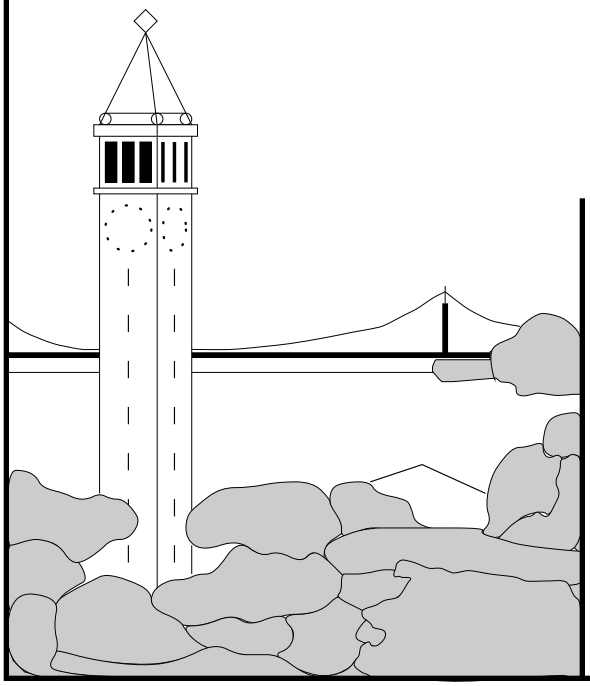
A Low Power 200 MHz Multiported Register File for the Vector-IRAM chip

Iakovos Mavroidis



Report No. UCB/CSD-01-1145

May 2001

Computer Science Division (EECS)
University of California
Berkeley, California 94720

A Low Power 200 MHz Multiported Register File for the Vector-IRAM chip

by Iakovos Mavroidis

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee:

Professor David A. Patterson
Research Advisor

(Date)

* * * * * * *

Professor Jan M. Rabaey
Second Reader

(Date)

A Low Power 200 MHz Multiported Register File for the Vector-IRAM chip

Iakovos Mavroidis

M.S. Report

Abstract

Vector IRAM integrates vector processing with embedded DRAM on a single chip to provide high multimedia performance at low energy cost. This report presents the design and the implementation of the VIRAM Vector Register File. Our design successfully faces many challenges such as the need for speed, low power consumption, compact design and multi-ported access. Using a 0.18μm technology and a 1.3 Volts supply voltage, it operates at 200 MHz, consumes an average power of 330 mW and 8 mm^2 of area, and provides eight read and three write ports. A number of available CAD tools were used, including layout tools from CADENCE, extraction tools from Avant!, hspice, timemill and powermill. This report gives emphasis on implementation issues and evaluates the performance and power consumption of our design.

Contents

1 Introduction

Vector machines exploit data parallelism by executing multiple homogeneous operations on an array of elements within a single vector instruction at the same time. Since one instruction describes a whole set of operations, the requirement on the instruction fetch bandwidth is greatly reduced. Moreover, eliminating off-chip accesses, embedded DRAM technology provides high memory bandwidth at low power. VIRAM [Koz99] provides high multimedia performance with low energy consumption by integrating vector processing with embedded DRAM technology. Both high performance multimedia processing and low power consumption make VIRAM an ideal candidate for future PDA-like devices.

VIRAM supports 3.2 GOPS (single-precision) peak performance and 25.6 GByte/s peak memory bandwidth. The target power consumption for the vector unit and the memory system is 2 Watts, achieved by using a modest clock frequency of 200MHz at a 1.3V supply voltage.

There are four vector functional units in VIRAM: two arithmetic, one flag processing, and one load-store. All vector functional units have multiple parallel datapaths to process multiple vector elements per cycle. The vector instruction set has been designed to support a clustered implementation of the vector unit. VIRAM contains four hardware clusters, referenced as *lanes* (see Figure 1). All lanes are identical and are given the same control signals. Figure 1 shows the floorplan of the VIRAM chip, which consists of a vector unit, a scalar core, an FPU, eight DRAM banks, and four vector lanes. Each vector lane contains two 64-bit integer execution units, two floating point execution units, a flag register file and one fourth of the vector register file.

Vector architectures require large multiported register files since they concurrently perform operations on many registers. In particular, VIRAM vector register file has a total capacitance of 8 KBytes and contains 32 vector registers, each holding 32 64-bit elements. The vector register file is split into four parallel 64-bit slices, one for each of the four parallel lanes in the vector unit. It is partitioned so that each lane stores the register elements that are processed by the local datapaths. The elements of a vector register are partitioned among the lanes in a round-robin fashion. Each lane contains a total of 256 64-bit registers (32 vector registers with eight elements per vector register). Vector registers can be subdivided to hold 64 32-bit elements or 128 16-bit elements. For this reason, the vector register file supports write operations on 16-bit granularity with the use of write masks, as we will explain in Section 2.

The vector register file operates at 200 MHz, which is the clock frequency of the VIRAM chip. It occupies $2mm^2$ in each lane, or $8mm^2$ in total.

The VIRAM chip is scheduled for fabrication by IBM in summer 2001. It uses a state-

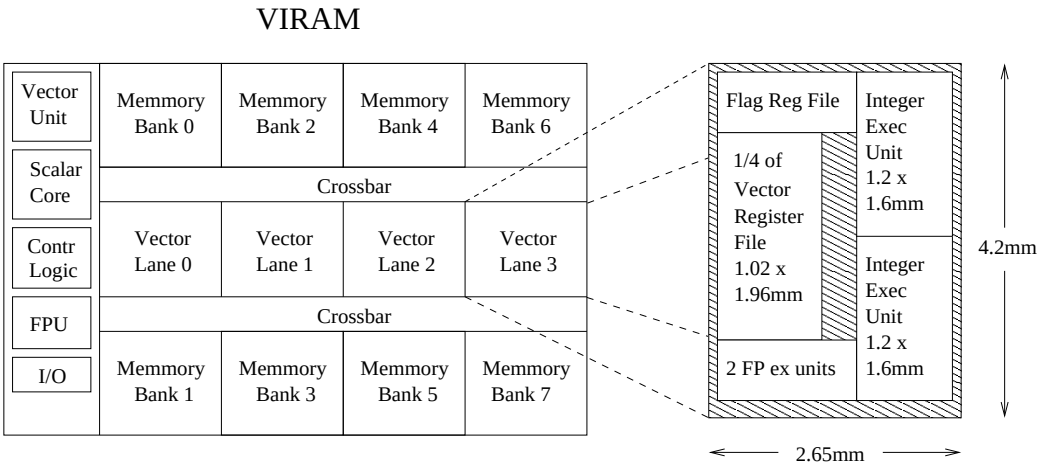


Figure 1: The floorplan of the VIRAM chip. Each vector lane contains one fourth of the vector register file which occupies $2mm^2$, or 18% of the lane area, and is accessed by the two Integer Execution Units that reside in the same vector lane, the two Crossbars, and the Control Logic.

of-the-art $0.18\mu m$ technology with six metal layers. The two integer execution units, the vector register file inside each vector lane, and the crossbars were designed using custom layout technology. The DRAM banks were provided as macro-cells by IBM, and the rest of the logic was designed using a standard cell library from IBM [IBM00]. The multiported nature of the register file and the small lane area resulted in routing many control and data signals on top of the register file. For this reason, we decided to design the register file using only the three bottom metal layers from the six available, and leave the top three metal layers for routing the control and data signals, the clock and the power supply.

The remainder of this report is organized as follows. Section 2 presents the architecture of the VIRAM vector register file. Section 3 explains the logic style, as well as the timing, of the vector register file. Section 4 discusses the implementation of the register file in detail. Section 5 shows the simulation and verification environment, and Section 6 evaluates the performance and measures the power consumption of the vector register file. Finally, Section 7 describes related work.

2 The Architecture

Table 1 shows the worst-case port requirements for the blocks that access the vector register file. As we see, a total of 8 read and 3 write ports are enough to eliminate any structural hazards that might result from port contention.

Block	Max number of read ports needed	Max number of write ports needed
Integer Exec Unit 1	3	1
Integer Exec Unit 2	3	1
Crossbars & Control Logic	2	1

Table 1: Worst-case port requirements for the blocks that access the register file.

However, a preliminary design of the memory cell revealed that a 256×64 register file with 8 read and 3 write ports would require approximately $3mm^2$ of area, and would be very difficult to fit in the vector lane. To make matters worse, the huge number of read ports attached to each read bit line (8 read ports per word times 256 words) would result in increased bit line capacitance and dramatically impact the timing and power budget. Splitting the register file into two 128×64 banks, with each bank providing 8 read and 3 write ports, could potentially improve the speed and power consumption by decreasing the bit line capacitance, but would result in more area which is unacceptable.

For this reason, we decided to split the register file into two 128×64 banks, where each bank would only provide half the number of read ports, thus 4 read ports and 3 write ports. If more than 4 read accesses contend for the same bank a structural hazard will have to occur, but fortunately this does not seem to happen often.

Splitting the register file into two 128×64 banks, greatly improves all aspects of the design. The smaller number of read ports results in dramatic reduction of the memory cell area by approximately 40%. Also, the reduced bit line capacitance, since each bit line is connected to 128 memory cells instead of 256, results in reduced power consumption and increased speed.

2.1 Block Diagram

Figure 2 shows the block diagram of the register file inside a lane. It consists of two 128×64 banks, one *read address mapping* block that routes the external read addresses to the correct bank ports, one *read data mapping* block that routes the data read from the

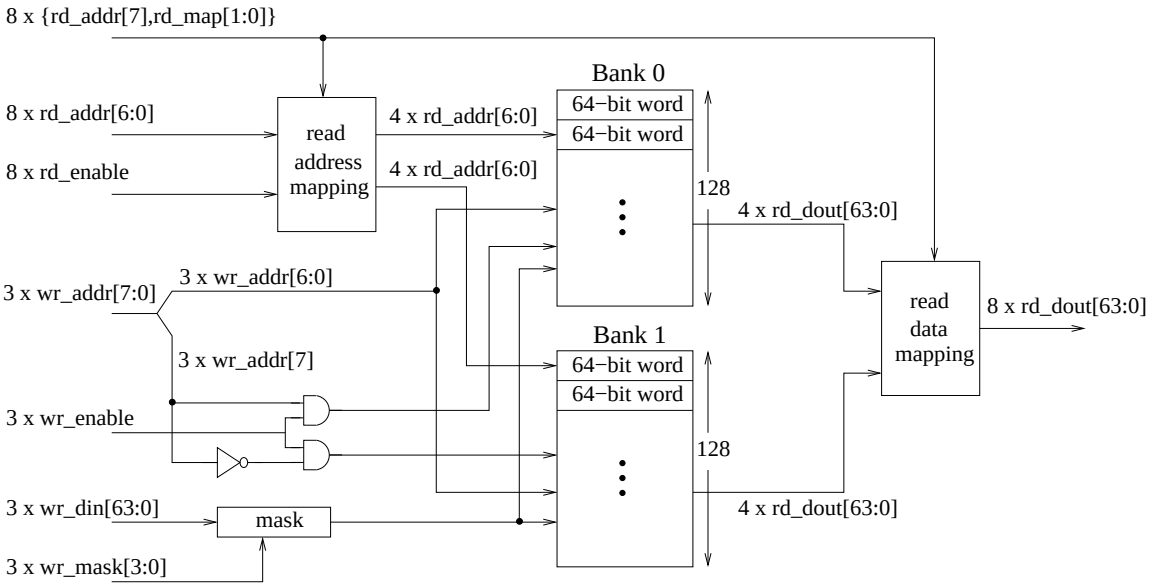


Figure 2: The block diagram of the register file. It provides 8 read and 3 write ports, using two banks with 4 read and 3 write ports each, thus allowing at most 4 read accesses to contend for the same bank. Two mapping blocks are used to route the read addresses from the external ports to the correct bank ports, and the read data from the banks to the correspoding external data buses.

banks to the correct external read data buses, seven address decoders per bank (not shown), and some write-mask logic to allow 16-bit write granularity.

We will next briefly explain the functionality of these components by describing how a read and a write access operate.

