Asynchronous DRAM Design and Synthesis

Virantha N. Ekanayake and Rajit Manohar*

Abstract

We present the design of a high performance on-chip pipelined asynchronous DRAM suitable for use in a microprocessor cache. Although traditional DRAM structures suffer from long access latency and even longer cycle times, our design achieves a simulated core sub-nanosecond latency and a respectable cycle time of 4.8ns in a standard 0.25um logic process. We also show how the cycle time penalty can be overcome by using pipelined interleaved banks with quasi-delay insensitive asynchronous control circuits. We can thus approach the performance of SRAM, which is typically used for caches, while still benefitting from the smaller area footprint of DRAM.

1 Introduction

This paper explores the feasibility of using on-chip dynamic random access memory (DRAM) as the basis for a fast cache in the context of an asynchronous processor. DRAM has long suffered from the following two major issues which we tackle in this paper:

Long access latency: Most commercial DRAM designs incur high initial latencies to get the first word out of memory. This mainly arises due to the very long bit lines used for density reasons. As pointed out by Poulton [1] and Speck[2], short bit lines enable greater bit line voltage differences and thus faster sensing with the use of simpler sense amps, while still having a density advantage of SRAMs. Thus, our design utilizes a large number of very small banks of DRAM.

Long cycle times: Unlike SRAM, a DRAM's *cycle* time for access to a particular row is slower than the *access* time [3], due to the need for a precharge, sense, reset cycle for every access to a random row. In other words, a word can be fetched from memory fairly quickly, but additional time is needed for a reset and precharge cycle to complete before we can access the memory again for another data word. The usual solution to this has been to use multiple interleaved banks so one can continue getting a word from a different bank while the previous bank is still resetting. However, this results in timing behavior that is dependent on the memory access pattern, which complicates control design in the synchronous world. Indeed, pipelined synchronous memories usually run at the cycle time of an individual bank[4, 5]. However, an asynchonous memory controller can be designed for the highest throughput case where successive memory accesses get directed to different banks, yet still handle the slower access pattern of consecutive accesses to the same bank without any additional circuitry [6].

In this paper, we present the design of a high performance pipelined asynchonous memory utilizing many small banks of DRAM and a two level banking scheme. Section 2 describes the memory core, while Section 3 presents the asynchronous interface to the core and a quasi-delay insensitive (QDI) banking scheme. We present simulation results in Section 4 and an architectural study in Section 5.

2 DRAM Core Design

In this section we show the core memory cell design and associated analog sensing circuitry.

2.1 Core Memory Cell

The simplest possible DRAM cell is the single transistor cell shown in Fig. 1. Although very dense, this design has a destructive read operation which means the sensing circuitry must drive the bit lines with the read out value to refresh the contents, thus slowing down the read and increasing power consumption. We instead use the two transistor (2T) cell shown in Fig. 2 which in addition to having a non-destructive read also has a separate read and write interface to the storage node, which makes interfacing to the asynchronous circuitry easier.

The 2T cell is a simplification of the standard 3T cell popular for on-chip DRAM [7] that gives higher density. During a write, we drive *wbl* with the data, and then raise *wl* and allow the data to be stored on the *store* node. The charge is held by a combination of gate and diffusion capacitance.

*Virantha N. Ekanayake <viran@csl.cornell.edu> and Rajit Manohar <rajit@csl.cornell.edu> are with the Computer Systems Laboratory in the School of Electrical and Computer Engineering at Cornell University, Ithaca NY 14853, U.S.A.

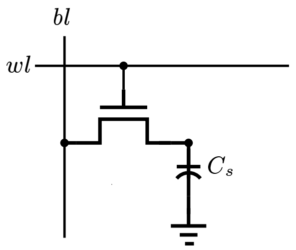

Figure 1. One transistor (1T) memory cell

Reading a value from the cell is slightly more complicated than the write. The basic operation consists of precharging the read bit line (*rbl*) and allowing the read transistor to turn on by pulling *rl* low. A logical one will start pulling *rbl* low, while a logical zero will leave the value of *rbl* unchanged. We allow this readout phase to continue for a preset margin of time t_r, during which time a logical one will drop the voltage on *rbl* to some level V_r. A readout can be completed successfully by comparing V_{rbl} to some reference voltage V_{ref} such that $V_{dd} > V_{ref} > V_r$ via a *differential sense amplifier* (which we will discuss in more detail shortly).

A refresh is accomplished by doing a read onto an internal bus, and then a write from this internal bus.

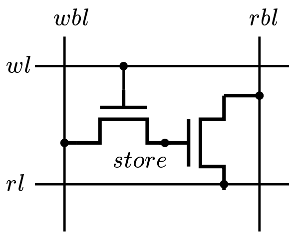

Figure 2. Two transistor(2T) memory cell

A DRAM with a one-bit wide datapath is formed by having multiple memory cells attached to the same read and write bit lines (*rbl* and *wbl*); this forms a bit line *column*, where only a single memory cell may be activated on a single cycle. Arraying many such columns makes the datapath wider, and forms *rows* which run perpendicular to the columns.

