QAN6
Page Mode DRAM Controller for 486DX2

1.0 SUMMARY

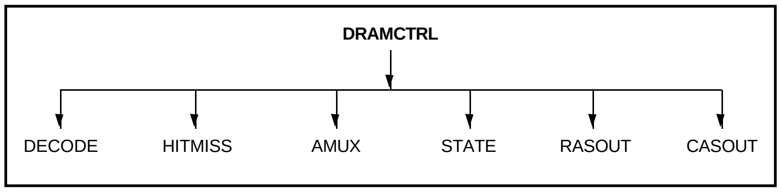
The Intel 486™ DX microprocessor is one of today's most advanced microprocessors. In addition to its popularity in personal computer applications it is increasingly chosen as the host in a wide variety of workstations.

Today, many designs are migrating to the Intel 486DX2, which is compatible with the 486DX, yet offers an internal clock rate at twice the external clock rate. These processors continue to integrate more and more functionality onto the chip to improve performance and decrease system form factors. However, because of the additional pin count and the continually changing DRAM market, DRAM controllers are typically not integrated with the processor, leaving the design up to the system engineer.

Interfaces to 66 MHz 486DX2 microprocessor

This application note presents an example of a high-performance page-mode DRAM controller implemented in a QuickLogic QL12x16 FPGA which interfaces to a 66 MHz 486DX2 microprocessor. The function integrates the address decoding and multiplexing, page hit/miss detection, a basic controller state machine, and the RAS/CAS output logic into a TQFP (Thin Quad Flat Pack) or an 84-pin PLCC package. Designs of this complexity and speed will typically require up to fifteen high-speed 22V10 PLD packages.

DRAMCTRL Major Function Blocks



The design illustrates the critical paths generally associated with a high-frequency DRAM controller and shows the advantages of the QuickLogic pASIC® 1 architecture in serving this class of applications. It was created using the QuickLogic pASIC Toolkit based on the Data I/O ECS schematic entry package. Copies of the detailed design schematics are available on request from QuickLogic (QS-QAN6). All timing delays quoted are worst case values for the 1.0 micron, QL12x16-0 device over the commercial operating range. QuickLogic's latest .65 micron QL12x16B exhibits significantly faster timing (20% - 30% faster).

QUICKLOGIC

2.0 BRIEF DRAM OVERVIEW

The acronym 'DRAM' stands for Dynamic Random Access Memory. The term 'dynamic' comes from the internal implementation of each bit storage cell. This type of cell uses capacitance to store the bit value instead of combinatorial feedback as is used in a static type latch. Using a capacitance to store the charge allows for a minimum number of transistors to achieve the highest possible density. By keeping the transistor count to a minimum, DRAMs can provide the lowest cost per bit in a random access memory. There is, however a price to pay for this low-cost solution.

Since total isolation from leakage current is not possible, charge stored on a bit cell capacitance will leak off over time. For this reason the charge on the capacitance must be restored to a full value periodically. This is known as refreshing the DRAM and some method of refreshing must be supported in the system design.

DRAMs have a second unique feature used to reduce overall cost and that is the use of a multiplexed address bus. Typical DRAMs today require on the order of 20 or more address lines. This, coupled with data lines, would require a fairly large package — increasing package cost, board space, and ultimately board cost, DRAMs use two strobe signals (i.e., RAS and CAS) to latch in the address. Half of the address is first latched using the RAS signal — this is known as latching the row. The second half of the address is latched using the CAS signal — this is known as latching the column.

There are a variety of timing specs associated with the row and column strobes that must be met for proper operation. The most common specs referred to are tRP, tRAC and tCAC. These represent specs for the RAS precharge time, RAS access time and CAS access time. The RAS access time is the time it takes for data to become valid from the RAS edge. This spec is normally used as the designator for the speed of the DRAM and is typically the most difficult timing to meet.

The CAS access time is also a data valid time, but from the CAS edge rather than the RAS edge. Both specs must be met before data will be valid. Current DRAMs typically have a tCAC value that is about one-fourth the tRAC value.