Read Access For a read access to occur, the 8-bit address $rd\_addr[7:0]$ is placed on one of the 8 external read ports and the corresponding read enable signal is set. The most significant bit of this address specifies the bank that will be accessed. Since each bank provides 4 read ports, the address will have to be mapped to one of these ports. A 2-bit mapping signal per read port, $rd\_map[1:0]$, determines the port number for each address. When more than one addresses contend for the same bank, they will need to be mapped to different ports. The logic that selects the correct port for each address and provides the eight mapping signals, is external to the register file logic. This external logic will have to deal with the structural hazards when more than four accesses contend for the same bank and decide which accesses to stall. The *read address mapping* block of the register file assumes that there are no conflicts in the mapping provided, and will route each address to the corresponding bank port. The *read data mapping* block will do the reverse operation to map the data read from the banks to the corresponding external read data buses.

Write Access Since the number of external write ports is the same as the number

of write ports that each bank provides, there is no need for mapping blocks for the write accesses. The seven least significant bits of all three write addresses, $wr\_addr[6:0]$, are routed to each bank. The most significant bit of each address specifies the bank that will be accessed if the corresponding write enable signal is set. A 4-bit write mask per write port, $wr\_mask[3:0]$, will determine which 16-bit fields of the 64-bit write data, $wr\_din[63:0]$, will be written.

2.2 Datapaths for the read and write accesses

A vector register is subdivided in a big number of smaller elements, and spans 32 words in the register file. Each vector operation on such a register would require reading from or writing to all these register file words, and would thus need multiple cycles to execute. Chaining, the extension of forwarding to vector architectures [HP96], allows overlapping the execution of dependent instructions, making it possible for two instructions to simultaneously operate on different elements of the same register. An instruction can start accessing the elements of a vector register, as soon as the previous instructions have finished accessing these elements. Depending on the type of accesses, chaining introduces three types of hazards for the vector elements: read after write, write after read and write after write.

The vector register file does its best to improve the performance for chaining. It supports one access per cycle for each read or write port, thus allowing up to four read (since each bank has four read ports) and one write accesses per cycle for the same word. Write-through capability allows a write and a read access to access the same word during the same cycle, in which case the read will get the new value.

Write Access Figure 3 shows the datapath for a write access. The write address and enable are available from the vector unit one cycle earlier than the write data and mask. The register file takes advantage of this by pipelining the access into two cycles: the address and enable signals are decoded in the first cycle, and the data is written in the second cycle. In fact, the data is written during the *first half* of the second cycle, making it possible for a read access on the same word to use the second half of the cycle to get the new value (write-through).

Read Access Figure 4 shows the datapath for a read access. Contrary to a write access, all inputs become available in the same cycle, and a read takes place in one cycle: the address is routed to the decoder of the correct bank port where it is decoded during the first half of the cycle, and the data is read from the memory array and routed to the correct external read data bus during the second half of the cycle. Negative edge-triggered registers and latches are used to split the cycle in two parts.

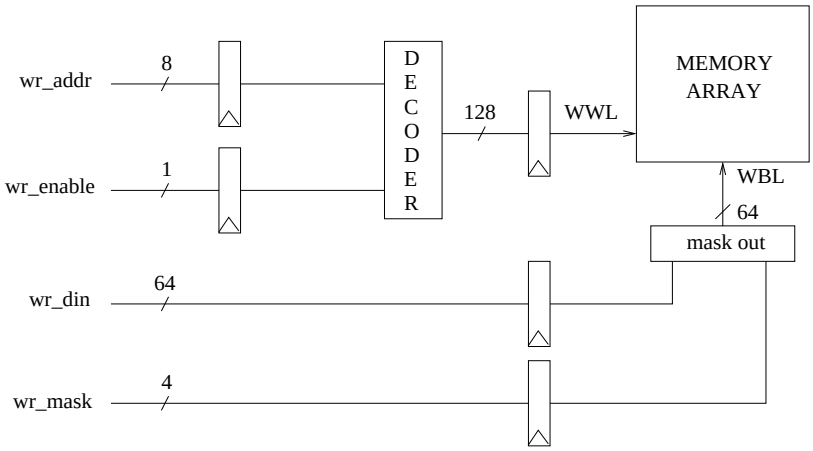


Figure 3: The datapath for a write access. A write is a pipelined two-cycle operation where the address is decoded during the first cycle, and the data is written during the first half of the second cycle.

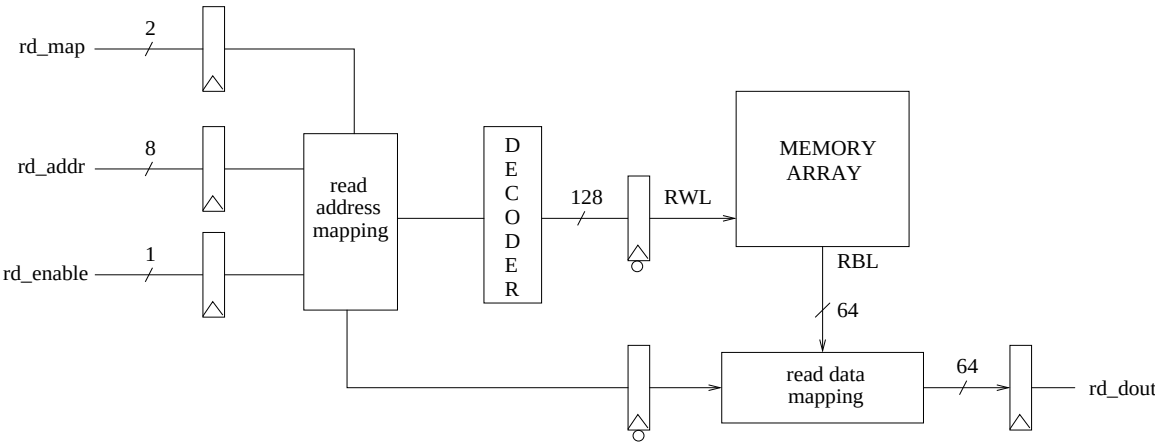


Figure 4: The datapath for a read access. The address is routed to the correct bank port and decoded during the first half of the cycle, and the data is read and routed to the correct external data bus during the second half of the cycle. Negative edge-triggered registers and latches split the path into two portions.

In order to facilitate the timing of the logic that interfaces to the register file, the inputs and outputs of all read and write ports are directly connected to or driven by positive edge-triggered registers.

3 Logic Style

3.1 Dynamic Logic in the Vector Register File

The periphery circuitry of the vector register file, including the mapping blocks and the decoders, was implemented in dynamic and pass-transistor logic styles. This choice allowed us to have a fast periphery circuitry, without compromising on the power requirements.

Dynamic logic uses a sequence of *precharge* and conditional *evaluation* phases [Kra82] to realize complex logic functions with fewer transistors than standard CMOS logic. The clock φ usually determines the boundaries of these two phases. The dynamic circuit concept results in simple, small and fast structures at the expense of a reduced robustness with regards to noise. One could argue that dynamic logic consumes more power than standard CMOS logic because it has higher switching activity ([Rab96], pages 234-40), and therefore it may not be appropriate for low power designs. However, a dynamic design can usually operate at a lower voltage supply than a standard CMOS design, causing the overall power to drop in a quadratic way. A fast logic, such as dynamic and pass-transistor [RWM93] logic styles, can adequately compensate for the loss of speed due to the lower voltage supply.

The read/write operations for each memory cell are also implemented in dynamic logic and therefore function in two phases. The read/write bit line is precharged during the precharge phase, and is discharged in the evaluation phase, if the read/written value is 0. The memory cell implementation is discussed in detail in Section 4.5.

The datapaths in Figures 3 and 4 are split by dynamic registers or latches into two portions. When the dynamic logic of a portion in a datapath evaluates or precharges, the dynamic logic of the other portion precharges or evaluates respectively. This timing is depicted in Figure 5 that shows the timing of each block.

As shown in Figure 5, the address and the enable signal of a write operation are received one cycle earlier than the input data. The address is decoded during the evaluation phase of the decoder and the new value is written in the first half of the next cycle. In parallel, the external read addresses are mapped to the internal read addresses and they are decoded. In the next half cycle the value of a word is read and mapped to an external output data bus. Figure 5 also shows that a word can be written in the first half of a cycle and read in the second half of the same cycle.

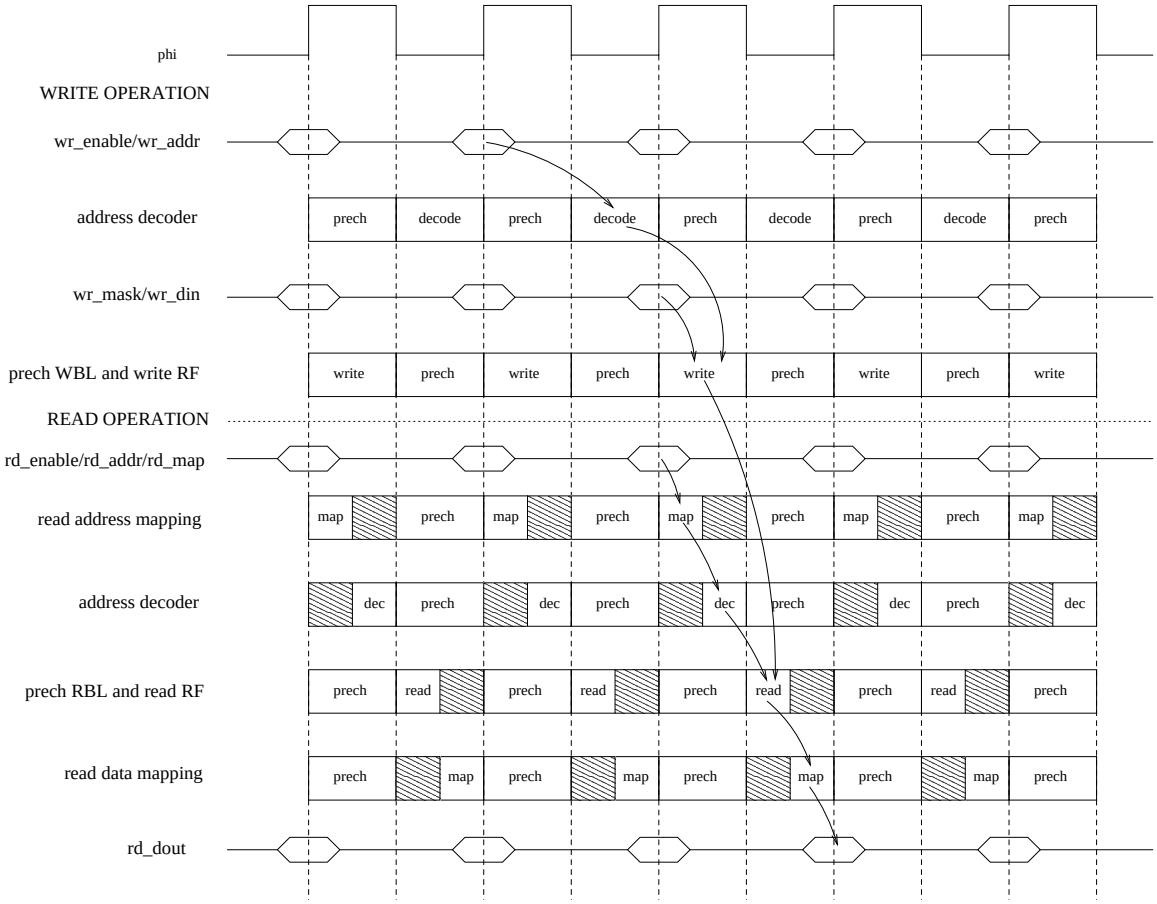


Figure 5: The timing diagram of the write and read operations. Dynamic logic operates in a sequence of precharge and evaluation phases. A cell can be written and read in the same cycle as shown in the Figure.

3.2 Dynamic Storage Elements in the Vector Register File

Storage in static sequential circuits relies on the concept that a cross-coupled inverter pair produces a bistable element and can thus be used to memorize binary values. The major disadvantage of a static flip-flop or latch over their dynamic counterparts is complexity. Its large size, as well as the large clock capacitance, becomes a dominant and restrictive feature. Storage in dynamic sequential circuits relies on the concept that capacitance can act as a memory element as well. The absence of charge denotes a 0, while its presence stands for a stored 1. Dynamic flip-flops and latches are fairly small, but they are sensitive to noise and leakage currents. The vector register file uses dynamic registers and latches because of their simplicity and small clock capacitance.

Dynamic CMOS flip-flops and latches can be implemented in many different ways.