One complication with a 2T cell is the following: V_{rbl} will not drop more than two n-transistor thresholds below the V_{dd} during the read. The reason can be seen from Fig. 3, where an adjacent bit in the same column has a one stored (thus having $V_{dd} - V_{th}$ stored). If the read bit line should drop more than two thresholds, this transistor will turn on and keep the bit line from falling further; in practice, however, this margin is more than adequate for correct sensing

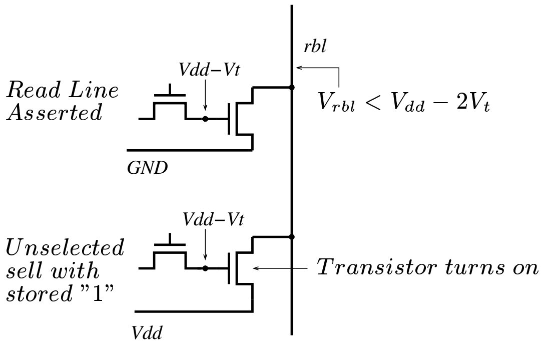

Figure 3. Allowing *rbl* to drop below $V_{dd} - 2V_t$ turns on adjacent non-activated cells storing a "1"

2.2 Sensing Circuitry

We now describe the circuitry used during a read operation.

2.2.1 Sense Amplifier

Sensing the difference between the read bit line and the reference voltage requires a circuit structure called a differential sense amplifier, which can convert a difference of several hundred millivolts into a full rail-to-rail transition. Most DRAM macros use somewhat complicated sense amp structures that by necessity must be very tolerant to manufacturing variations: Because most DRAMs are designed with very long bit lines (spanning on the order of hundreds to thousands of rows), the bit line changes that need to be sensed are very small. However, we will be using small banks of 64 cells per bit line column, allowing us to use very simple sense amp structures.

The lower bound of two threshold drops on the bit line somewhat constrains our choice of sense-amplifier, because most designs couple their output rails with their input rails. Thus, a rail-to-rail transition on the outputs also drives the input bit lines and allows, for instance, refreshing the contents of a DRAM cell with a destructive read (such as a 1T design). In our case we decided to use the DCVSL sense amplifier circuit [7] shown in 4 which looks like a pair of cross-coupled inverters. They have very high gain, and are simple and also separate the output rails from the inputs. We do need to make sure that both output rails are equalized immediately before we turn on the sense-amplifier via the SE signal; otherwise the sense amplifier will transition just based on the differences in the output rails.

2.2.2 Output Filter

During the times the sense amplifier is disabled, its output can drift to any arbitrary value. Naturally, to maintain stability and non-interference in a QDI circuit we need to ensure that both rails appear to stay at their neutral values at these times. A standard filter circuit (used in arbiter circuits) that provides this function is shown in Fig. 4; this circuit holds its outputs at ground until their inputs have separated by more than a threshold voltage.

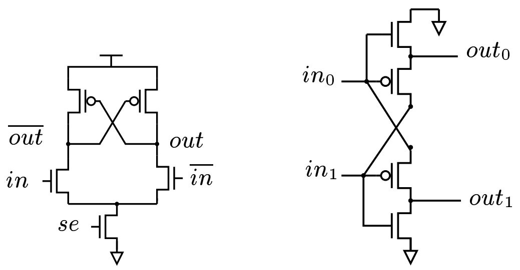

Figure 4. Sense Amplifier (left), and Output Filter (right)

2.3 Bit Line Column Organization

We now describe how all the component circuitry was put together to provide a practical DRAM bank. A block diagram of a single bit line column containing 64 memory cells is shown in Fig. 5 (only one DRAM cell in a column is activated during a read or write). The individual sections referenced in the figure are detailed below:

A. Bit Line Precharge - Because we have a $2V_{th}$ sense margin to work with, we use a reference voltage of $(V_{dd} - V_{th})$. Both inputs to the sense amplifier are precharged via n-transistors to this level, and the bit lines equalized. Then, the bit line itself is pulled up using a weak p-transistor while the read is taking place, allowing it to drift up to V_{dd} if a zero is being read, or be pulled down close to $V_{dd} - 2V_{th}$ if a one is being read.

B. Folded Bit Line - We use a folded bit line design which reduces word line to bit line coupling noise by turning it into a common mode signal at the sense amplifier. A bit cell on an even row (addresses 0,2,4, etc.) has its read bit line connected to *rbl0* and odd rows are connected to *rbl1*.

C. Bit Line Equalization - Before the sensing operation starts, we need to equalize the voltages on both *rbl0* and *rbl1* to ensure that both sense amp inputs start at the same point.

D. Sense Amplifier Output Equalization - Our choice of sense amplifier dictates that we must equalize the outputs of the sense amplifier as well, so that only a differential input will switch the sense amplifier outputs.

E. Output Sense Correction - Because of the folded bit line design, accessing an odd row of the array necessitates flipping the dual-rail outputs of the read operation, which we achieve by using pass transistor logic. The *flip* signal can thus be generated by the low order address bit.

3 Asynchronous Control

Now we present the self-timed interface to the DRAM core, followed by the banking circuitry. The asynchronous design techniques we use generate circuits that are quasi-delay-insensitive(QDI) [8]. This circuit methodology assumes that gates have arbitrary delay and only makes relatively weak timing assumption on the presence of isochronic forks. We use the Martin synthesis method [9] that was used successfully on the design of an asynchronous MIPS R3000 (MiniMIPS [10]). This proposed on-chip DRAM system leverages the design of the buses used in the MiniMIPS SRAM cache [11].

