low speed. Unfortunately, this controller is sufficiently complex to run at 250MHz that we felt that the risks involved might jeopardize the rest of the chip.

Instead, using a synchronous PLA for the controller presented the advantage of being a simpler design, although it would not be possible to change the test programs. The drawback of having hardwired programs was overcome by an iterative process in which each new program was programmed in the PLA and simulated. The programming was done using the logic synthesis program BDSYN and the logic optimization program ESPRESSO. A PLA generator was then written to take ESPRESSO output and generate a MAGIC layout.

3.7.2 Block Diagram Overview

The block diagram of Figure 3.18 outlines the components of the PLA. It consists of an AND plane that evaluates during the p-phase and an OR plane that evaluates during the n-phase.

The AND plane of the PLA can generate up to 64 minterms. It consists of row drivers, **and_driver**, that drive the ROM array, **and_plane**. The output of the array is sensed by **and_sense** and latched on the rising edge of the clock. The OR plane is symmetrical to the AND plane. It is made of another set of row drivers, **or_drv**, the ROM array, **or_plane**, and output sense amplifiers, **or_sense**. The outputs that go to the datapath controls are latched on the falling edge of the clock and the drivers, **line_drv**, provide the necessary strength to drive the signals inside the datapath. Because the PLA operates synchronously over two phases, it can easily be turned into a finite state machine without any extra logic by connecting some of its outputs back into its inputs.

The PLA implemented for the fast datapath has 10 inputs, 64 minterms and 26 outputs. It occupies a total area of $1900 \times 1630\lambda^2$. Six outputs are used as addresses and are fed back to the input to implement a finite state machine. The additional inputs to the PLA are generated externally and consists of a reset signal, `reset`, and a program identification number, `tid[2:0]`. A `done` signal is output by the PLA to indicate that a program has completed.

3.7.3 Circuit Overview

Figure 3.19 shows the circuits used in the PLA. Except for device sizing, the circuit is symmetrical around the p-latch. Both the AND plane and the OR plane are made of a NOR gate with the proper inversion on the inputs or the outputs.

In the AND block, M1/M2/M3/M4/M5/M6 drive the inputs and their complementary values into the ROM array. The row select line must be stable by the beginning of the low phase. The ROM array of the AND plane is made of transistors M7/M8/M9 and M10. It is precharged high by M9 during the high phase of the clock. It is programmed with M10, by either connecting or removing it. The parallel combination of several M10 transistors connected to different row select line makes a precharged NOR gate that evaluates during the low phase. Because the NAND gate is made of small pull-down transistors to get a compact design, the output of the ROM array is sensed by the precharged inverter M11/M12