Two-phase dynamic flip-flops and latches operate using two clocks that should usually not overlap. Malfunction may occur when the clocks overlap, and therefore some constraints are usually imposed such as the even-inversion constraints between latches ([Rab96], pages 355-9) or small clocks overlap. Single-phase dynamic flip-flops and latches use a single clock. Compared to two-phase dynamic flip-flops and latches, the only disadvantage of this approach is an increased number of transistors per flip-flop or latch. On the other hand, this design has the advantage that virtually all constraints are removed. For this reason we used *single-phase dynamic edge-triggered flip-flops and latches* in the register file.

Different dynamic storage elements were used throughout the register file, depending on the timing requirements imposed by the inputs and outputs of each block. Registers were used to store the input and output data of the two banks, as well as the inputs of the write address decoders. The word lines are connected to latches, while the inputs of the two mapping blocks come from dynamic flip-flops that latch their inputs at the start of the evaluation phase and reset their outputs to either zero or one at the start of the precharge phase. The next section describes these special flip-flops in detail.

4 Design Components

4.1 Flip-Flops and Latches

Figure 6 shows the schematics of the flip-flops and latches that were used in the design of the register file. They are all dynamic and use a single clock phase.

The *charge-discharge* flip-flops, which we introduce here, combine the flip-flop functionality with the charging or discharging required in dynamic logic. They latch the input value to their output at the clock edge marking the beginning of the evaluation phase, and charge or discharge their output during the precharge or predischarge phase. For example, a *latch-discharge* flip-flop latches the input value at the positive edge of the clock, and discharges its output during $\bar{\phi}$. This kind of flip-flops was used to latch the addresses of all read and write ports ($8 \times rd\_addr[7:0], 3 \times wr\_addr[7:0]$), as well as the read mapping signals ($8 \times rd\_map[1:0]$), the write masks ($3 \times wr\_mask[3:0]$) and the enable signals ($8 \times rd\_enable, 3 \times wr\_enable$), resulting in a total of 127 *charge-discharge* flip-flops. Another 192 *true single-phase clocked logic* (TSPCL) flip-flops [YS89] per bank, were used to latch the write data inputs ($3 \times wr\_data[63:0]$).

All the input ports are latched using positive edge-triggered flip-flops. Similar to the input ports, all output ports (i.e., $8 \times rd\_dout[63:0]$) are also driven by positive edge-triggered flip-flops. Since these signals are internally produced by dynamic logic, however, and will thus be precharged to V_{DD} during $\bar{\phi}$, we were able to use the simpler single-phase

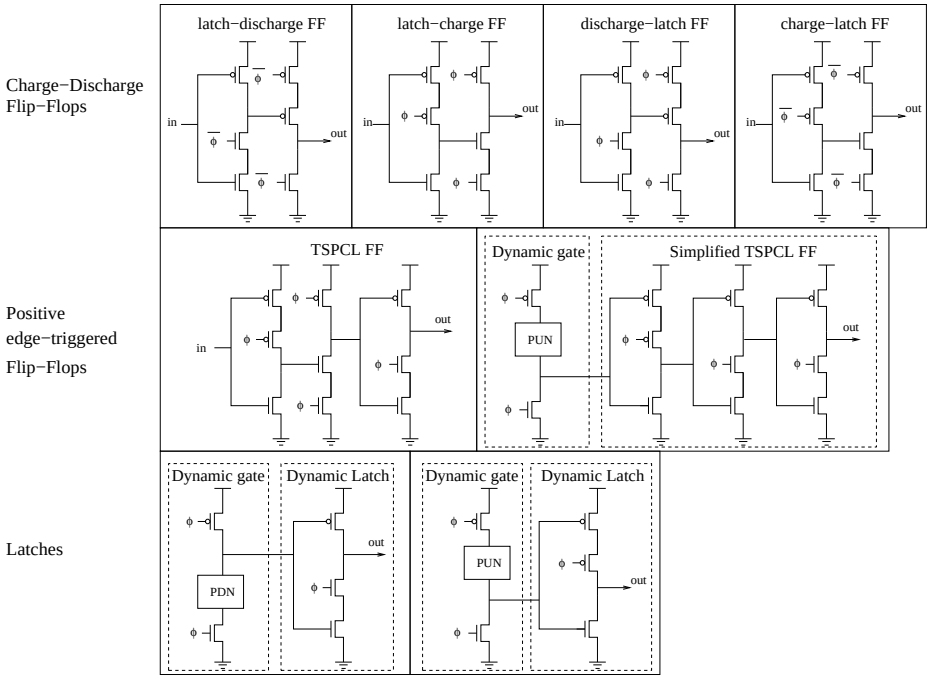


Figure 6: The schematics of the dynamic single-phase storage elements used in the design of the register file. The *charge-discharge* flip-flops combine the flip-flop functionality with the charging or discharging required in dynamic logic. They latch the input value to their output at the clock edge marking the beginning of the evaluation phase, and charge or discharge their output during the precharge or predischarge phase.

version of the TSPCL flip-flop shown in Figure 6.

Internal to the register file logic, the outputs of the address decoders are latched, as can be seen in Figures 3 and 4. Since, as Figure 5 shows, the address decoders outputs will be precharged to V_{DD} during $\bar{\phi}$ (for the read address decoders) or during ϕ (for the write address decoders), we were able to use the simple single-phase latches shown at the bottom of Figure 6.

However, using latches for the output of the read address decoders introduces some static power consumption. The reason is that latches, being transparent through the evaluation phase of the decoders, may result in activating a read word line and thus starting a read access before the precharge phase of the read bit lines is over. Doing so will lead to static power consumption, since both the memory array and the precharge logic will be simultaneously driving the read bit lines to different values. To avoid this, we could have used negative edge-triggered flip-flops which would capture the result of the evaluation phase of the decoder at its end. The reason we opted for latches is that latches have about one fourth the clock capacitance of the flip-flops, and thus result in great clock power savings, while

on the other hand the static power consumed on the bit lines is minimal, as we will show in Section 6.2.

4.2 Read Address Mapping Block

As we discussed in Section 2, the external to internal read address mapping block is used to route each read address to the correct bank port. The MS bit of the address selects the bank, and the 2-bit mapping signal specifies the port.

Figure 7 shows the schematic of this block for a single read port. It consists of a 3-bit decoder to decode the MS address bit and mapping signal, and a 7-bit 1-to-8 demux to route the seven LS address bits to the port specified by the decoder. One more 1-bit 1-to-8 demux is used for the read enables.

The decoder is implemented in static CMOS logic. For the pull-down tree network it uses a binary reduction scheme to reduce area and present a small capacitive load to the flip-flop outputs.

The 1-to-8 demultiplexer is implemented using dynamic pass-transistor logic. All outputs are precharged to V_{DD} during precharge phase $\bar{\phi}$, and the bits of one of the outputs are selectively discharged during discharge phase ϕ depending on the decoder output and the read address. Using only a single pass-transistor to selectively discharge each output line makes it possible for the read address mapping to finish at the first half of the evaluation phase, leaving the second half to the address decoders, as we saw in the timing diagram of Figure 5.

4.3 Read Data Mapping Block

The read data mapping block does the reverse mapping, by routing the data read from a bank to the external read port that requested it. The block consists of eight 64-bit 1-to-8 demultiplexers that map each of the eight 64-bit data-out buses of the two banks to the correct external data-out bus.

Figure 8 shows the schematic of this block for a single data-out bus. The demultiplexer is implemented using dynamic pass transistor logic, which makes it fast enough to operate in the second half of the evaluation phase $\bar{\phi}$, as we saw in the timing diagram of Figure 5. The 8-bit select signal of the demultiplexer comes from the decoder of the read address mapping block (Figure 7), in order to avoid decoding the same signal twice.

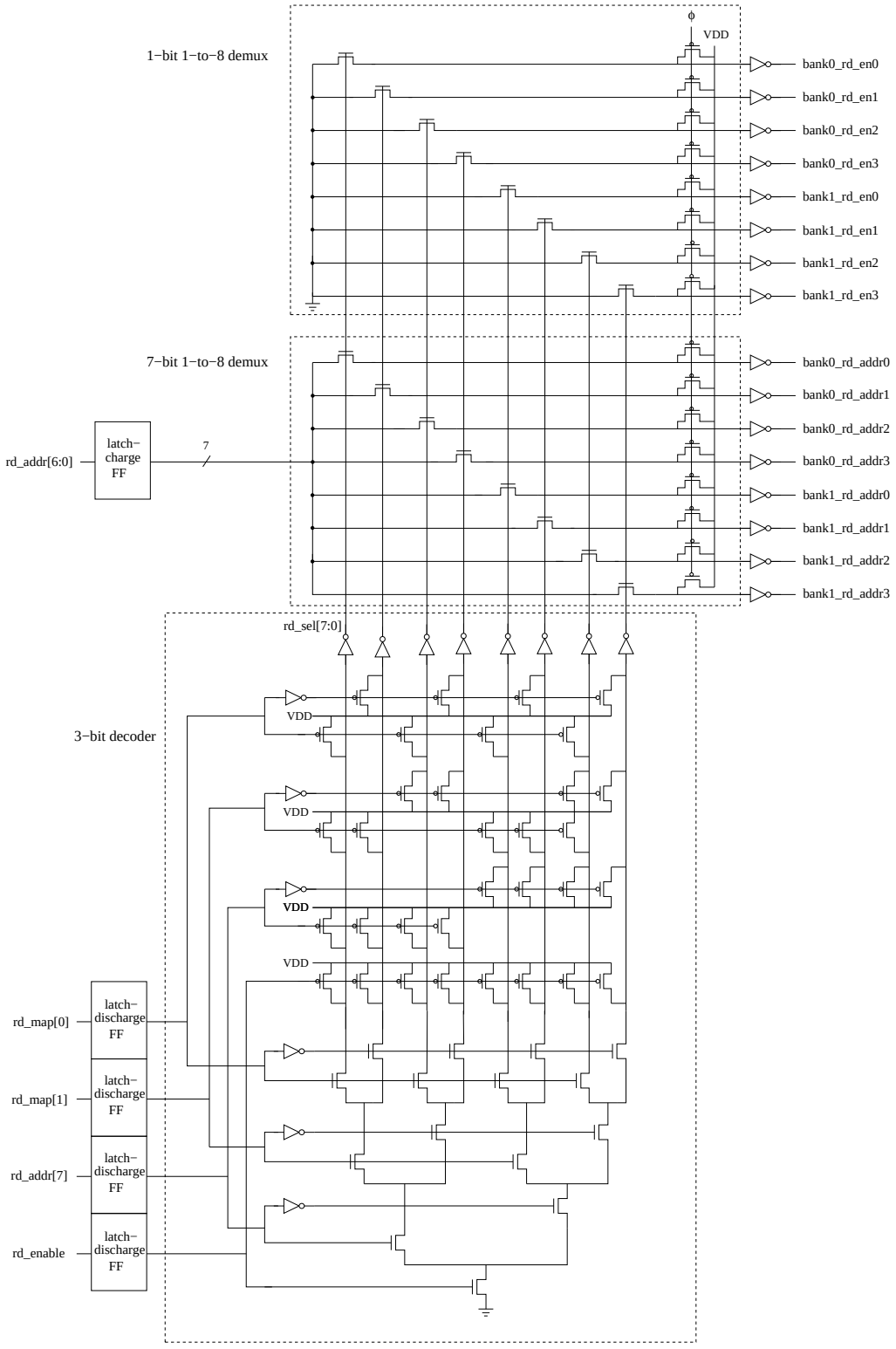


Figure 7: The schematic of the read address mapping block for a single read port consists of a 3-bit static CMOS decoder and a 7-bit 1-to-8 dynamic pass-transistor demultiplexer. It is used to route read port address to the bank port specified by the MS address bit and the 2-bit mapping signal. This design is replicated eight times for the eight read ports.