3.1 Core CHP Representation

We would like to surround the previously described analog DRAM core with control circuitry as presented in Fig. 6. *Addr*, *Ctrl* and *Di* are the input channels to the core that

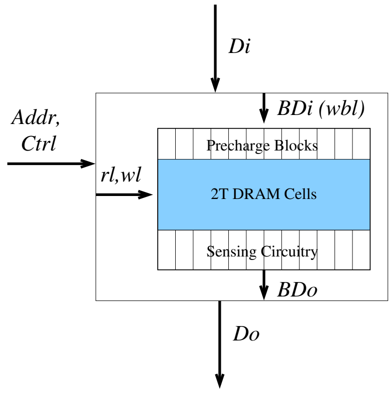

Figure 6. Encapsulating the analog DRAM core with control circuitry (refresh path not shown)

communicate the row access number, the type of operation (read, write or refresh), and the write data respectively. On a read, output channel *Do* communicates the stored data to

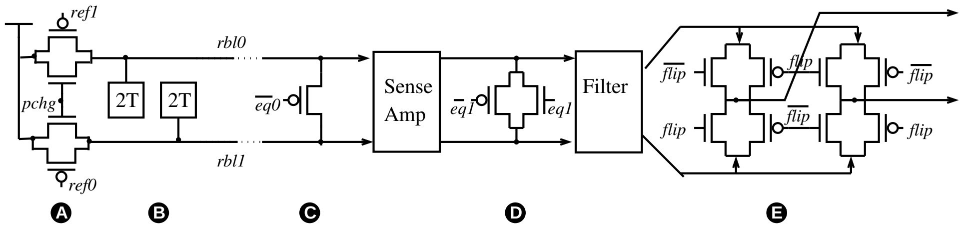

Figure 5. A bit line column organization

the environment. *BDi* represents the write bit lines and *BDo* the data output from the analog core.

The synthesis method allows us describe circuits algorithmically using CHP (Communicating Hardware Processes), whose syntax is described in the Appendix. The following CHP program describes the simplest set of operations executed by a core DRAM bank:

$$
\begin{array}{l}
\mathit{core} \equiv \\
\quad *[\mathit{Addr}?a,\ \ \mathit{Ctrl}?c; \\
\quad\quad \mathit{decode}(a, \mathit{goR}, \mathit{goW}), \\
\quad\quad [c ='' \mathit{read}'' \longrightarrow\ \ \mathit{goR}\uparrow; \mathit{BDo}?d; \mathit{Do}!d \\
\quad\quad [\!]\, c ='' \mathit{write}'' \longrightarrow\ \ \mathit{goW}\uparrow; \mathit{Di}?d; \mathit{BDi}!d; \\
\quad\quad [\!]\, c ='' \mathit{refresh}'' \longrightarrow\ \ \mathit{goR}\uparrow; \mathit{BDo}?d; \mathit{RDo}!d; \\
\quad\quad\quad\quad\quad\quad\quad\quad \mathit{RDo}?d; \mathit{goW}\uparrow; \mathit{BDi}!d \\
\quad\quad]; \\
\quad\quad \mathit{goR}\downarrow,\ \ \mathit{goW}\downarrow; \\
\quad]
\end{array}
$$

We have introduced the channel *RDo* to represent the internal refresh bus, and the process *decode* which represent the address decoders that drive the individual read and write selects going to the core. The signals *goR* and *goW* are generated internally specifically to support the refresh operation, which requires the address decoders to drive both the read and write lines during a single cycle.

We use a standard precharge full buffering template similar to that used in the MiniMIPS to implement the control circuitry. We use 1-of-4 signalling on the address bus and externally on the data inputs and outputs, while the internal data read and write lines use dual-rail signalling. The full details of the decomposition can be found in [12], and a block diagram of the QDI circuitry can be seen in Fig. 7. The blocks labelled *C1* through *C5* represent the completion circuitry necessary to acknowledge transitions in a QDI design. These completion blocks can consume a significant layout area, so in our final design we make a few optimizations to reduce the circuitry.

We note that the completion of the input channel *Di* can use the *output completion* signal from the bus without violating QDI, a well known technique [13] that allows us to remove *C1*. Next we consider the output stage – We have one completion stage (*C3*) that directly computes the validity and neutrality (reset) of the data coming from the core (after the sense-amp and filter circuitry), and thus cannot be eliminated. We however note that the validity and neutrality check circuitry across this data consist of a NOR gate per 1-of-2 bit, and then a tree of C-elements, which takes many transitions.

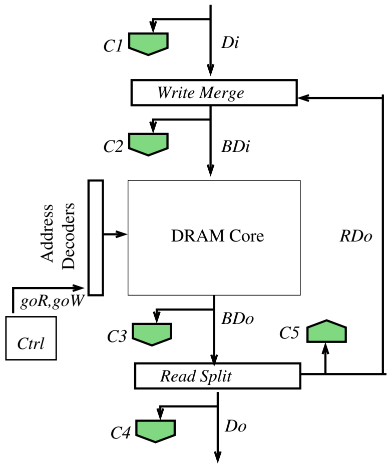

Figure 7. Control Circuitry

We are left with the two completion blocks,*C4* and *C5*, that check the completion of the outputs of the split. However, the conditional split itself takes only *two* transitions to generate, and is almost certainly guaranteed to complete before *C3* finishes computing assuming we do not excessively load the outputs of the split, and the control inputs are not delayed. Thus, we can use the output of *C3* as our validity signal for signaling the completion of the split output. With careful circuit design, this small relative timing assumption between gates allows us to eliminate two extra completion blocks, and the optimized block diagram is shown in Fig. 8.

We thus far have interfaced the analog memory core with self-timed circuitry that responds to $*[\mathit{Addr}?a, \mathit{Ctrl}?c; \mathit{Do}!d]$, $*[\mathit{Addr}?a, \mathit{Ctrl}?c, \mathit{Din}?d]$, and $*[\mathit{Addr}?a, \mathit{Ctrl}?c]$ corresponding to reads, writes and refreshes.