Due to the ever increasing need to always have faster access times and higher performance, DRAM manufacturers have come up with many different modes that help to improve the access time for certain applications. One of the most common modes used today is page mode. Page mode allows successive addresses to the same page to keep RAS active while the data access time is totally dependent on tCAC (and some address timing specs) for subsequent cycles. Thus, each cycle following the leadoff cycle is able to execute at a much higher rate. However, the logic to implement a page mode controller is significantly more complicated, as it requires keeping a latched copy of the previous page address and comparing it to subsequent cycles. When the page does not match, the RAS output must be precharged as defined by the tRP spec. Once tRP has been met, a new cycle with a new page may be started. The end result is a slower cycle for page misses and a faster cycle for page hits. Generally, the use of page mode gives an overall increase in performance.

In addition to the many modes available in DRAMs, today there exist many different internal configurations that may use asymmetrical combinations of row and column address bits. This makes support for multiple DRAMs even more difficult. Additionally, DRAMs come in an assortment of data out bits and the number of RAS, CAS, and WE signals it may need. All of these variations can make for a very complex design when designing a DRAM controller.

3.0 DESIGN OBJECTIVE

The majority of system designs using an Intel 486DX2 require large amounts of local DRAM memory, and thus, some sort of DRAM cycle controller. The performance of this controller is of primary concern since it will greatly affect the overall system performance. Additionally, board space that the controller takes up is also of concern. It is no longer acceptable to use discrete devices of small integration in order to achieve high performance. A single-chip solution is almost always required and generally includes additional board-level functions.

The objective of this design is to provide the highest performance DRAM interface for a 486DX2 at 33 MHz while using minimal board space and readily available DRAMs. The desired specifications for this controller are:

- Support 486DX2 burst mode cycles of 5-2-2-2
- Staggered refresh cycles using CAS before RAS refresh mode
- Bank size configurability via mode select pins
- CAS wait state selection via mode select pin giving 5-3-3-3
- Fast page mode cycle support
- DRAM select decode with range programmability

4.0 DESCRIPTION OF DESIGN PROBLEM

Speed is always the primary concern when designing DRAM controllers. With today's microprocessors running at 33 MHz and higher, circuit delays of a few nanoseconds may mean additional wait states to the processor access. Additionally, market requirements typically dictate support for multiple DRAM sizes, organizations, and speed. Designing a DRAM controller to accommodate these needs directly translates into additional levels of logic and additional delays.

The design presented in this application note supports two different sizes of banks that can be mixed across the four banks. The major problem areas and critical speed paths in a design of this type are typically:

- Bank decode logic
- Row/Column address mux delay
- Page compare logic and cycle start delay
- Output logic delay and variation

Bank Decode Delay

The bank decode is by far the most complicated section of logic. This may seem surprising, since in the past, it has typically been the simplest part of the design. No longer is the bank decode simply a 2-to-4 decoder of two address lines. If you choose to support different sizes of banks simultaneously, the address decode gets significantly more complicated.

The design presented here supports two sizes of banks simultaneously, provided the larger memory is mapped to the smallest address possible. Support for any combination of banks becomes even more difficult, but still possible. The possible configurations reduce down into five distinct cases. Within each case, it is possible to have additional cases that do not have all banks populated. Table 1 shows the configurations supported in this application note in more detail.

Note that the decode uses four address bits, and which bits are used is dependent upon the configuration.

TABLE 1
Bank Decode Table

CASE	ADDRESS				BANK NUMBER				RANGE SIZE (MEG)
	25	24	23	22	0	1	2	3	
1	–	–	0	0	4				4, 8, 12, 16
	–	–	0	1		4			
	–	–	1	0			4		
	–	–	1	1				4	
2	–	0	X	X	16				16, 20, 24, 28
	–	1	0	0		4			
	–	1	0	1			4		
	–	1	1	0				4	
3	0	0	X	X	16				16, 32, 36, 40
	0	1	X	X		16			
	1	0	0	0			4		
	1	0	0	1				4	
4	0	0	X	X	16				16, 32, 48, 52
	0	1	X	X		16			
	1	0	X	X			16		
	1	1	0	0				4	
5	0	0	X	X	16				16, 32, 48, 64
	0	1	X	X		16			
	1	0	X	X			16		
	1	1	X	X				16	

4.2 Row/Column Address Mux Delay

The row/column address mux directly affects how early the address can be driven to the DRAM. The fastest and simplest 2-to-1 mux is generally used to switch between row and column. Any additional delay on the address mux causes both the row and column address to be delayed. This, in turn, translates to either delaying RAS and CAS (adding additional wait states) or a violation of address setup times to RAS and CAS.