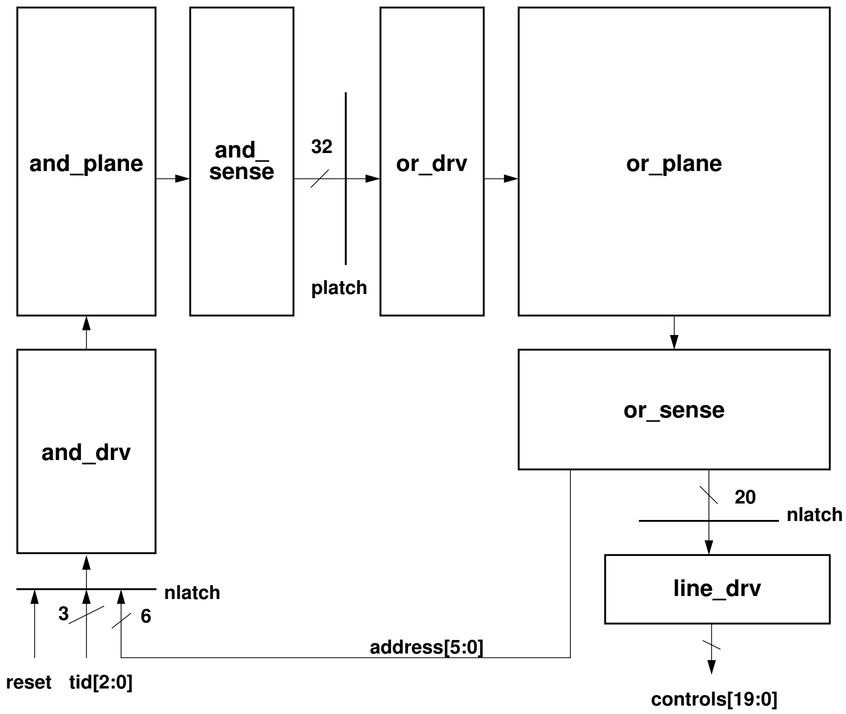


Figure 3.18: Controller Block Diagram

and the value is latched on the rising edge of the clock. However, the output must be stable before the rising edge so that the driver of the following stage is stable when the evaluation of the OR plane begins. Transistor M8 is required to fully precharge the ground switch node of the ROM array and prevents charge sharing. The timing of the OR block of the PLA is identical, but on the opposite phase. The ROM array of the OR block is programmed by transistor M13.

3.7.4 Operation

The PLA communicates with the outside world via a simple request/acknowledge protocol as is shown on the timing diagram of Figure 3.20. First, we select a 3-bit program identification number, `tid[2:0]` while the `reset` is asserted. The program starts executing as soon as the `reset` is de-asserted. Upon completion of the program, a `done` signal is asserted by the pla until the reset is asserted again. This simple protocol lets a slow-speed logic analyzer control the operation of the datapath while the chip is always clocked at high-speed. Although the operation of the external tester and the fast datapath are asynchronous, the input control signals are synchronized inside the chip to prevent logic glitches.

Eight different programs are implemented in the PLA. Their functionality is shown in Table 3.7.5. A typical sequence consists of loading register 1 and register 31 from the input shift register with program 2 and 3. The two registers can then be added together and their result stored in register 31 with program 4. Finally, the result can be checked by loading output shift register with the contents of register 31 with program 5 and serially shifting out the data. Every register location can be written or read using the shift program 1 that shifts all the register values up by one register. Finally, programs 6 and 7 are convenient for loading all the registers with patterns to check faults.

3.7.5 Power Consumption

The circuit simulator CAzM estimates an average power consumption of 0.63W for the PLA. This power consumption figure depends on the PLA programs run. It is typical of the program sequences run for