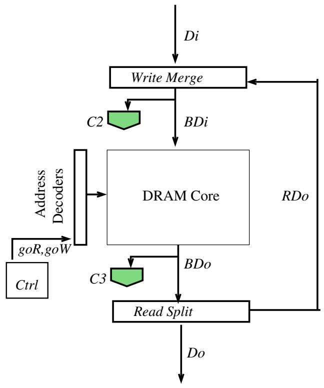

Figure 8. DRAM Bank with Reduced Completion Blocks

3.2 Buses for Interleaved Bank Access

We now present the design of the buses that enable interleaved access to the memory banks. Our banking method uses a two-level split and merge in a tree architecture with a branching factor of four, as shown in Fig. 9.

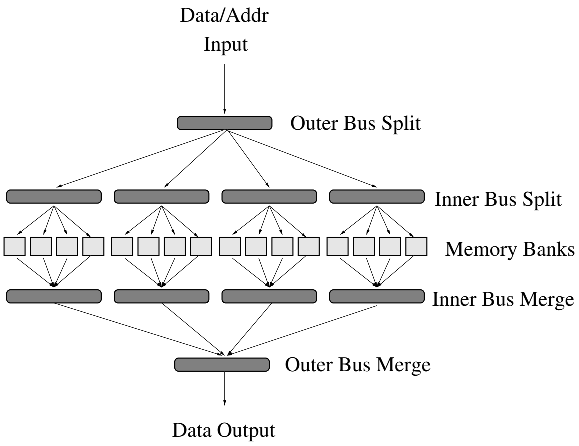

Figure 9. Banking with a two level bus (16 banks)

3.2.1 Inner Bus Specification

The novel feature of this bus, which interleaves four DRAM banks, is the incorporation of the refresh and also a simultaneous broadcast of the refresh command to all four banks. This reduces the frequency of the refresh command (one refresh per line per period yields 1/64 refreshes per period versus 1/256 refreshes if each bank were separately refreshed). It also has the practical effect of making the refresh invisible to the control at higher levels of the banking scheme.

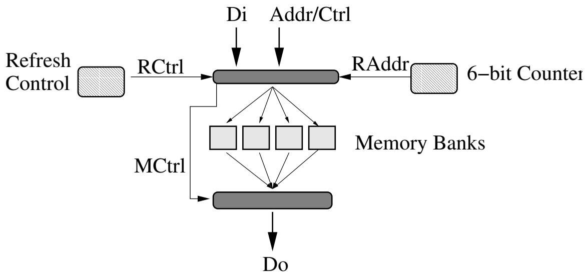

Figure 10. Decomposition of the inner bus to support refresh

The CHP specification for this bus is shown below, and a block diagram is shown in Fig. 10:

$$
\begin{array}{l}
SBUS0 \equiv \\
\quad *[RCtrl?c; \\
\quad\quad [c =' refresh' \longrightarrow\ < i : 4 : BCtrl[i]!c, \\
\quad\quad\quad\quad\quad\quad BAddr[bank]!Raddr? > \\
\quad\quad [] else \longrightarrow\ Addr?(bank, a); \\
\quad\quad\quad\quad BCtrl[bank]!c,\ \ BAddr[bank]!a \\
\quad\quad], \\
\quad\quad [c =' write' \longrightarrow\ Di?d; BDi[bank]!d \\
\quad\quad [] c =' read' \longrightarrow\ MCtrl!bank \\
\quad\quad [] c =' refresh' \longrightarrow\ skip \\
\quad\quad] \\
\quad] \\
MBUS0 \equiv *[MCtrl?bank;\ \ BDo[bank]?d;\ \ Do!d]
\end{array}
$$

We introduce a new channel *RCtrl* that communicates the read/write/refresh control, and *Raddr* which communicates the address to be refreshed; if a refresh is pending then the bus broadcasts the refresh to all four banks, otherwise it executes a normal read/write.

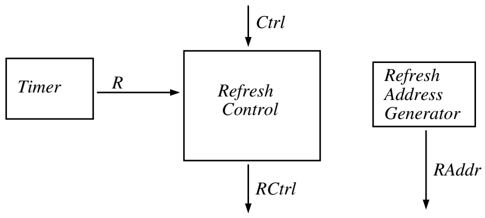

Figure 11. Decomposition of Refresh Control

The most interesting aspect of the inner bus is the implementation of the refresh control (channel *RCtrl*. A conceptual diagram of the decomposition is shown in Fig. 11 with a timer process, control arbitrator and address generator. The timer process generates a periodic synchronization request on channel *R*. The refresh control arbitrates between this refresh request and the external read/write control and provides the required refresh address via a simple counter incorporated in the refresh address generator.

Generating a correct refresh signal with a relatively long (on the order of milliseconds) period can be complex in an asynchronous design. We implemented the *R* refresh request with the use of the following two processes:

$$\begin{array}{l} Src \equiv\ *[\ D!\ ] \\ \quad\quad \| \\ Req \equiv\ *[\ D?;\ \ R!\ ] \end{array}$$

We implement the request line of *D* with a delay element, and process *Req* interfaces with a normal 4-phase handshake as shown below:

$$*[[di];do\uparrow;[\neg di];ro\uparrow;[ri];do\downarrow;ro\downarrow;[ri]]$$

The important point to note is that *Req* waits for both the up going and down going transitions on *di* before initiating a request on *R*. The circuit implementing this handshake is shown in Fig. 12. The nice part about this circuit is that we can use an *arbitrary* delay element on *di* and not affect the rest of the system. One could use inverter chains, or RC delay elements, although we decided to use a low power CMOS delay element proposed by Kim et al[14].