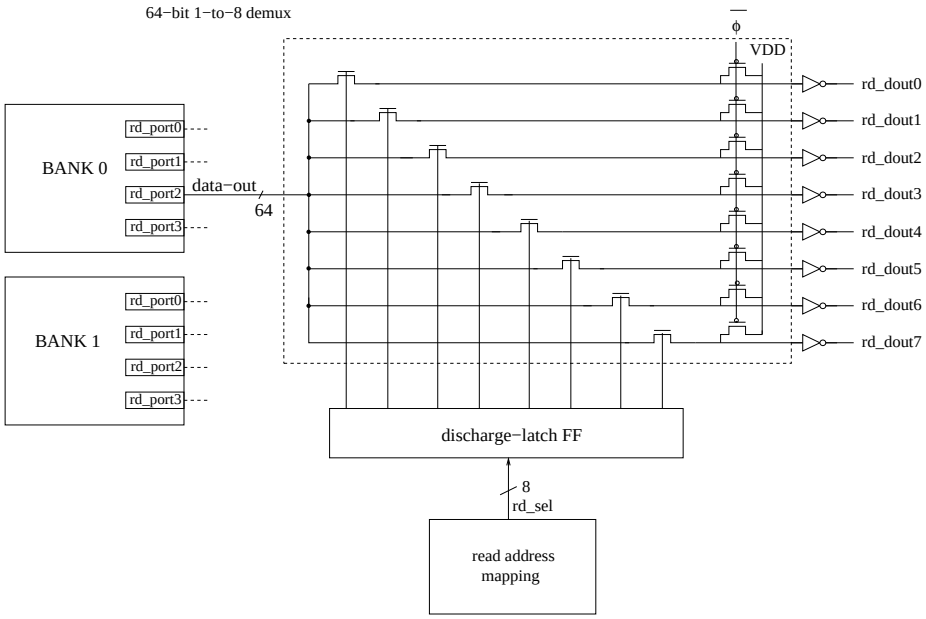


Figure 8: A 64-bit 1-to-8 demux implemented in dynamic pass transistor logic is used to map each bank data-out bus to the read port that initiated the read access. The Read Data Mapping Block consists of eight such demultiplexers, one for each data-out bus of the two banks.

4.4 Address Decoders

A 7-bit address decoder per read or write port is used to decode the port address and enable the corresponding word line, for a total of seven decoders per bank. Each address decoder is implemented as a dynamic NAND decoder, using one 3-bit and two 2-bit static CMOS predecoders ([Rab96], pages 591-4).

Figure 9 shows the design of the 7-bit address decoder. The 7-bit address is partitioned into three segments which are predecoded by the CMOS decoders at the bottom of the Figure. The partitioning shown was carefully selected to minimize the decoding delay. The outputs from the predecoders are then combined using dynamic 3-input NAND gates to produce the fully decoded array of word line signals. In order to resolve charge sharing problems, exposed in dynamic 3-input NAND gates, we also precharge some internal nodes during the precharge phase.

A challenging part of the design involved the floorplanning of the seven decoders for each bank. As Figure 9 shows, every decoder uses two columns of PMOS devices to precharge its internal and output nodes. This mix of PMOS columns with the NMOS devices in the NAND gates would result in increased area consumption if we were to place the seven decoders one next to the other. Additionally, the decoding speed would be determined by the slowest decoder which would be the one farthest from the memory array.

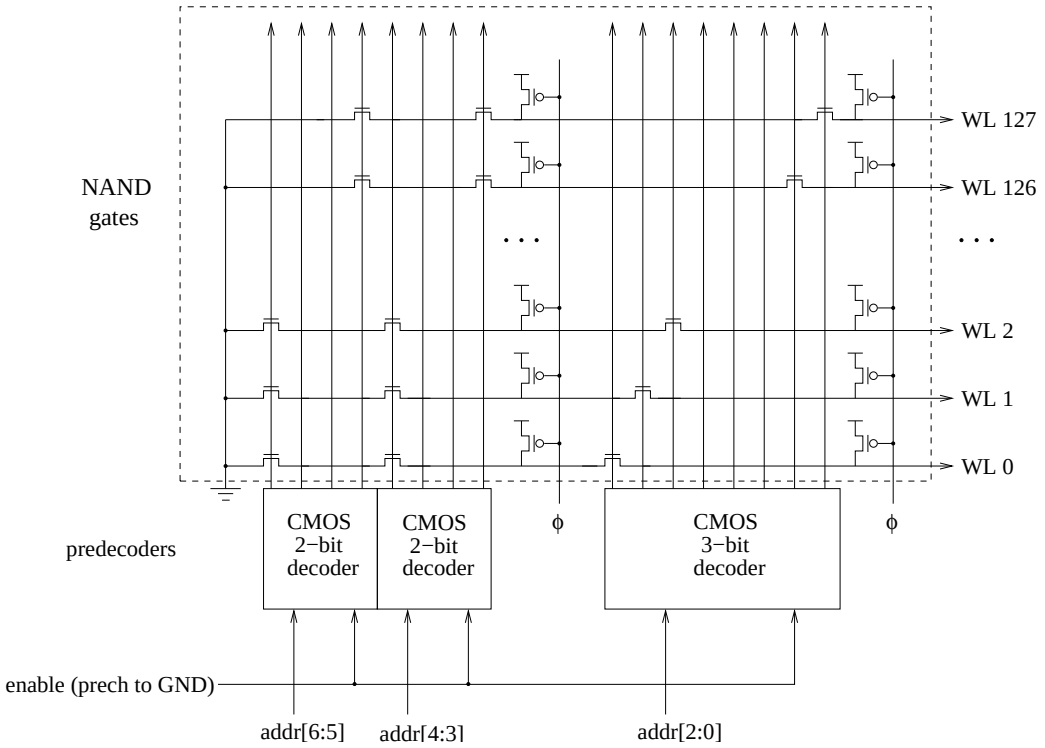


Figure 9: The 7-bit address decoder uses three predecoders. The address is partitioned into three segments which are separately predecoded by three static CMOS decoders. The resulting signals are then combined, using dynamic 3-input NAND gates, to pull down one of the 128 word lines.

We were able to solve these problems by interleaving the predecoders of the seven decoders in the way shown in Figure 10. Letters A to G are used to denote the decoder that each predecoder belongs to. As we can see the PMOS devices are combined into two columns that are shared by all decoders. Also, the predecoders of each decoder are equally spaced from one another resulting in smaller propagation delay.

4.5 Memory Array

The memory array in each bank consists of 128 words, where a word consists of 64 memory cells (bits). Figure 11 shows the schematic of a generic memory cell, as it was used. Each bit value is stored in a cross-coupled inverter pair with weak PMOS transistors and strong NMOS transistors, and is accessed by three write and four read ports. Each write port consists of a write word line, and two write bit lines holding the value to be written and its inverse. When both write bit lines are charged to V_{DD}, access to the cell is disabled. Each read port consists of a read word line and a read bit line which holds the cell value. As we will show shortly, a single read bit line is sufficient to read the value of the cell fast enough,

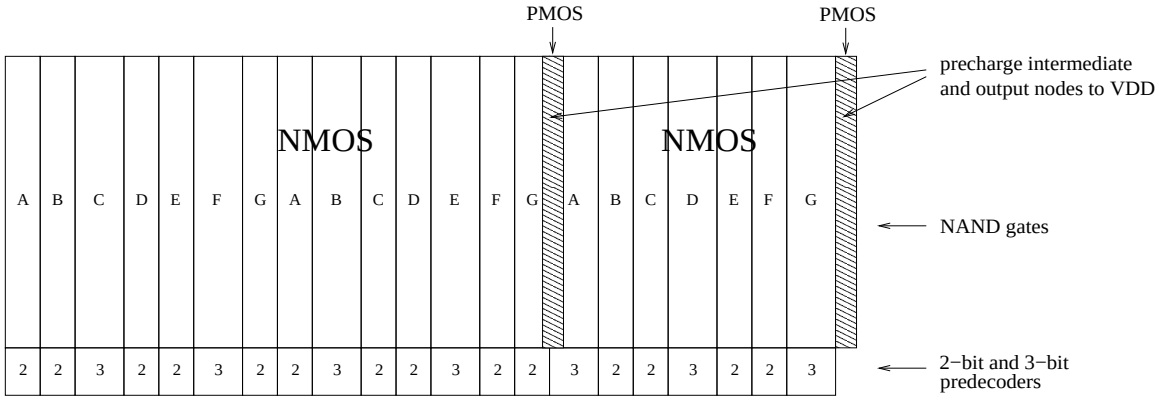


Figure 10: Physical placement of the seven decoders of each bank. Their predecoders are interleaved to reduce area consumption and increase decoding speed. The two columns of PMOS transistors are used by all seven decoders.

while reducing significantly the cell area. The layout of the memory cell is very compact since the word lines and bit lines are as close spaced as possible. In the next two sections we will describe in detail how a cell is written and read.

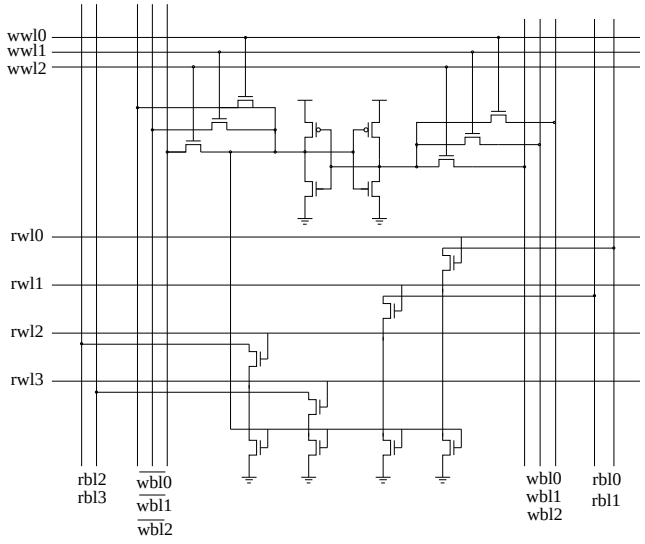


Figure 11: The schematic of the memory cell with four read and three write ports. The read port uses a single bit line, contrary to most designs that have two bit lines per read port.

Write Operation In order to write a cell, one of its three write word lines needs to be charged to V_{DD}, while the appropriate pair of bit lines is loaded with the bit value to be written and its inverse. If both write bit lines are charged to V_{DD} the cell value remains unchanged. This property is used in order to mask write operations in 16-bit word

boundaries, using a 4-bit mask. When a bit of the write mask is set to 0 all the write bit lines of the corresponding 16-bit subword are charged to V_{DD}, preventing all write operations to that subword.

Notice that disabling write operations in this way leads to minimal static power consumption. The reason is that one of the two NMOS transistors of the cross coupled inverters will try to discharge the write bit line, at the same time that the write mask charges it to prevent the write operation.

One more thing to notice is the race condition between the write bit lines and the write word line. If the word line is enabled before the bit lines of the masked-out subwords have been charged, old bit line values will accidentally be written to these subwords. For this reason, all write bit lines are precharged in the clock phase before the actual write operation takes place.

Figure 12 shows the logic used to drive each write bit line. The latch-charge flip-flop latches its input at the positive edge of the clock and precharges its output when clock is low. As a result the internal mask, that is fed to the NAND gates, is discharged and both write bit lines are precharged during $\bar{\phi}$. The write operation takes place during ϕ, when the internal mask has the correct value and the write bit lines are selectively discharged.

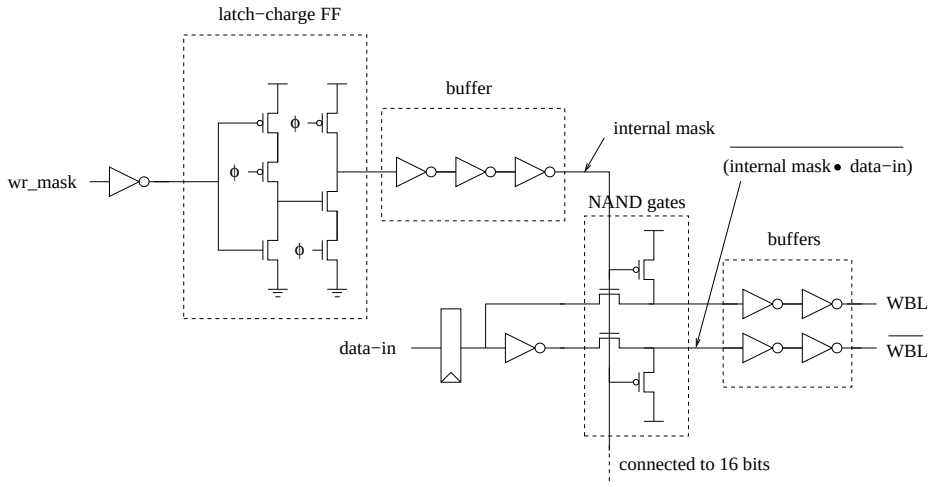


Figure 12: The schematic of the logic that sets the write bit lines. The latch-charge flip-flop discharges the *internal mask* when the clock is low, which causes the write bit lines to be precharged, thus preventing any write operation until the actual values are set at the outputs of the data-in flip-flops.