Support for multiple DRAM sizes and organizations presents yet another problem that requires further levels of logic in the address mux. This is due to the fact that different sized DRAMs will require different address bits in the row address. This can be observed by first looking at the configuration for a 1M bit DRAM (256Kx4) organized in the system as 256Kx32. In this configuration, a depth of 256K would need to be addressed by the row and column address. The width of 32 would be addressed by 4-byte enables.

Addressing of 256K requires 18 address lines — generally nine row bits and nine column bits. If we are designing the address mux (MA[0:10]) around this specific requirement, we would have the assignments shown in the first row of Table 2.

TABLE 2 Row Address Map

DRAM DEPTH	ROW/ COLUMN	10	9	8	7	6	5	4	3	2	1	0
256K	9x9	22	20	19	18	17	16	15	14	13	12	11
1M	10x10	22	20	19	18	17	16	15	14	13	12	21

Since A1 and A0 are used up in the byte selection, addressing to the DRAM starts at A2 and goes up to A19. Note that the highest order bit of the column is A10, with the row starting at A11. To support a DRAM of depth 512K (organized 10x9), A20 simply needs to be driven on an additional row address — MA9. However, to support a depth of one meg (organized 10x10), A21 needs to be driven on either the row or column. The easiest solution would be to add it to the column address on MA9. However, this would fragment the page of the DRAM into two halves, lowering the page mode performance.

In order to keep the page contiguous, A21 must be driven in the row address. This is accomplished by replacing A11 on the row address with A21, and driving A11 on the unused column address. As such, the address mux for MA0 now becomes a 3-to-1 mux selecting between A2, A11, and now A21. As mentioned previously, if the delay incurred for each additional mux input is too large, the cycle will require an additional wait state.

4.3 Page Compare and Cycle Detect Delays

The start of page mode DRAM cycle is typically gated by:

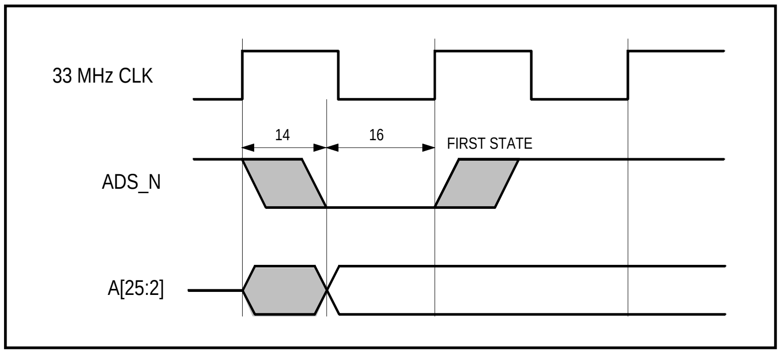
- DRAM address range decode select
- Page hit/miss detection

The DRAM address range decode determines if the address of the cycle goes to DRAM or some other device. This decode can range from simply matching upper address lines to 0, or actually comparing an address between two registers depending upon the granularity of the banks and the combinations of bank population. Both mechanisms typically require several levels of logic to implement.

The page hit/miss detection determines whether the page of the pending cycle is equal to the page of the previous cycle. If the page value is different, then the row of the DRAM will need to be precharged. This typically consists of a comparator implemented as several levels of XNOR gates. Both the address range decode and the page hit/miss detection depend upon a valid address from the processor. The total delay before starting the cycle will then be the maximum sum of the processor address delay with either the range decode delay or the page detect delay.

The processor address delay is given in the 33 MHz data sheet to be 14 ns. In order for the state machine to start the DRAM cycle on the first following clock edge, both the range detect and the page detect must be performed and allow for a setup time to a latch in less than 16 ns (see Figure 1). Performing XOR combinations can be quite costly in terms of delay and depend upon the number of address bits needing comparison.