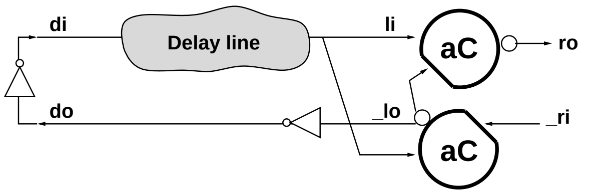

Figure 12. Interfacing a delay line with a four-phase handshake

This refresh request channel is then fed into the control arbitration unit shown in Fig. 13 (A full decomposition can be found in [12]). This circuit arbitrates between the external read or write (*Ctrl*) and the refresh request (*R*), and selects one via the *Ref* channel. The following process then provides the control to the banks:

$$\begin{array}{l} *[REF?r; \\ \quad [r = 1 \longrightarrow \quad RCtrl!'refresh' \\ \quad [\!]\, r = 0 \longrightarrow \quad RCtrl!(Ctrl?) \\ \quad]] \end{array}$$

Although there is a finite probability of the refresh request taking too long to resolve due to metastability in the arbiter, having a retention time margin of milliseconds reduces this risk of failure.

The remaining parts of the bus were implemented using standard QDI precharge buffering pipeline stages.

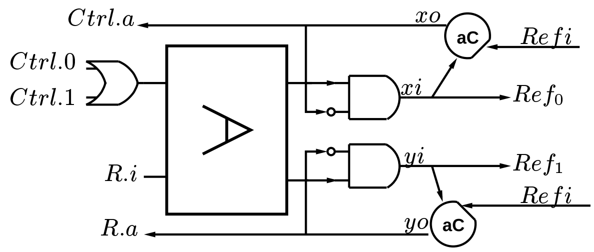

Figure 13. Refresh Selection with Arbiter

3.2.2 Outer Bus Specification

The design of this bus looks exactly like the implementation of the inner bank, except we have no refresh control circuitry. Because of this, we can obtain a slightly higher throughput, even though we are working with a larger address datapath. The CHP specification of this bus is shown below, and was implemented using the same style of buffering stages as the inner bus:

$$\begin{array}{l} SBUS1 \equiv *[Ctrl?c, Addr?a; \\ \quad\quad\quad [c =' write' \longrightarrow \quad Di?d; BDi[bank]!d \\ \quad\quad\quad [\!]\, c =' read' \longrightarrow \quad MCtrl!bank \\ \quad\quad\quad]] \\ MBUS1 \equiv *[MCtrl?bank;\ \ BDo[bank]?d;\ \ Do!d] \end{array}$$

4 Simulation Results

4.1 Bank Operation

The core bank and asynchronous interface layout was done using Magic. Banks with a fixed number of 64 rows and bit line widths (number of columns) of 32, 64, 128 and 256 bits were designed yielding bank sizes of 0.25, 0.5, 1, and 2 Kbytes respectively.

The circuits were simulated using `aspice` with TSMC 0.25μm CMOS process parameters at 25^{o}C. A typical read waveform for a single bank gathered using the circuit simulator is shown in Fig. 14.

Figs. 15 and 16 show the throughput and latency variation based on supply voltage for a bank containing 64 rows of 32 bits each. We show correct operation for supply voltages down to 1.7V (nominal for our process is 2.5V).

The read latency measures the time taken for a precharged bank to generate data bits at the output after the address and read control signals are presented to the bank. The cycle time refers to the time between consecutive operations to the bank. We show sub-nanosecond read latency at 2.5V for this bank structure, which is very low compared to other embedded DRAM structures. The read cycle time is very good for a DRAM structure at sub 5ns as well (with the completion circuitry occupying about 25% of that), with

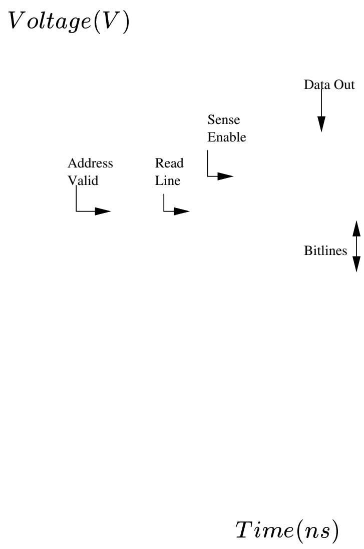

Figure 14. Typical simulated DRAM Read waveform

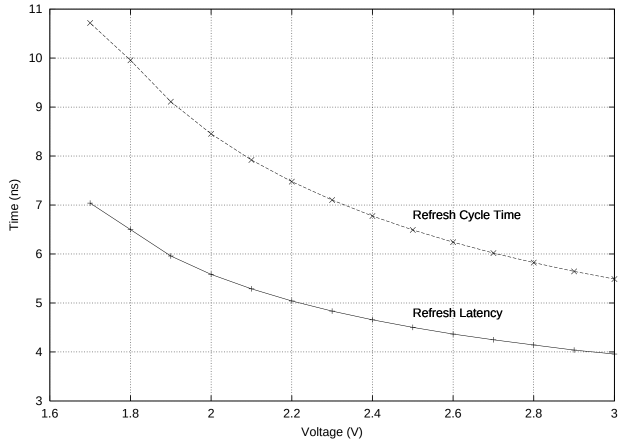

Figure 16. 64x32 DRAM bank refresh operation with varying supply voltage

typical cycles times in the 6ns range [4]. At the nominal 2.5V, the read power consumption is on the order of 11mW, and the write power consumption is on the order of 6mW.