The 3-stage buffer at the output of the memory element plays dual role. First, it is needed due to the large fan-out of the latch-charge flip-flop. Second, it is used to delay the charging of the internal mask until the data input to the NAND gates is stable. In this

way, glitches at the write bit lines and their associated power consumption are avoided. Moreover, since the NAND gates may only discharge their outputs during the evaluation phase ϕ, a single pass-transistor instead of a more complex transmission gate is sufficient.

Figure 13 shows the timing diagram of the write operation. During $\bar{\phi}$, the 4-bit internal mask is discharged, causing the write bit lines to precharge. During ϕ, the mask bits are selectively charged, causing selective discharging of the write bit lines, to allow access to the correct subwords.

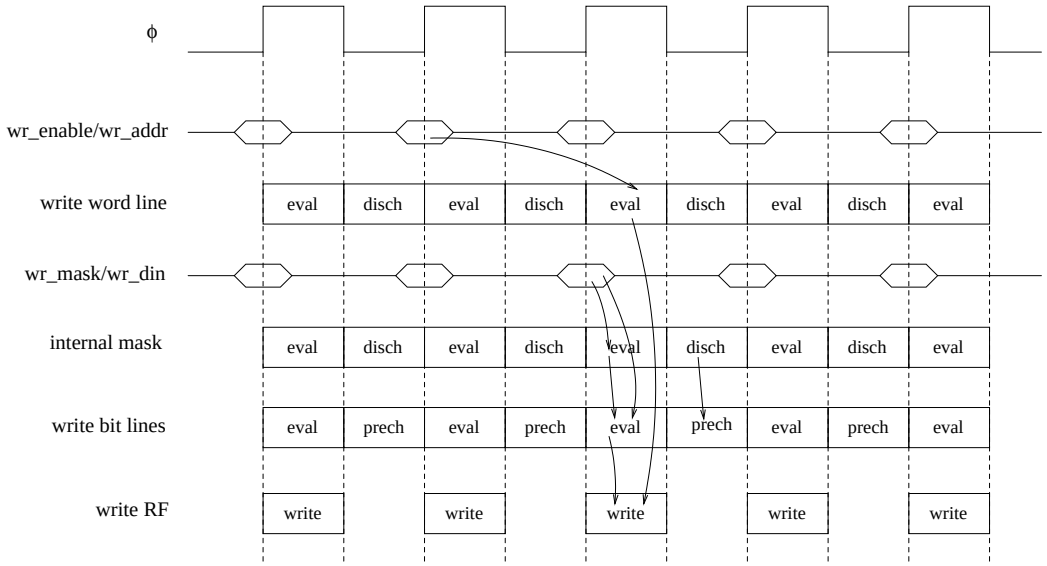


Figure 13: The timing diagram of the write operation. A word is written during the first half of a cycle, while the write bit lines are precharged during the second half. In this way, we prevent potential race conditions between the write bit lines and the write word lines.

Read Operation The two memory arrays of the vector register file perform read operations in two phases. All the read bit lines are first precharged. The read operation starts by asserting the word line of a read port. If the stored value is 0 the read bit line is discharged. Sense amplifiers are used by most memory arrays to speed-up the read operation. A number of important functions that influence the functionality, performance, and reliability of the memory are attributed to the sense amplifiers. Differential amplifiers [HS97] present numerous advantages over their single-ended counterparts, one of the most important being the *common-mode rejection* [DP98] which makes them less sensitive to noise. Sources of noise can be switching spikes on the supply voltages and capacitive cross talk between word and bit lines. Single-ended sense amplifiers are usually more sensitive in noise, but are simpler to implement and result in smaller memory arrays since they require a single read bit line.

For these reasons we used the single-ended sense amplifier of Figure 14. This simple approach is fast and has good noise characteristics, as we will shortly explain. The sense amplifier consists of the two inverters E and F in series. When the voltage of the read bit line drops below the threshold of inverter E, the sense amplifier reads a 0, otherwise its output stays at 1.

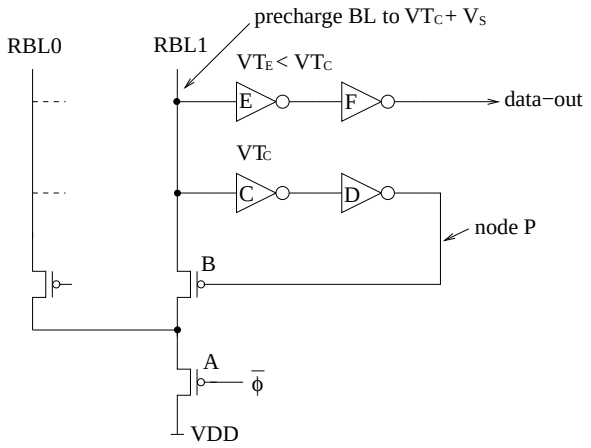


Figure 14: Precharge and read the bit line using inverters connected in series. Separate inverter pairs with different inverter thresholds are used for reading and precharging the bit line in order to guarantee the correct functionality of the read operation. The read bit line is precharged slightly above VT_C. The low voltage swing of the read bit line speeds up the read operation and leads to low power consumption.

Figure 14 also shows the precharge logic of the read bit line. This logic affects directly the performance and functionality of the sense amplifier. PMOS transistors A and B and inverters C and D are used to precharge the bit line. During the precharge phase the read bit line is precharged slightly above threshold VT_C of inverter C. This occurs because inverters C and D switch as soon as the voltage of the read bit line exceeds VT_C. Since the read bit line is not precharged to V_{DD}, both inverters connected to it consume static power. In Section 6.2 we will show that this power is usually minimal.

Figure 15 shows the timing diagram of the read operation. First the read bit line is precharged since both gates of the PMOS transistors A and B are charged to GND. Since the threshold of inverter E is lower than the threshold of inverter C, data-out, as well as node P, are precharged to V_{DD}. As shown in Figure 15, the read bit line stops being precharged when node P is precharged to V_{DD}. In the next half cycle a 0 is read, as soon as the corresponding read word line is activated, and therefore the voltage of the read bit line drops below the thresholds of inverters C and E. This causes data-out to be discharged. If a 1 is read the voltage of the read bit line stays the same and the voltage of data-out stays

at V_{DD}. The low voltage swing of the read bit lines results in a fast and low power read operations.

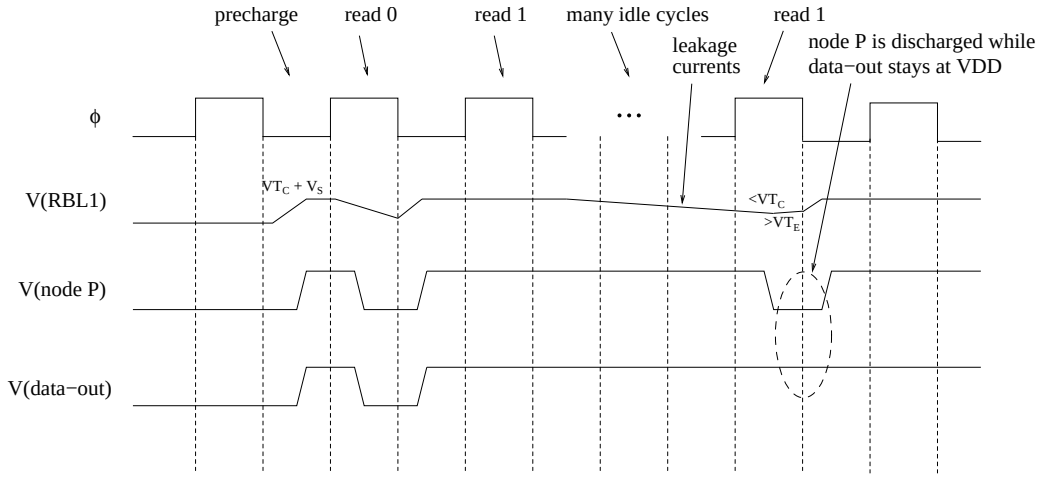


Figure 15: The timing diagram of the read operation. PMOS transistor B is open at the start of an idle period, and therefore the bit line is not precharged. Due to leakage current, the charge of the read bit line leaks away, resulting eventually in switching inverters C and D, but not E and F because $VT_E < VT_C$. As a result, PMOS transistor B closes and in the next precharge phase the read bit line is precharged back to VT_C + VT_S. Therefore, data-out stays to V_{DD} during the whole period.

The additional *charge* V_S above VT_C that the bit line is precharged to plays a critical role since it affects the speed of the read operation, the static power consumed by the inverters connected to the read bit line, as well as the noise immunity of the sense amplifier. The larger the charge V_S is, the smaller the static power consumption and the sense amplifier noise sensitivity, but the slower the read operation will be. Charge V_S is determined by the strength of the PMOS transistors A and B and the delay of inverters C and D of Figure 14. The stronger A and B are or the slower C and D are, the bigger V_S will be. All transistors involved were carefully sized to optimize timing and power consumption, while at the same time ensure correct operation even with the worst-case device parameters.

The reasons why we had to use two inverter pairs, one to precharge the bit line and one to read it, with different switching thresholds, are the bit line leakage current and noise considerations. If the same inverter pair had been used for both purposes, then leakage current or a small noise on the bit line voltage might cause an incorrect reading of the memory cell value. However, if two inverter pairs are used, no such danger exists, as Figure 15 shows. If the charge of the read bit line gradually leaks away, it will get precharged when it drops below VT_C, without ever affecting the voltage on data-out, assuming that the memory cell value is 1. Thus, the voltage difference between the voltage that the read bit line is precharged to, which is determined by VT_C, and threshold voltage VT_E of inverter

E provides a safety margin against bit line leakage current or noise.

4.6 Clock Trees

The clock generator of the VIRAM chip generates a *global clock* which is distributed to the various blocks of the chip, one of which is the register file of each vector lane. Even though different blocks are supposed to see the "same" clock, mismatches in wire lengths can result in *clock skew*. The global clock is guaranteed to have a clock skew of at most $0.3ns$ and a rise and fall time of at most $0.2ns$. Each block uses this global clock as the input to its local clock trees, which should introduce an additional clock skew of at most $0.1ns$.

In the register file, the clock signal connects to all flip-flops, latches, and precharge, predischarge or evaluation transistors of the dynamic gates. The vector register file uses four separate clock trees, each one producing a clock optimized for use by certain types of flip-flops or latches.

The need for these four *local clocks* arises as follows. Figure 6 on page 10 shows the different types of flip-flops and latches used in the design. As we see, they use both ϕ and $\bar{\phi}$ as their clock inputs. In order to increase speed, the clock received by a positive edge-triggered flip-flop should be optimized for a small rise time. Similarly, the clock received by a negative edge-triggered flip-flop should be optimized for a small fall time. For example, a latch-discharge flip-flop should ideally receive $\bar{\phi}$ with optimized negative edge as its clock input. In a similar way, latches should receive a clock optimized for their latch phase, and dynamic gates should receive a clock optimized for their evaluation phase. Optimizing only one of the two clock edges allows for a clock tree design that uses inverters sized to favor a certain edge, thus resulting in smaller area and reduced power consumption.

Figure 16 shows *spice* waveforms for the four different local clocks that are generated. Each clock is either ϕ or $\bar{\phi}$ with optimized positive or negative edge. There is a fixed delay between the global clock and the optimized edge of any local clock, which is $1.12ns$ when simulated with a supply voltage of $1.3V$ and worst-case technology parameters.

The global clock is received in the center of the register file, where it is bufferred and split into four clock trees, one per local clock. Each local clock is distributed in the vector register file and buffered close to the clock inputs of the flip-flops or latches that it connects to. Some local clocks are slightly modified according to the requirements of each block. For example, the clock of the precharge logic in the 7-bit decoders was delayed in order to avoid precharging the decoder outputs before their value is captured by the subsequent latches.