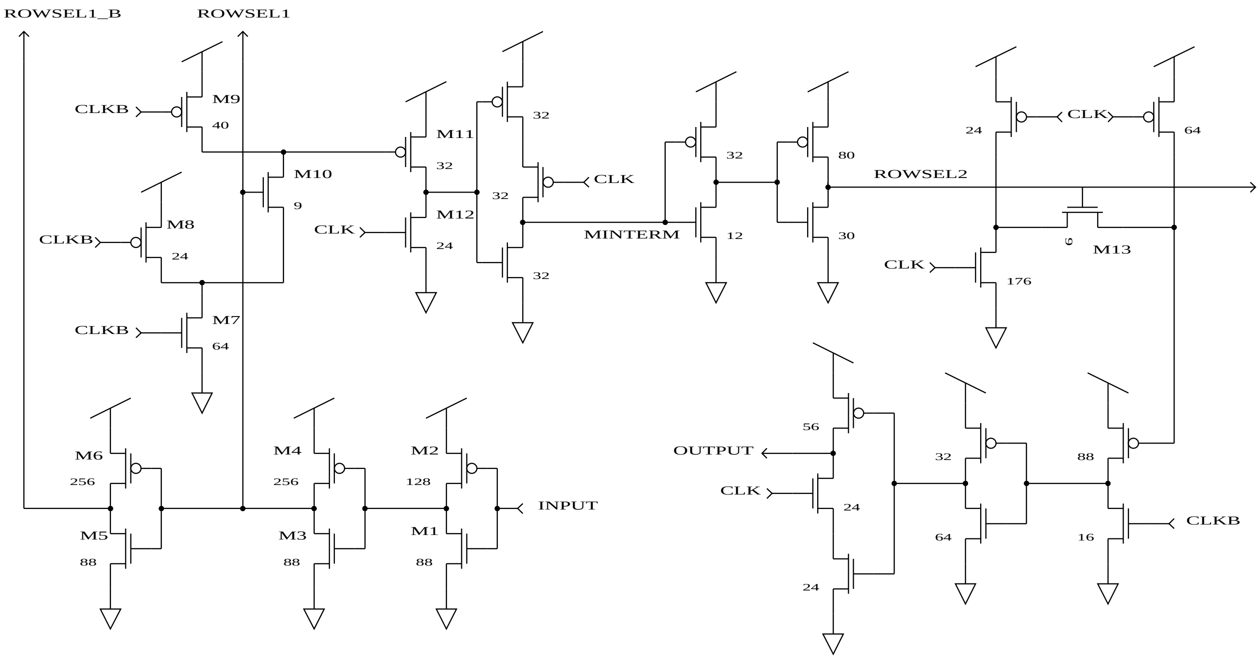


Figure 3.19: PLA Circuit Diagram

the fast datapath. Most of the power is dissipated in the row select drivers of the ROM arrays. Because the ROM array is precharged, it has a zero static power consumption.

TID	Mnemonic	Description	
0	shift	reg[n]	= reg[n-1]
1	add	reg[n]	= reg[(n-1) mod 32]+reg[(n-2) mod 32]
2	load1	reg[1]	= **buffer_in**
3	load31	reg[31]	= **buffer_in**
4	add_31_1	reg[31]	= reg[1] + reg[31]
5	out31	**buffer_out**	= reg[31]
6	load_even	reg[2n]	= **buffer_in**
7	load_odd	reg[2n+1]	= **buffer_in**