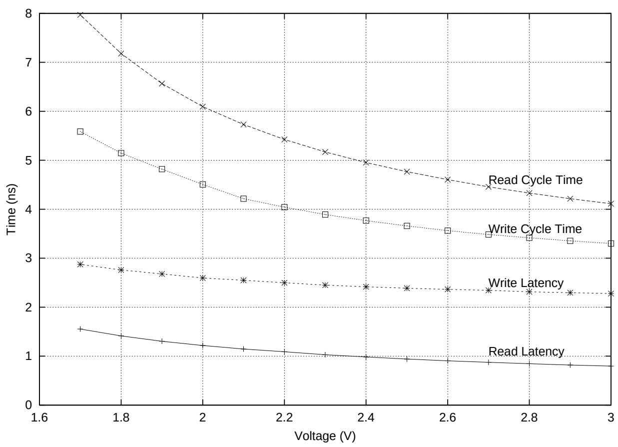

Figure 15. 64x32 DRAM bank read/write operation with varying supply voltage

4.2 Scaling Trends

Table 1 shows the effects of increasing the width of the bank datapath while keeping the number of rows fixed at 64 (to limit the capacitance on the bit lines), up to a bank size of 2 Kbytes. Although we only show the read timing variation, the write and refresh timings show similar trends. The latency increases are due to increased RC delays on the word lines, and the cycle time increases occur due to increased completion overhead.

In terms of area, the layout for a 2KB bank is 48% smaller than an equivalent SRAM memory bank (compared against the MiniMIPS[11] cache).

4.3 Retention Times

The individual 2T cells were designed to have between 30 and 40fF of storage capacitance. A graph showing the estimated retention times for a memory cell is shown in Figure 17. We estimate about a 1.5 pA leakage current [15, 12] which gives us retention times on the order of milliseconds.

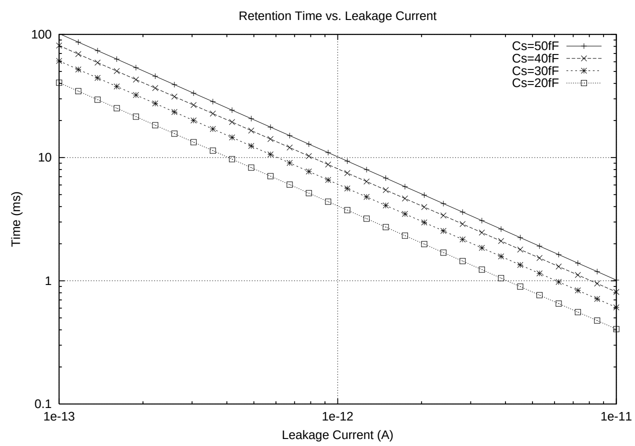

Figure 17. Log-log plot of projected retention times for the 2T cell in a 0.25μ process for varying storage capacitance (C_s)

Bits/Row (Bank Size)	Read Latency(ns)	Read Cycle Time(ns)	Area λ^2 (Normalized)
32 (256B)	0.94	4.78	4,528,811 (1.0)
64 (512B)	1.18	5.64	7,445,748 (1.6)
128(1KB)	1.41	6.85	13,447,225 (2.9)
256(2KB)	1.86	8.63	25,265,818 (5.6)

Table 1. Spice simulation results for different DRAM bank sizes

4.4 Bus performance

The inner bus has enough buffering (pipelining) to enable successive operations to different banks to proceed while the previous operation is still resetting. This enables us to gain a throughput of slightly over 522MHz with an access pattern of consecutive reads/writes that do not access the same bank, using a 32bit datapath. We predict only slight performance hits as we scale the data path wider, due to the byte-sliced pipelined vertical datapath. The two transition latency through the split, and an equivalent delay through the merge adds about 0.4ns latency. This inner bus consumes energy at a rate of 112mW during reads and 70mW during writes, and adds less than 10% overhead to the layout area of 4 banks.

Spice simulations for the outer bus show a throughput of 551MHz during continuous reads to accesses to different inner buses, at 210mW power. It writes at a throughput of 540MHz with 180mW.

5 Architectural Performance

We have performed a simple architectural analysis to validate the average case performance of the banking scheme. Consecutive accesses to the same bank will lead to the memory operating at the cycle time of an individual bank, something we would like to avoid. In this section, we present some results that show that this worst-case scenario is rare enough during standard benchmarks to ensure little performance degradation.

We used `atom`, a binary instrumentation package for the Alpha to look at memory access patterns for some standard benchmarks. Specifically, we studied the spacing between consecutive accesses to the same memory bank in terms of number of instructions, which gives us some idea, at least in an in-order issue processor, of the way a first level cache would be accessed. For this study, we ran a subset of the SPEC2000int benchmarks (`256.bzip2, 252.eon, 176.gcc, 164.gzip, 181.mcf, 197.parser, 300.twolf, 175.vpr`) using the training set data and averaged across them.

The least significant bits of the memory address was used to select the bank. We vary the number of banks used and keep the bank size fixed at 1KB with 64 rows of 128 bit lines. The results can be seen in Fig. 18 and Fig. 19 for instruction and data memory respectively. For instructions, with 16 banks the occurrence of consecutive accesses to the same bank were well under 1%, while the data memory shows this worst-case access pattern only about 1% of the time. This shows that we can hide the effects of long DRAM cycle times by using asynchronous banking schemes and achieve excellent average case performance.