In order to keep clock skew introduced by the local clocks at a minimum, the four clock

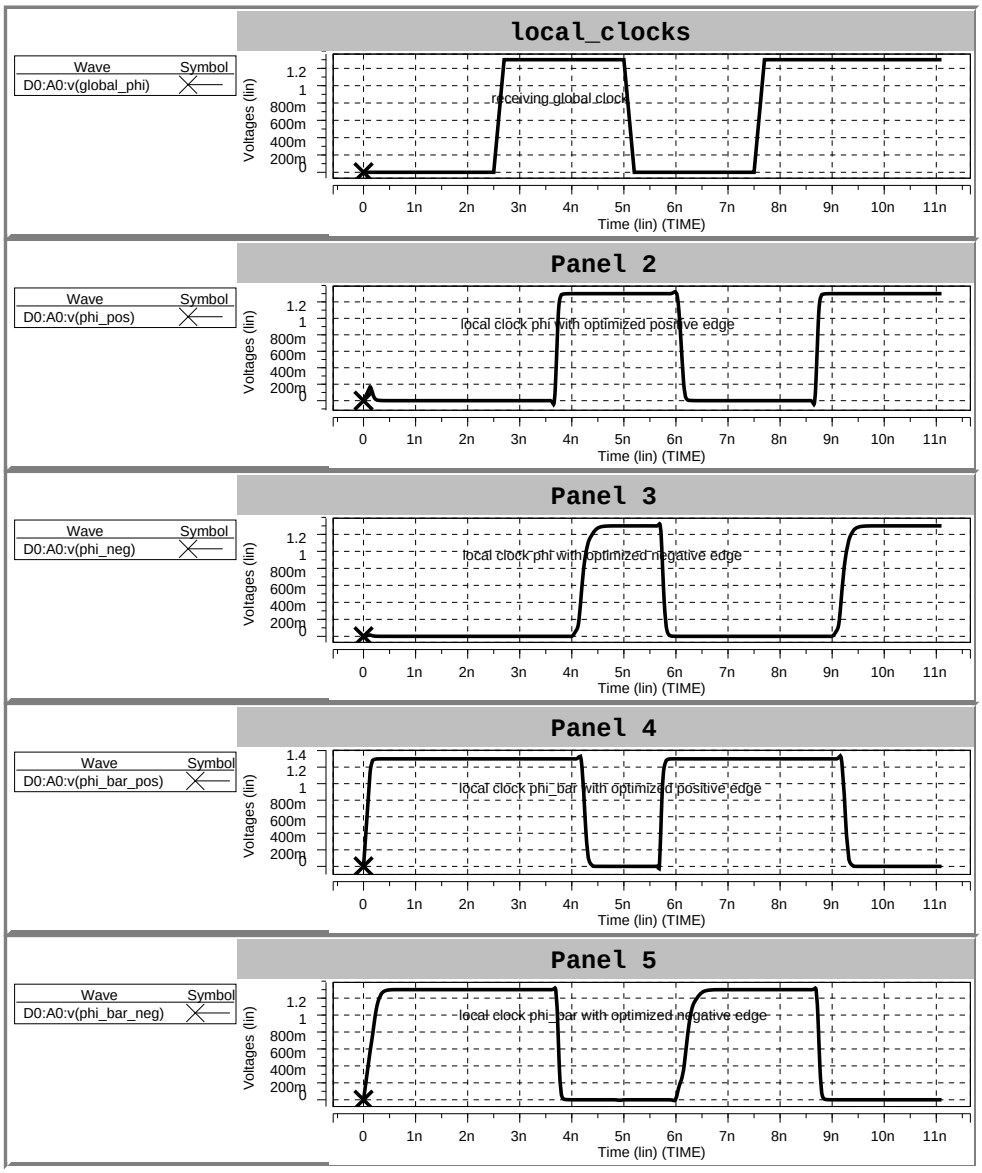


Figure 16: *Spice* waveforms of the received global clock and the four locally generated clocks. Each local clock is optimized for the positive or the negative edge of the global clock or its inverse. There is a fixed delay of $1.12ns$ between the global clock and the four local clocks.

trees were designed to be as balanced as possible. When a local clock reached an array of closely spaced latches or flip-flops, it would not be practical to separately drive the clock input of each individual latch or flip-flop of the array. Instead, their clock inputs are shorted by connecting them all together, and the local clock will connect to several points of this connection.

4.7 Floorplan

Figure 17 shows the floorplan of the register file, drawn slightly off-scale. It occupies a rectangular area of $1.03mm \times 1.93mm = 1.99mm^2$. The two memory arrays are the largest blocks and occupy 50% of the total area. Another 20% is occupied by the address decoders, 9% by the two mapping blocks, 12% by the flip-flops, latches and the precharge logic of the bit lines, and 9% by the unused area on top of and below the read address mapping block.

All address, enable and mapping input signals are routed to the left side of the register file. The write data and mask inputs of all three write ports are routed to *both* the top and the bottom of the register file, as the Figure shows, since they might be used by any of the two banks. Finally, the read data outputs are generated between the two memory arrays and are routed on top of the register file.

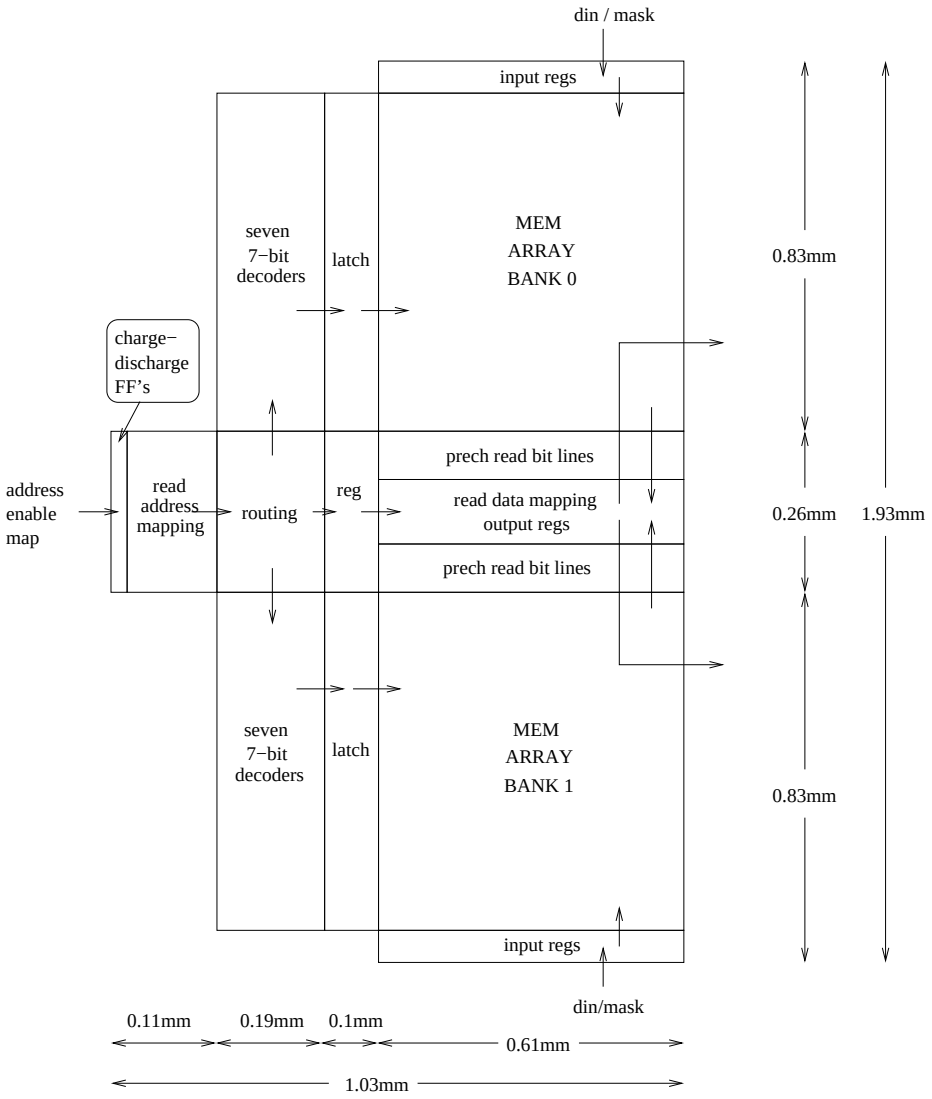


Figure 17: The floorplan of the register file inside a vector lane. It occupies a total of $1.03mm \times 1.93mm = 1.99mm^2$, with the memory arrays of the two banks being the largest blocks and consuming 50% of this area.

5 CAD Flow and Verification Approach

Figure 18 depicts the CAD flow used for the design and simulation of the vector register file. The layout was done using the Cadence layout tools. Avant! *starex* tool was used to extract a spice model with the ideal netlist of the layout, and *starrc* for a more accurate spice model which included all the parasitics.

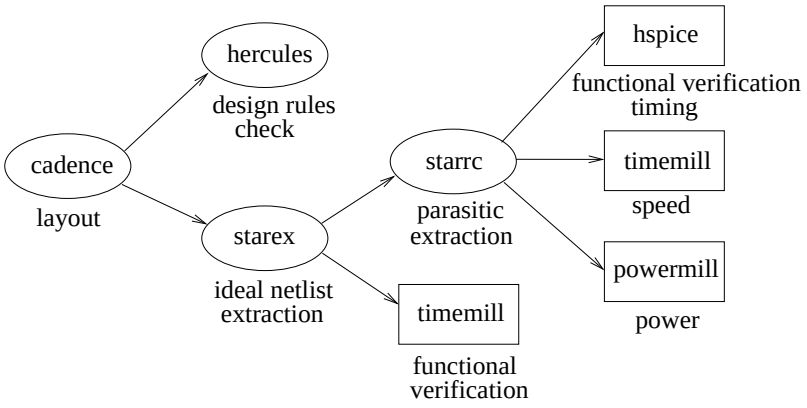


Figure 18: The CAD flow used for the design and simulation of the register file. Cadence layout tools were used for the layout and Avant! extraction tools, starrc and starex, for the generation of spice models with or without parasitics. Functional verification was performed at various levels of block integration using hspice and timemill. Timemill and Powermill were also used for speed and power measurements.

All blocks were individually simulated and functionally verified with *hspice*, using their more accurate spice models. The largest scale of block integration that was verified using hspice was the full path for writing and reading a single cell. However, simulating the whole register file with hspice was too slow to simulate. Instead, we were able to simulate it using Synopsys *timemill* and *powermill* tools, which are less accurate but faster than hspice. The functionality for the whole register file under different operating conditions was mainly verified in timemill using its less accurate (ideal netlist) spice model. A number of operating conditions were created by varying supply voltage, clock frequency and technology parameters. Timemill and powermill were also used with the more accurate spice model in order to measure its speed and power consumption.

In order to automate and improve the coverage of the register file functionality verification, we followed a *randomized self-checking* verification approach. As Figure 19 shows, a *Perl* script was built to generate a random sequence of input test vectors, which are used to stimulate the spice model of the register file. The simulation is then performed in timemill and produces the output vectors that consist of the values on the data-out buses of the eight

read ports. If these output vectors are correct the process starts over, otherwise the layout is fixed to behave correctly.

A verilog behavioral model (not shown in the Figure) of the register file was also built and verified against the layout design. Such a model is necessary to allow cosimulation of the register file with the rest of the IRAM chip.

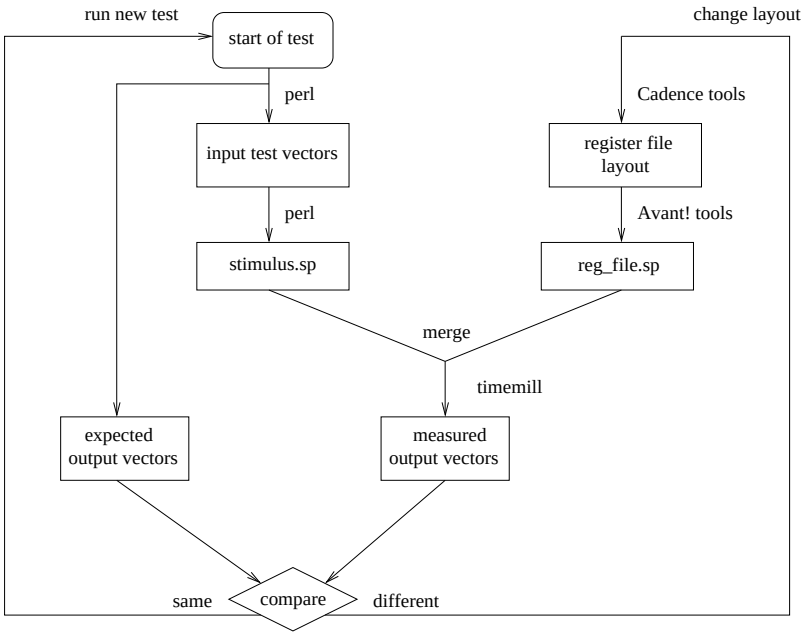


Figure 19: Flowchart for the randomized self-checking verification approach. A Perl script is used to generate random stimulus to the spice model of the register file and verify its functionality. The process is repeated until a design problem is found, at which point the layout is fixed.