Table 3.3: Controller Programs

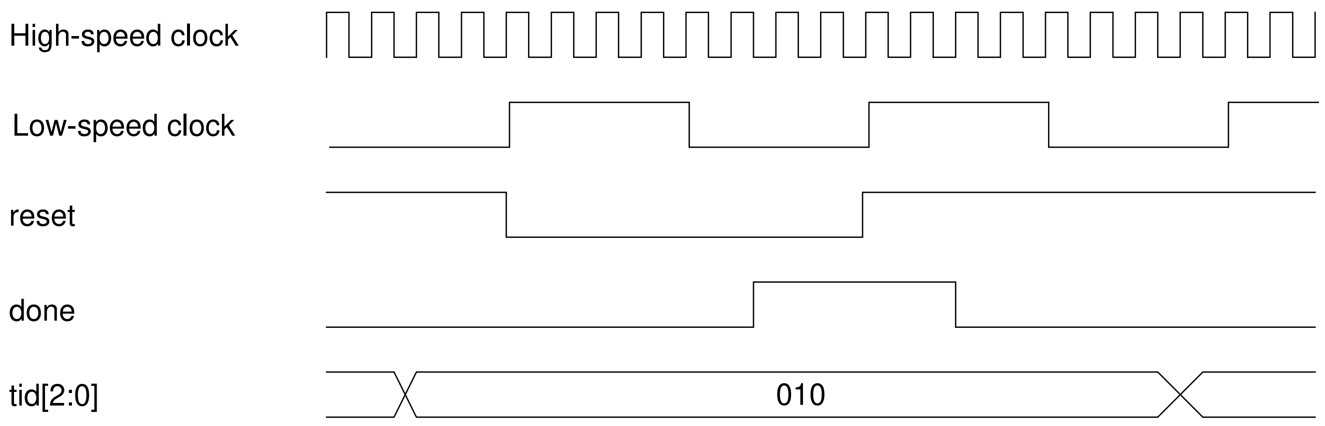


Figure 3.20: PLA Timing Diagram

Chapter 4

Results and Conclusions

4.1 Testing

4.1.1 Setup

To test the fast datapath, we fabricated a custom 2-sided PC board. Bypass capacitances were added on the board between V_{dd} and ground to minimize the effect of the inductance of the power supply wires. The chip was powered with an HP 6.0 Volt 6.0 Ampere power supply. We attached a heat sink on top of the package and used a small fan to cool the chip. An HP 8082A 250MHz signal generator was used to generate the high-speed clock. A 0V to 5.0V clock signal with 50% duty cycle was fed from a terminated 50Ω transmission line cable connecting the clock generator directly to the clock input pin to minimize signal reflection. A Tektronix DAS 9100 was used to drive the program test identification signal and reset. It was also used to shift patterns in the input shift register and capture the data out of the output shift register and the done signal. External signal probing was done with a Tektronix 2465 300MHz oscilloscope.

4.1.2 First Silicon

The first silicon of the fast datapath was send for fabrication on December 4, 1991. It was not functional. In this first version, power for the logic and power for the clock driver were separated. Also, no on-chip bypass capacitance was present as we did not anticipate a problem with the bonding and package inductance. The testing of the chip revealed two fatal problems.

First, missing well contacts in the input and output shift registers created latchup in the core of the datapath. Since the shift registers were the only way to get data in and out of the core, we could not assess any further the functionality of the core. However, we were able to test separately the clock driver by grounding the power and ground pins of the core eliminate latchup. Internal fine point probing of the V_{dd} and ground buses showed voltage bounces of over 1.8V and a very degraded internal clock because of the bonding and package inductance.

4.1.3 Revision

After adding the missing well contacts and on-chip bypass capacitance, a revision of the fast datapath was taped out on April 29, 1992. Because the internal logic and the clock driver do not switch simultaneously, we decided that it would be better to connect together their power and ground wires to take advantage of the intrinsic bypass capacitance of the logic. This revision was successfully tested at 180MHz at $39°C$ and 5.0V power supply. The maximum operating frequency of the fast datapath is limited by the adder. Above 180MHz, it was determined that the last stage of the carry chain is first to fail. The register file functions up to 210MHz and the PLA-based controller up to 230MHz. The power consumption is 1.4A at 180MHz, including a 0.2A static power consumption. The fast datapath operates at power supply of 4.0V to 6.0V. Below 4.0V, the input and output shift registers failed to work because of an NMOS pass transistor circuit used. At 4.0V, the fast datapath still worked at 160MHz. Above 5.0V, no significant increase in speed was observed.

4.1.4 Problem Investigation

The debugging of the fast datapath consisted of investigating the cause of an abnormal 0.2A static power consumption and the lower than expected operating speed.

Static power consumption

The chip current consumption at 5.0V is a linear function of the operating frequency. The graph of Figure 4.1 clearly shows that the static component does not go away at higher frequency.