FIGURE 1
Cycle Detect Delays



4.4 Output Logic Delays and Variation

There is typically very little margin for variation among the output pin timings of a DRAM controller. Today's DRAM timing specifications are all done with respect to each other, i.e., pin-to-pin timing and not pin-to-clock. Any variation of one pin may greatly affect its timing in relation to another pin. Figure 2 illustrates a typical requirement among RAS, DATA, and MA.

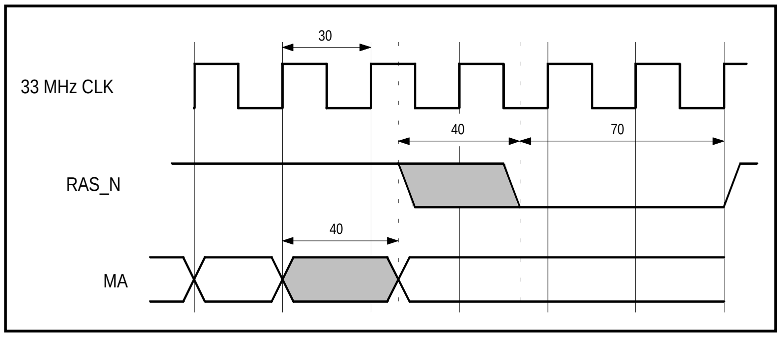


**FIGURE 2
RAS/DATA/MA
(Version 1)**

As shown in Figure 2, RAS must be active 70 ns prior to the end of the DRAM cycle. If the variation in output timing is 40 ns, then the point for activating RAS would be designed to be 110 ns prior to the end of the cycle. This would ensure that the slowest manufactured part would still provide the needed 70 ns prior to the end of the cycle. If the variation of MA is also 40 ns (and this delay does not track with the RAS delay), then MA would need to be designed to be valid 150 ns before the end of the cycle. If the variation in delay for RAS and MA can be kept to 20 ns, then MA would need to be valid 110 ns prior to the end of the cycle (see Figure 3). This difference of 40 ns would mean the difference in one wait state to the processor at 33 MHz. The same scenario also applies to MA and CAS, WE, CAS, and other timings.

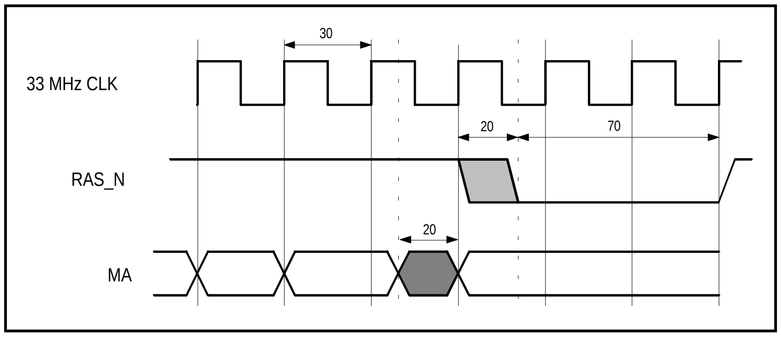


**FIGURE 3
RAS/DATA/MA
(Version 2)**

The key to keeping the output delays and variation in delay to a minimum is by using fast logic and placing the controlling signal as close to an output pin as possible (i.e., so there are a minimum number of gate delays from the clock to the output pad). The most common way of doing this is to place the output latch as the very last device in the path. This means a separate latch for each output signal. The abundance of flip-flops coupled with the extremely fast logic makes QuickLogic's 12x16-0 FPGA an ideal solution.

5.0 DESCRIPTION OF DESIGN

5.1 Overview

A Page-Mode DRAM Controller meeting the demands of the design problems outlined in the prior pages was implemented in an 84-pin version of the QuickLogic QL12x16 2000-usable gate FPGA. Complete design schematics for this controller are available from QuickLogic. The remainder of this application note describes the functions of each of the blocks in the design.