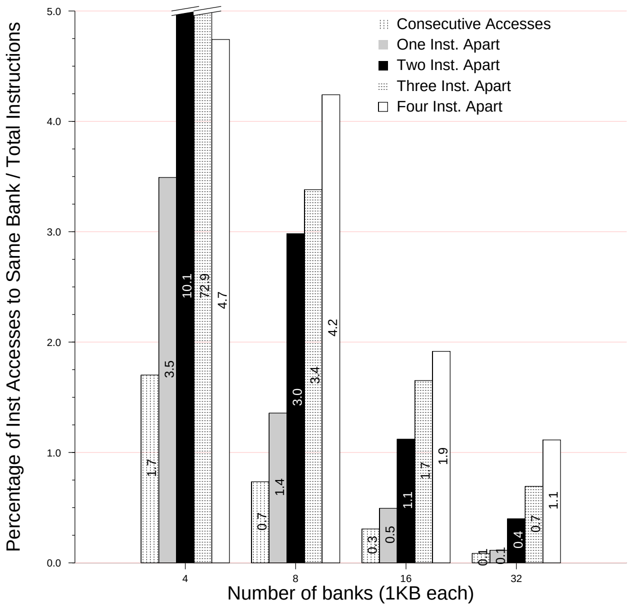

Figure 18. Instruction memory access patterns

6 Related Work

Recent research into embedded DRAM have mainly utilized ASIC processes that merge normal logic capability with elements such as trench capacitors for embedding memory. In 2001, Tomishima et al[16] demonstrated a 4MB memory with a datapath of 16bits. It had a burst access cycle time of 4.3ns with an initial column access latency of about 10ns. Random access cycle times were quoted at 17.5ns. The smallest active bank had 8K rows of 16 bits; the

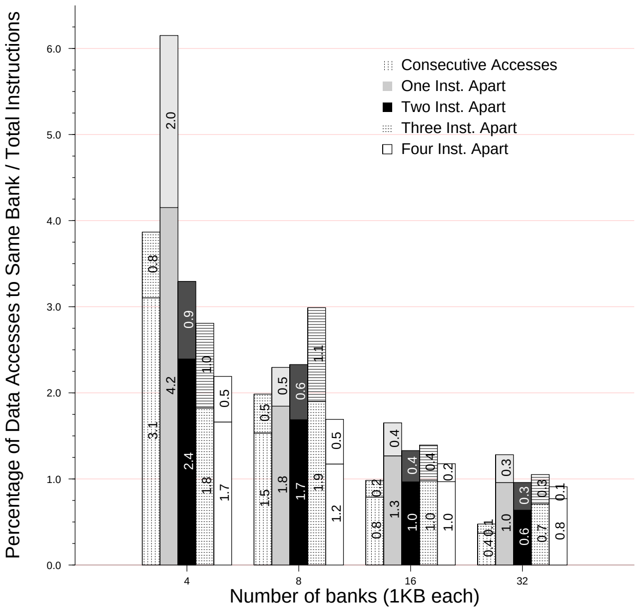

Figure 19. Data memory access patterns. Upper segments represent stores, while lower segments represent loads.

long bit lines probably account for their slow cycle times. More recently, in 2002, a DRAM was demonstrated at IBM [17] with a 6.6ns random access cycle time in 0.12um process. This DRAM was organized into 1MB blocks with sub-partitions of 512 rows by 256 bits. Notice how the shortening of the bit lines almost directly corresponds to the faster cycle times. Our design comfortably outperforms these numbers in a standard logic process, albeit at a lower density (2x over SRAM versus 4x for the IBM DRAM).

The trend towards using many banks with short bit lines has appeared most notably in the context of two recent designs. MoSys (not to be confused with Mosis, the chip fabricator) licenses what they call 1T SRAM, which is just a 1T DRAM cell based memory with a SRAM cache to hide refreshes. They use the same TSMC 0.25μm process we target, and achieve a 6ns (166Mhz) cycle time, with small banks of 1024 rows of 32 bits each in their smallest configuration [5]. Our memory core has a faster cycle time (under 5ns), and we need minimal circuitry to hide the refresh externally.

The state of the art in embedded DRAM currently lies with another research project at IBM, which recently (summer 2002) announced a new architecture for the 1T embedded DRAM called destructive read[4]. This novel design does not refresh the contents of the cells on a read; instead, the result is saved in a SRAM memory they call the writeback buffer (similar to the SRAM cache in the MoSys design). The destructive read allows them to use a single-ended direct sensing scheme, instead of a differential sense-amp pair to refresh the cell contents. This results in faster sensing and smaller bitline voltage swings. They also used small banks of 256 rows by 128 bits. On a 0.13um ASIC process they were able to achieve a cycle time of 2.9ns and about a 1.8ns latency in each core bank. Our core latency is shorter, and although our core cycle time is larger, with our banked scheme we can achieve faster average case cycle times.

Our design also compares favorably with SRAM memories. The MiniMIPS QDI asynchronous SRAM cache had about a 2ns bank latency and a 5.7ns cycle time for reads, in a 0.6μm process using 64 rows of 48 bits. IBM's Cu-11 ASIC 0.13u process [18], which is typical of industry offerings, provides a single-ported SRAM macro that offers a 1.2ns latency and 1.25ns cycle time for a 16KB bank with width of 32bits. Although DRAM cannot compete with the core cycle time of SRAM, the interleaved banking can effectively hide this limiting factor, and combined with the low latency approach of our design, makes this DRAM suitable for on-chip memories where fast access times are needed.