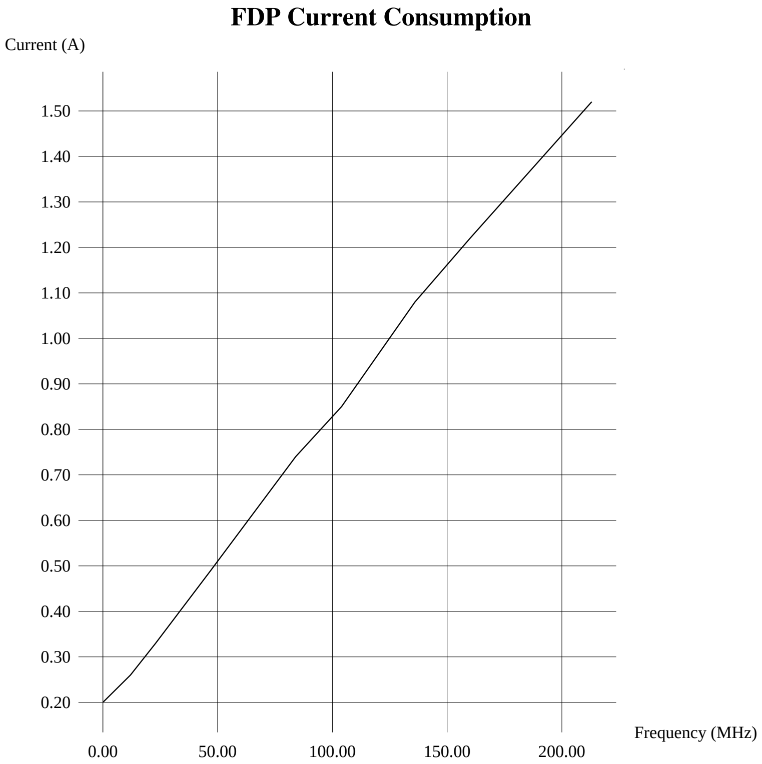


Figure 4.1: Fast datapath current consumption versus frequency.

A measurement of the static current versus the power supply voltage is shown on Figure 4.2. Two sets of measurements are done for an external clock at 5V and at 0V. The high level of the internal clock was the same as the power supply voltage. The characteristic of the current curve when the clock input is low suggests the presence of a reverse biased diode structure with a breakdown voltage a about 2V. We were not able to pinpoint the location of the structure, but we believe that in the last stage of the clock driver, a substrate contact too close to an active diffusion is at the source of the problem.

Limited Speed

Since the chip was designed to operate at 250MHz, we think that two factors limit the speed of the fast datapath to 180MHz. First we did not consider the transit time of electrons in the channel when designing the gate-oxide capacitors. The channel transit time is $\tau = \frac{L^2}{\mu V_{ds}}$. It is often overlooked when designing because minimum channel length devices are usually used. However, the gate oxide capacitors used are $48\mu m$ wide. For charge stored in the channel, the distance traveled is one half of this. Thus, the

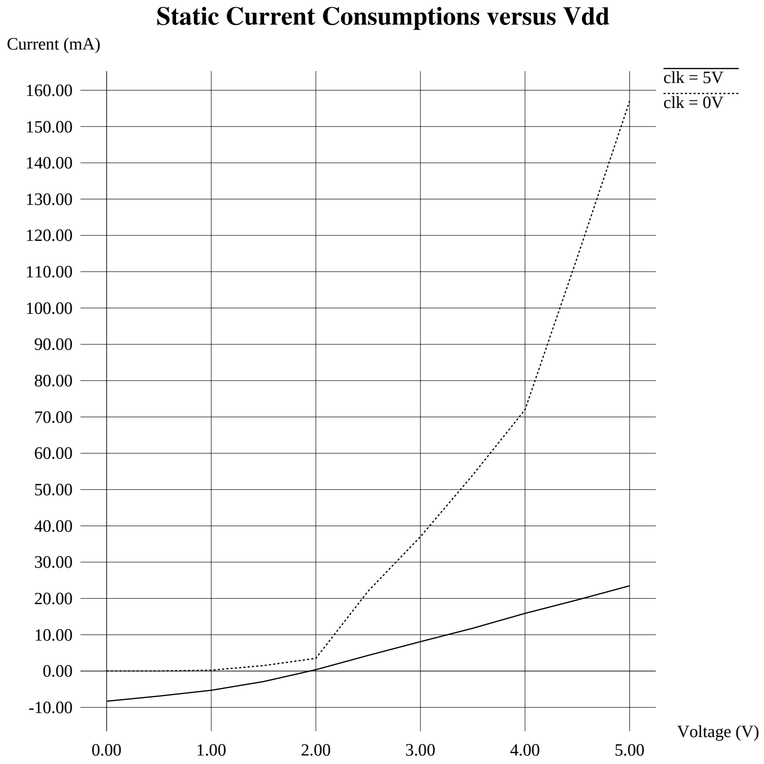


Figure 4.2: Fast datapath current static power consumption

channel transit time is approximately 1.5ns for NMOS gate-oxide capacitors and 3.0ns for PMOS gate-oxide capacitors. At high speed, this would limit the instantaneous current available from the charged capacitors. As a result, the instantaneous peak current through the pads is higher than expected and increased internal power and ground voltages bounce makes the logic fail.