6 Performance

6.1 Speed

The maximum operating frequency of the vector register file was derived from simulations performed with *Timemill* and *Hspice*. Since Hspice is significantly slower than timemill, only parts of the design were simulated using it, while the full design was simulated with the much faster Timemill. However, the part of the design simulated with Hspice was sufficient to fully read and write one cell. Both tools verified that the vector register file operates at the target frequency of 200 MHz and supply voltage of 1.3 V, with the worst-case technology parameters.

Figure 20 shows the maximum operating frequency of the vector register file as a function of the supply voltage for varying technology parameters. We derived the operating frequency of our design for the worst-case (slowest), the typical, and the best-case (fastest) technology parameters provided by IBM. The operating frequency changes almost linearly to the supply voltage, as expected.

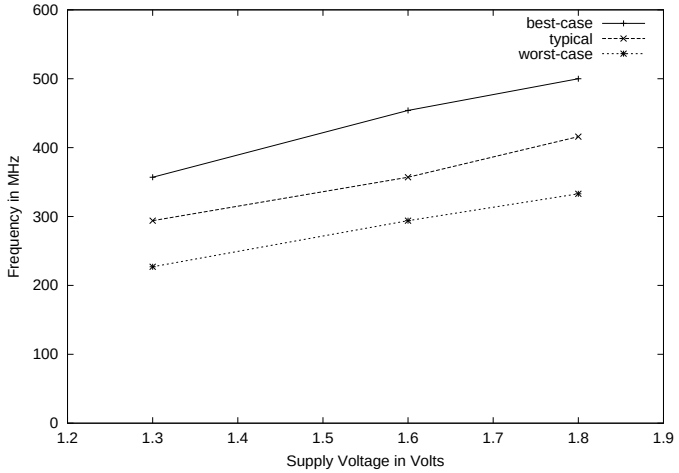


Figure 20: Maximum operating frequency versus supply voltage for the worst-case, typical and best-case technology parameters. The register file can operate at up to 300 MHz with the typical technology parameters and a nominal supply voltage of 1.3 V.

Table 2 shows the delays of the various blocks in the register file for read and write operations, at a supply voltage of 1.3 V with the worst-case technology parameters.

Write Operation		**Read Operation**	
decoder enabling	0.2ns	address mapping	0.6ns
address decoding	0.7ns	address decoding	0.7ns
WWL charging	0.4ns	RWL charging	0.4ns
WBL discharging	0.9ns	RWL sensing (read 0)	1.2ns
cell value loading	0.6ns	data mapping	0.3ns

Table 2: The delays of the main operations performed by the register file, using the worst-case technology parameters at 1.3 V supply voltage.

6.2 Power

Low power consumption was one of the main challenges and considerations throughout the design of the vector register file, and greatly influenced most of our design decisions, as has already been discussed. When operating at the target clock frequency of 200 MHz and supply voltage of 1.3 V, it consumes between 60 and 430 mW, depending on the frequencies of read and write accesses. The VIRAM target applications are expected to generate about 4 read and 3 write accesses per cycle, in which case the estimated power consumption is *330 mW* (the exact number also depends on the actual data values that are being written and read).

Figure 21 shows the break-down of the power consumption into the individual blocks of the design, for various access frequencies. The numbers shown are the average power consumed during randomized simulations of the register file, at the nominal operating conditions of 1.3 V at 200 MHz with typical technology parameters. In these simulations the write and read addresses were randomly picked from a random set of 5 to 10 addresses, write data values were set at random, and a write mask bit would disable a subword write access with a 25% probability. Due to the small set of addresses used per simulation, the same address would be often used for multiple read accesses and (at most) one write access during a cycle. We will next briefly discuss the power consumed by each block.

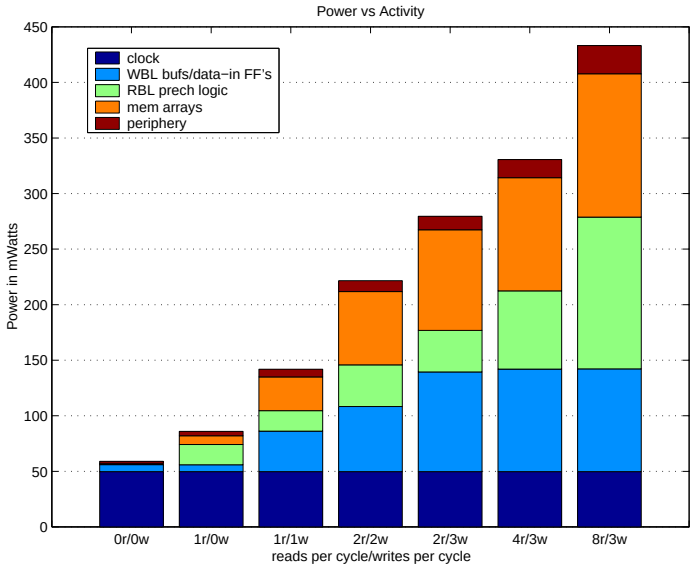


Figure 21: Power versus activity at 200MHz with a 1.3V supply voltage and the typical technology parameters. A write operation per cycle consumes 30mW and a read per cycle consumes 15mW.

The write bit lines along with the logic that drives them (see Figure 12 on page 16) are responsible for a big portion of the power consumption. They consume close to 30 mW

for each additional write access per cycle, ranging from 6 mW when no write accesses are performed, to 92 mW when all 3 write ports are constantly active. The power they consume is due to the charging and discharging of the write bit lines. One small portion is also attributed to the static power consumption caused by the write-mask when disabling access to certain subwords.

The read bit lines consume 15 mW for each additional read access per cycle, ranging from 0.4 mW when no read accesses are performed to 136 mW when all 8 read ports are constantly active. Both dynamic and static power consumption contribute to these numbers. Precharge logic and fast sense amplifiers were used in an effort to operate the read bit lines in as low voltage swing as possible, and thus minimize the dynamic power consumption which is usually the main contributor. However, this unavoidably resulted in some minimal static power consumption, of 0.4 mW, by the inverters of the precharge logic, as we discussed in Section 4.5, which is present even when the read ports are inactive. Another source of static power consumption, which increases with the read activity, is the one that results from using latches at the outputs of the address decoders (Section 4.1).

The clock consumes a constant of only 50 mW, which is a result of the extra effort that went into designing the appropriate types of flip-flops or latches for each block (Section 4.1), and generating multiple optimized local clocks depending on their usage (Section 4.6).

Finally, the memory array and the periphery circuitry, that consists of the address decoders and the two mapping blocks, will consume at most 25 mW.

Figure 22 shows how the power consumption varies with the supply voltage for different read and write activites. As expected, the power increases with V^2.

Figure 23 shows how power consumption is affected by varying technology parameters. The reason why it stays practically the same is that varying the technology parameters mainly affects the static power consumption, which is a small portion of the total. Using the worst-case parameters results in slowing down the read bit line precharge logic, which stops at a lower voltage of 0.77 V instead of 0.9 V with the typical technology parameters. This value is closer to the switching threshold of the inverters connected to the read bit lines, leading to up to 50mW of static power consumption. On the other hand, using worst-case technology parameters also slows down the address decoders and leads to decreasing the static power consumed by latching their outputs.

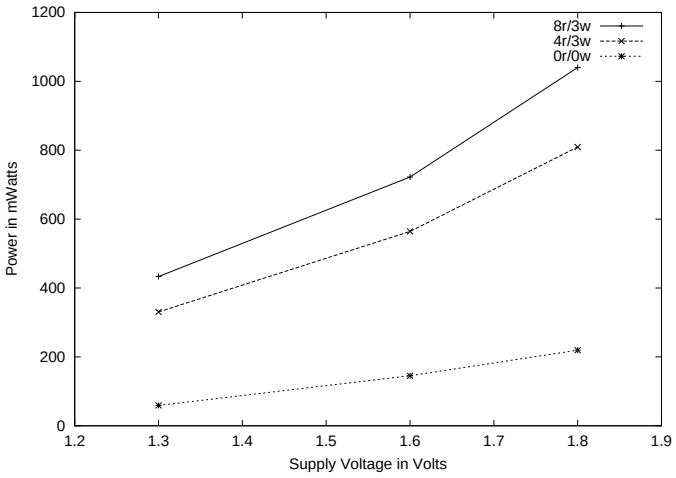


Figure 22: Power versus supply voltage at 200MHz with the worst-case technology parameters. The power increases with V^2.

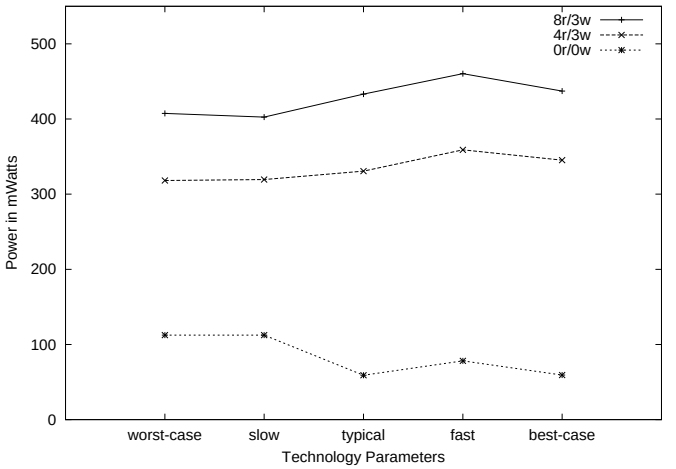


Figure 23: Power versus technology parameters at 200Mhz with a 1.3V supply voltage. The power is practically independent of the technology parameters when the activity is high. The static power of the inverters connected to the read bit lines is high with worst-case and slow technology parameters, resulting in an increased power consumption when the activity is low.

7 Related Work

Many register file designs are presented in the literature. Multi-ported register files play a significant role for parallel processing of instructions in high-speed microprocessors or digital signal processors (DSP).

The architecture of the vector register file, as well as the architecture of the whole IRAM chip, was influenced by the design of T0 (Torrent-0) Microprocessor at the University of California, Berkeley [Asa98], the first single-chip vector microprocessor. T0 is a compact but highly parallel processor that can sustain over 24 operations per cycle while issuing only a signle 32-bit vector instruction per cycle. The chip was implemented at a $0.8\mu m$ technology with three metal layers and runs at 40MHz with a 5V supply voltage.

The T0 vector register file contains 16 vector registers, each holding 32 32-bit elements. It is split into eight parallel 32-bit wide slices, one for each of the eight parallel lanes in the vector unit. Each lane contains a total of 60 32-bit elements (15 vector registers with four elements per vector register).

The T0 vector register file provides five reads and three write ports. Its storage is partitioned across the eight lanes, using a single bank per lane. In contrast, the VIRAM vector register file uses two banks per lane with half the number of total read ports per bank. This results in potential conflicts when more than four read operations contend for the same bank, but greatly improves speed, area and power consumption, as we discussed in Section 2.

The address decoders of T0 vector register file are shared between the lanes, driving global decoded word select lines across all lanes, where these selects signals are gated locally with a per-lane enable signal to form local word selects. Our design used separate decoders for each bank of the vector register file because driving global word select lines would dramatically impact the timing.

The write through operation in T0 vector register file can be supported by adding external multiplexers. Such multeplexers would affect its performance, power and area. The VIRAM vector register file supports this functionality internally as explained in Section 2.2.