7 Summary

We have presented the design of DRAM memory cores featuring sub-nanosecond latency for a 4 byte datapath. In addition, we show that long cycle times can be alleviated through the use of a pipelined multi-level banking scheme that can interleave memory accesses to multiple banks. We have presented the design of novel bus system that incorporates the refresh mechanism and hides the most cumbersome aspect of using DRAM, and the memory for all purposes appears like SRAM externally. Although the memory timing is dependent on the access pattern, the use of QDI asynchronous design techniques allows us to avoid adding complexity to deal with the variable latency, and we get true average case performance. The use of such an embedded DRAM combined with asynchronous logic should facilitate a simple method of increasing sizes of on-chip cache given the finite processor die space available.

8 Acknowledgments

This work was supported in part by the Multidisciplinary University Research Initiative (MURI) under the Office of Naval Research Contract N00014-00-1-0564, and in part by a National Science Foundation CAREER award under contract CCR 9984299.

A Summary of CHP Notation

The CHP notation we use is based on Hoare's CSP [19]. A full description CHP and its semantics can be found in [9]. What follows is a short and informal description.

- Assignment: $a := b$. This statement means "assign the value of b to a." We also write $a\uparrow$ for $a := true$, and $a\downarrow$ for $a := false$.
- Selection: $[G1 \rightarrow S1 \,[]\, ... \,[]\, Gn \rightarrow Sn]$, where Gi's are boolean expressions (guards) and Si's are program parts. The execution of this command corresponds to waiting until one of the guards is $true$, and then executing one of the statements with a $true$ guard. The notation $[G]$ is short-hand for $[G \rightarrow skip]$, and denotes waiting for the predicate G to become true. If the guards are not mutually exclusive, we use the vertical bar "|" instead of "[]."
- Repetition: $*[G1 \rightarrow S1 \,[]\, ... \,[]\, Gn \rightarrow Sn]$. The execution of this command corresponds to choosing one of the $true$ guards and executing the corresponding statement, repeating this until all guards evaluate to $false$. The notation $*[S]$ is short-hand for $*[true \rightarrow S]$.
- Send: $X!e$ means send the value of e over channel X.
- Receive: $Y?v$ means receive a value over channel Y and store it in variable v.
- Probe: The boolean expression $\overline{X}$ is $true$ iff a communication over channel X can complete without suspending.
- Sequential Composition: $S; T$
- Parallel Composition: $S \parallel T$ or S, T.
- Simultaneous Composition: $S \bullet T$ both S and T are communication actions and they complete simultaneously.

References

[1] J. Poulton. An embedded DRAM for CMOS ASICs. In *Proc. of the 17th Conf. on Advanced Research in VLSI*, pages 288–302, 1997.

[2] D. Speck. The mosaic fast 512k scalable CMOS DRAM. In *Proceedings of Conference on Advanced Research in VLSI*, pages 229–244, 1991.

[3] Kunle Olukotun Tadaaki Tamauchi, Lance Hammond. The hierarchical multi-bank DRAM: A high-performance architecture for memory integrated with processors. In *Proc. of the 17th Conf. on Advanced Research in VLSI*, pages 303–319, 1997.

[4] Chomg-Lii Hwang et al. A 2.9ns random access cycle embedded DRAM with a destructive-read architecture. In *Symposium on VLSI Circuits Digest of Technical Papers*, pages 174–175, 2002.

[5] MoSys. 1t SRAM. `http://www.mosys.com`.

[6] R. Manohar and M. Heinrich. A case for asynchronous active memories, 2000.

[7] J.M. Rabaey. *Digital Integrated Circuits: A Design Perspective*. Prentice-Hall, 1996.

[8] Alain J. Martin. The limitations to delay-insensitivity in asynchronous circuits. In *Sixth MIT Conference on Advanced Researh in VLSI*, 1990.

[9] Alain J. Martin. Synthesis of asynchronous VLSI circuits. In J. Straunstrup, editor, *Formal Methods for VLSI Design*, pages 237–283. North-Holland, 1990.

[10] Alain J. Martin, Andrew Lines, Rajit Manohar, Mika Nystroem, Paul Penzes, Robert Southworth, and Uri Cummings. The design of an asynchronous MIPS R3000 microprocessor. In *Advanced Research in VLSI*, pages 164–181, 1997.

[11] Alain J. Martin Andrew Lines, Mika Nystrom. A pipelined asynchronous cache system. Internal Report.

[12] Virantha Ekanayake. Asynchronous dynamic random access memories. Master's thesis, Cornell University, Ithaca, New York, 2002.

[13] R. Ozdag and P.A. Beerel. High-speed QDI asynchronous pipelines. In *Eight Internation Symposium on Asynchronous Circuits and Systems*, pages 13–22, 2002.

[14] Gyudong Kim, Min-Kyu Kim, Byoung-Soo Chang, and Wonchan Kim. A low-voltage, low-power CMOS delay element. *IEEE Journal of Solid-State Circuits*, 31:966–971, July 1996.

[15] Doyle et al. Transistor elements for 30nm physical gate lengths and beyond. *Intel Technology Journal*, 6, May 2002.

[16] Shigeki Tomishima et al. A 1.0v 230mhz column-access embedded DRAM macro for portable MPEG applications. In *IEEE International Solid-State Circuits Conference*, pages 384–385,469, 2001.

[17] John Barth, Darren Anand, Jeff Dreibelbis, and Erik Nelson. A 300 mhz multi-banked eDRAM macro featuring GND sense, bit-line twisting and direct reference cell write. In *IEEE International Solid-State Circuits Conference*, page 9.3, 2002.

[18] IBM Research. IBM Cu-11 embedded DRAM macro datasheet. `http://www-3.ibm.com/chips/techlib/techlib.nsf/products/Embedded_DRAM_C%u-11_Macro`, July 2002.

[19] C.A.R. Hoare. Communicating sequential processes. *Communications of the ACM*, pages 666–677, 1978.