A second potential source of speed limitation is that the capacitance extracted from the layout is lower than the actual value. The MAGIC extractor is known to have limitations when extracting capacitance. This would result in under-sizing the transistors and a lower functional speed.

4.2 Conclusions

The project described in this report successfully demonstrated that the combination of a single-phase clocking methodology, dynamic logic and custom layout can be used to built high-speed datapaths. A 64-bit integer datapath was designed and fabricated using these techniques and runs at 180MHz in a 1.2μm technology. This is about three times faster than the speed of commercial processors using similar processes.

The extra design effort involved consists mostly in controlling the noise and avoiding charge sharing. It is a time consuming process where SPICE level simulation and careful analysis are required. This is an area where software analysis tools will be required in the future to find potential faults of dynamic circuits in noisy environment.

We also showed that packaging is a great concern in high-speed circuits. The large pin inductance of conventional packages like PGAs is problematic when used in a high switching current environment, and on-chip bypass capacitance must be used to minimize bounce. Gate oxide capacitance can be used to make on-chip capacitors, but because gate-oxide has a low electric breakdown voltage and does not make efficient use of the die area, a combination of better packages and more capacitive structures will be required for that type of circuit in the future.

A shortcoming of this project is that we were not able to investigate the operation of the fast datapath at power supply below 4.0V to observe the tradeoff between operating speed, noise immunity and power supply voltage. As we decrease the power supply voltage, the power consumption of the circuit decreases with the square of the voltage. This is of interest because the slowdown is not a linear function of the supply voltage because of the velocity of the electrons in a minimum channel length MOS is saturated at 5V and is not directly proportional to the electric field applied across the channel. Also, observing noise immunity of the circuit to supply voltage would have been useful in understanding how the type of circuits used will scale in the future when we start using power supply voltages of 3.3V and below.

An important lesson learned during the course of the project is the breakdown of power consumption in the different blocks of the chip. The clock driver consumes an overwhelming 40% of the power and over 62% when loaded with all the latches and precharge transistors of the chip. Looking back at the design, we realize that we have used precharged logic almost everywhere. This was a mistake because it quickly increases the clock load. Instead, we should have used static logic whenever timing requirements allowed it. Synchronous designs also have the problem of large current induced on the edge of the clock. Static circuitry would have helped by distributing the computation over time, thus reducing the instantaneous current.

The high-speed clock generation was not addressed in this project and this somewhat impeded the testing. Distributing an external 5V 250MHz clock signal is not a viable alternative in high-speed systems. This solution was adopted here for simplicity but turned out to be a bad choice when testing. The 250MHz clock signal was difficult to generate and careful impedance matching had to be done on the clock input to minimize signal reflection. Instead, we believe that a low voltage differential clock signal would have been better. Another alternative would have been to generate a low-speed external clock and have an on-chip phase locked loop multiply the clock to the desired frequency.

Nonetheless, given the success of single-phase clocking in our design, we also believe the design could have been more aggressive by putting logic in the latches. We were a little hesitant to do this given the novelty of the clocking methodology, but we could have increased the performance of the chip by doing so. Actually, we believe that the adder could have been implemented in a single cycle this way. Although this is a test chip, a one and a half cycle adder is a penalty for most integer datapaths and should be avoided.

In view of these results, it becomes clear that future research on high-speed design will have to focus on reducing the clock power consumption. Also, further work is required to investigate how these techniques will scale as we go to smaller geometry and lower power supply voltage.