Both vector register files have sigle-ended read ports and differential write ports. The word and bit lines of T0 vector register file are time-multiplexed at twice the clock frequency in order to provide both a read and a write access in the same cycle, using a small number of word and bit lines. This technique saves area but requires self-timing with dummy rows to provide the extra timing edges necessary to control read precharge and write drive. Our design uses separate word and bit lines for each port. It would need more design effort to support this technique and the dynamic periphery circuit would likely not

	VIRAM	T0
Techn, Supply, Freq	$0.18\mu m$, 1.3V, 200MHz	$0.8\mu m$, 5V, 40MHz
Configuration	4 Lanes x 2 Banks	8 Lanes x 1 Bank
Elements	32 regs x 32 elems x 64 bits	16 regs x 32 elems x 32 bits
Size	8 KBytes	2 KBytes
Ports	eight read / three write	five read / three write
Conflicts	Yes (rarely)	No
Mapping blocks	Yes	No
Decoding	seven decoders per bank	5 global time-multiplexed dec.
Write-through	Internally supported	Need of external muxes
bit/word line time-multiplexing	No	Yes
Total area	997 Mλ^2 = $8mm^2$	167.6 Mλ^2 = $41.9mm^2$

Table 3: Differences between the VIRAM and T0 vector register files.

be suitable.

Table 3 summarizes the differences between the two designs.

IBM technology library [IBM00] provides macro-cells for register arrays. These register arrays are multi-ported register files with asynchronous write and read functions. Each write port has a write clock pin that specifies the start of a write function. In this way, when the write clock pins are inactive the register file does not consume any power.

Using a gated input clock, the vector register file could consume minimal power when it is idle. However, if any port is accessed the clock will consume about 50 mW, as shown in Figure 21. If the vector register file had separate gated clocks per port, similar to the write clock pins provided by the IBM register files, the clock power consumption would depend on the number of ports that are used during a cycle. As a result, the clock would consume less power, if some but not all of the ports were accessed. Since it would be hard to generate a separate clock tree for each port and the clock power consumption is a small portion of the total, we decided not to follow this approach.

8 Conclusions

VIRAM is a microprocessor optimized for multimedia applications by integrating vector processing with embedded DRAM on a single chip. This report presented the vector register file of this processor, which has a total capacity of 1024 64-bit words and provides eight

read and three write ports. We described its architecture and discussed its implementation in detail. We also evaluated its speed and power consumption. It operates at 200 Mhz with a 1.3 V supply voltage and consumes 8 mm^2 of area and 330 mW of power on average.

Acknowledgments

First I would like to thank my advisor, Professor David Patterson, for his overall guidance, remarks and encouragement. I would also like to thank all the members of the VIRAM group in U.C. Berkeley, and in particular Joseph Gebis, Christoforos Kozyrakis, and Sam Williams, for their support, technical advise and feedback. I also wish to thank Professors Jan Rabaey and Borivoje Nikolic for teaching me Digital Integrated Circuits and providing feedback to my work. My brothers, Ioannis and Dimitris, helped me a lot with both discussing the design of the vector register file and reviewing this report. Finally, I would like to thank my parents, Manolis and Evangelia, for their love and support during my graduate studies.

Appendices

A Layout

The layout was implemented using CAD layout tools from *Cadence*. Our design is very compact, using the first three metal layers from the six available. The first metal layer was used for local routing and the second and third for vertical and horizontal global routing respectively. In this way, routing was greatly simplified since we did not have to worry about one direction colliding with the other. Figure 24 shows the layout of the memory cell and Figure 25 shows an annotated layout of the whole register file that is included in a vector lane.

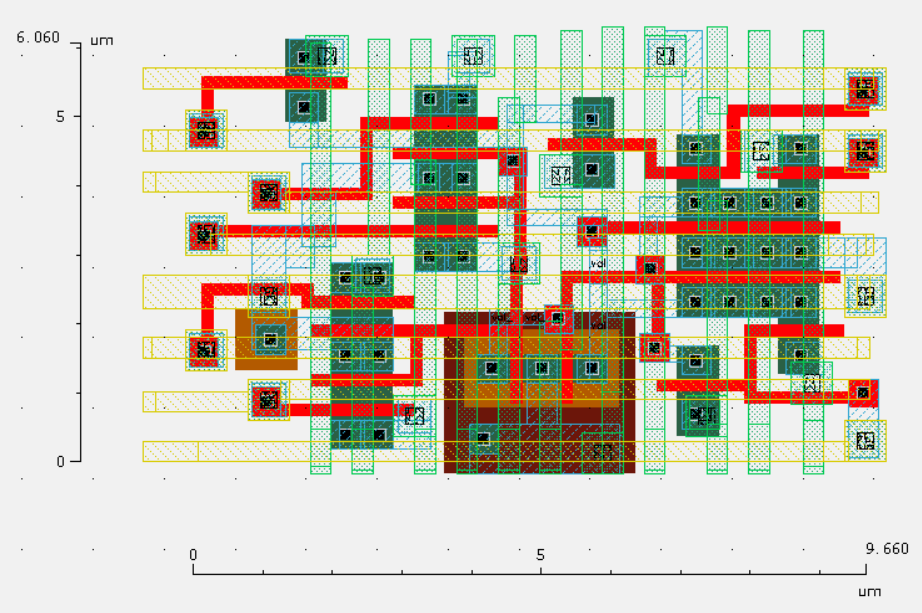


Figure 24: Layout of the memory cell. The schematic for the cell can be seen in Figure 11.

B Simulation Results

The exact values of the quantities plotted in Figures 20, 21, 22 and 23, are included in Tables 4, 5, 6 and 7 respectively.

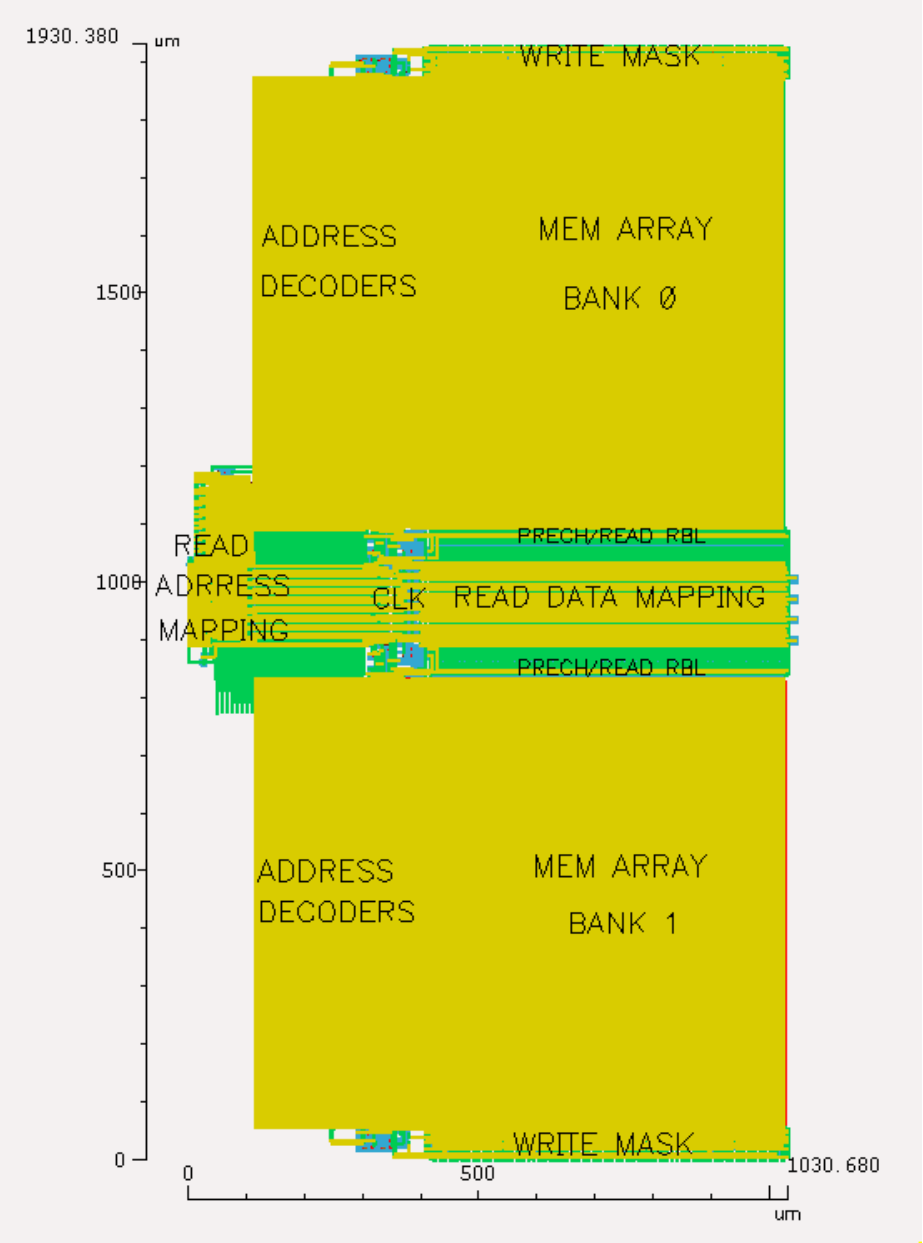


Figure 25: Annotated layout of the register file included in a lane. See Figure 17 on page 23 for the corresponding floorplan.

	1.3V	1.6V	1.8V
worst-case	227MHz	294MHz	333MHz
typical	294MHz	357MHz	416MHz
best-case	357MHz	454MHz	500MHz

Table 4: Frequency versus supply voltage for the worst-case, typical and best-case technology parameters.

Activity	Clock	WBL buf/FF's	RBL prech logic	Mem arrays	periphery	total (mW)
0r/0w	49.7	6.3	0.4	0.7	1.8	59.0
1r/0w	49.7	6.2	18.2	7.9	4.0	86.0
0r/1w	49.7	36.5	0.4	23.5	4.2	114.4
1r/1w	49.7	36.5	18.3	30.4	6.9	141.9
2r/2w	49.7	58.6	37.5	65.9	9.8	221.6
2r/3w	49.7	89.7	37.4	90.5	12.2	279.5
4r/2w	49.7	59.1	71.8	79.9	14.2	274.8
4r/3w	49.7	92.2	70.4	101.8	16.3	330.6
8r/3w	49.7	92.5	136.6	129.0	25.4	433.2

Table 5: Power in milliwatts versus activity at 200MHz with the typical technology parameters.

	1.3V	1.6V	1.8V
0r/0w	59.1	145.2	219.5
4r/3w	330.6	564.2	809.3
8r/3w	433.2	722.3	1040.2

Table 6: Power versus supply voltage at 200MHz with the typical technology parameters.

	worst-case	slow	typical	fast	best-case
0r/0w	112.4	112.4	59.0	78.2	59.3
4r/3w	318.2	319.4	330.6	359.0	345.2
8r/3w	407.5	402.5	433.2	460.4	437.2

Table 7: Power versus device parameters at 200MHz with a 1.3V supply voltage.

References

[Asa98] Krste Asanovic. *Vector Microprocessors*. PhD thesis, Technical Report UCB//CSD-98-1014, Computer Science Division, University of California at Berkeley, May 1998.

[DP98] William J. Dally and John W. Poulton. Digital Systems Engineering. Press Syndicate of the University of Cambridge, ISBN 0-521-59292-5, 1998.

[HP96] J. Hennessy and D. Patterson. Computer Architecture: A Quantitative Approach, second edition. Morgan Kaufmann, 1996.

[HS97] Rogert T. Howe and Charles G. Sodini. Microelectronics: An Integrated Approach. Prentice Hall, ISBN 0-13-588518-3, 1997.

[IBM00] IBM. ASIC SA-27E Technical Library . http://www-3.ibm.com/chips/techlib/techlib.nsf/products/ASIC_SA-27E, 2000.

[Koz99] Christoforos Kozyrakis. A Media-Enhanced Vector Architecture for Embedded Memory Systems. Master's thesis, Technical Report UCB//CSD-99-1059, Computer Science Division, University of California at Berkeley, July 1999.

[Kra82] R. Krambeck. High Speed Compact Circuits with CMOS. *IEEE Journal of Solid State Circuits*, 17(3), June 1982.

[Rab96] Jan M. Rabaey. Digital Integrated Circuits: A Design Perspective. Perspective, Prentice Hall series in electronics and VLSI, ISBN 0-13-178609-1, 1996.

[RWM93] D. Radhakrishnan, S. Whittaker, and G. Maki. Formal Design Procedures for Pass-Transistors Switching Circuits. *IEEE Journal of Solid State Circuits*, 20(4), April 1993.

[YS89] J. Yuan and C. Svensson. High Speed CMOS Circuits Techniques. *IEEE Journal of Solid State Circuits*, 17(3):62–70, June 1989.