Nonetheless, one project in our group has successfully used these techniques to fabricate a 50MHz microprocessor [KIA+91].

4.3 Acknowledgments

First, I would like to thank the very special people that supported me and encouraged me all these years: Pierre, Elisabeth, Muriel and Cynthia. This is also to you, Danièle.

The ideas for this work and many of the preliminary circuit designs originated in a group effort during an advanced VLSI design course here at Berkeley in Spring 1991. In particular, I would like to thank Krste Asanović and Stuart Kleinfelder for their early design of the register file, Paul deDood and Brian Kingsbury for their contribution to the design of the adder, John Lazzaro for his contagious chip design paranoia that made me a lot more cautious and to Jim Beck for his chip testing skills. I would also like to thank Professor Robert Brodersen and Lars Thon for helping with the fabrication of the chip.

Last but not least, I would like this express my gratitude to my research adviser, Professor John Wawrzynek, for his technical guidance and financial support throughout this project.

This work was funded by the National Science Foundation through the Presidential Young Investigator (PYI) award, MIP-8958568 and by the National Foundation Infrastructure Grant CDA-8722788. Fabrication of the chip was supported in part by a grant from DARPA.

Bibliography

[AS90] Morteza Afghahi and Christer Svensson. A Unified Single-phase Clocking Scheme for VLSI Systems. *IEEE Journal of Solid-State Circuits*, pages 225–233, February 1990.

[Bak90] H. B. Bakoglu. *Circuits, Interconnections and Packaging for VLSI.* Addison Wesley Publishing Company, 1990.

[DdD89] Alvin Despain and Paul de Dood. Unpublished notes on fast arithmetic. January 1989.

[Dob92] Daniel Dobberpuhl. A 200MHz 64b Dual-Issue CMOS Microprocessor. In *IEEE International Solid-State Circuits Conference*, pages 106–107, February 1992.

[ER90] Donal J. Erman and Donald J. Rose. *CAzM, Circuit Analyzer with Macromodeling.* MCNC Center for Microelectronics, June 1990.

[GD85] Lance A. Glasser and Daniel W. Dobberpuhl. *The Design and Analysis of VLSI Circuits.* Addison Wesley Publishing Company, 1985.

[HCL87] Tackdon Han, David A. Carlson, and Steven P. Levitan. Fast Area Efficient VLSI Adders. In *IEEE International Conference on Computer Design*, pages 418–422, October 1987.

[HP90] John L. Hennessy and David A. Patterson. *Computer Architechture, A Quantitative Approach.* Morgan Kaoufman Publisher, 1990.

[KIA+91] Brian Kingsbury, Bertrand Irissou, Krste Asanovic, John Wawrzynek, and Nelson Morgan. Recent Work in VLSI Elements for Digital Implementations of Neural Networks. Technical report, International Computer Science Institute, December 1991. TR-91-074.

[LS87] Richard F. Lyon and Richard R. Schediwy. CMOS Static Memory with a New Four-Transistor Memory Cell. In *Advanced Research in VLSI, Proceedings of the 1987 Stanford Conference*, pages 111–132, 1987.

[Met90] Meta-Software. *HSPICE User's Manual*, 1990.

[SMHO86] W.S. Scott, R.N. Mayo, G. Hamichi, and J.K. Ousterhout. VLSI Tools: still more work by the original artists. Technical report, University of California at Berkeley, EECS Department, 1986. UCB/CSD 86/272.

[VIE89] VIEWLogic Systems, Inc. *Schematic User's Guide*, December 1989.

[WS80] Robert W. Won and Carlo H. Sequin. A guide to LSI Implementation. Technical report, Palo Alto Research Center, XEROX, 1980.

[YS89] Jiren Yuan and Christer Svensson. High-speed CMOS Circuit Techniques. *IEEE Journal of Solid-State Circuits*, pages 62–70, February 1989.