Figure 4 is a top view of the controller as created in the Data I/O ECS schematic capture tool. It consists of the Processor and DRAM Interface Signals and two blocks containing the input pads (INPADS) and the DRAM controller logic (DRAMCTRL). Tables 3 and 4 list the signal name and description of all the I/O pins.

Within the DRAMCTRL block of Figure 5, are six major logic blocks. The schematic hierarchy diagram of Figure 6 shows that six more blocks are embedded at lower levels. For example, the REFRESH block is contained inside the STATE block (see Figure 7).

FIGURE 4
DRAM Controller

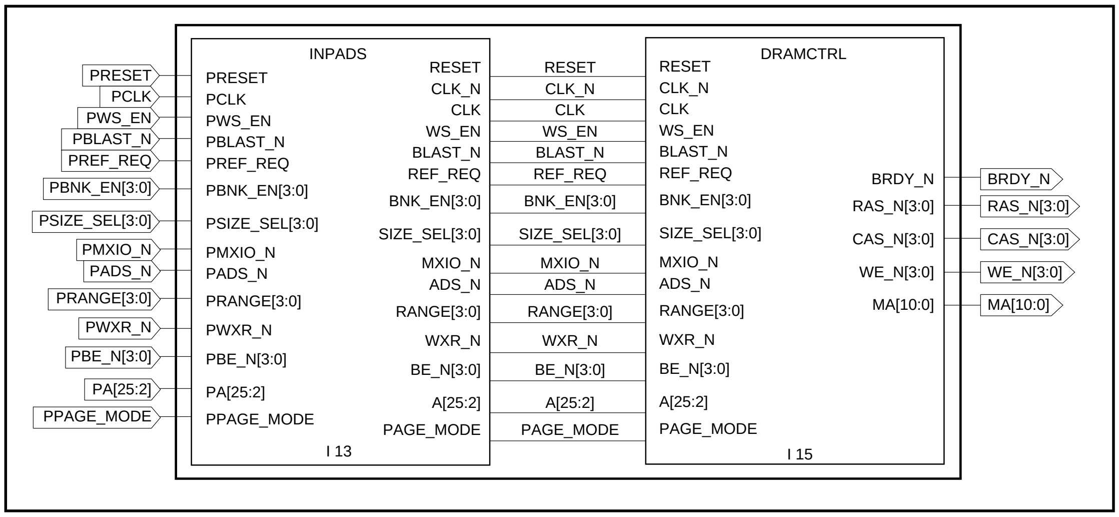


FIGURE 5 DRAMCTRL Major Function Blocks

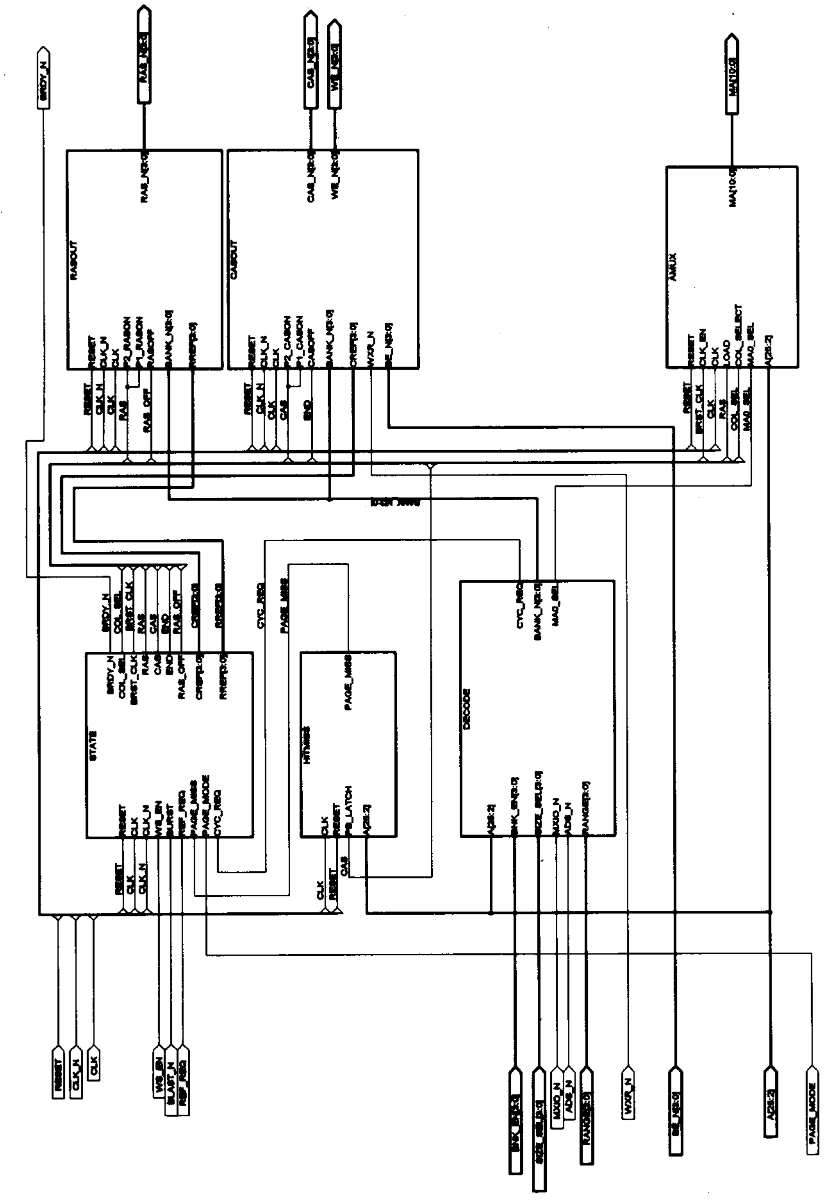
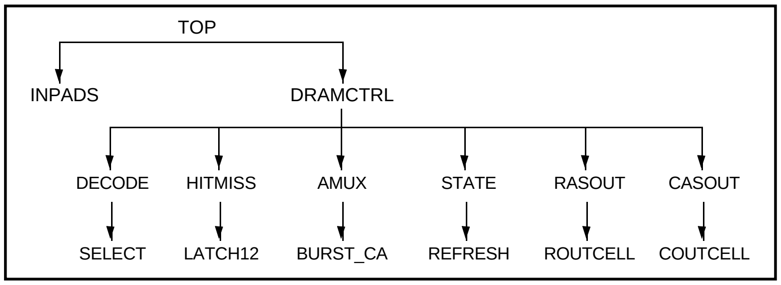


FIGURE 6
Schematic Hierarchy



A description of all twelve upper and lower level blocks follows:

DECODE – Address Decode: This block takes care of decoding the address range to determine if it is the DRAM range, and also bank selection.

SELECT – Cycle Select Logic: This block consists of a 4-bit range comparator and logic that takes in ADS_N (Address Strobe) and MXIO_N (Memory/Input/Output Status Signal) to determine the start of a cycle.

HITMISS – Page Hit/Miss Detection: This block detects whether the page address of the current cycle matches the page cycle of the last address. The output status is used as an input to the state machine.

LATCH12 – 12-Bit Register Latch: This block takes the 12 address lines that make up the page and latches them.

STATE – State Machine: Controls the entire DRAM controller. The state machine takes input of the form 'cycle request,' 'refresh request,' 'page miss,' and other configuration information to determine the cycle type and control the outputs accordingly.

REFRESH – This block is within the state machine block and is essentially the state machine for the refresh cycles.

AMUX – Address Multiplexer: This block takes care of driving the correct row and column address to the DRAM.

BURST_CA – Burst Column Address Generator: Within the AMUX block this block takes care of generating the correct CA[3:2] when burst cycles are performed.

INPADS – Input Buffers: This block contains all input pad declarations for the chip.

RASOUT – RAS Output Logic: This block clocks out the appropriate RAS_N signal at the correct time. It instantiates 4 ROUTCELL blocks to accomplish this.

ROUTCELL – RAS Out Cell: Takes the set and reset signals from the state machine and combines them with both CLK and CLK_N to clock out the RAS_N signals. Additionally, this block uses the bank select signals as enables.

CASOUT – CAS Output Logic: Similar to RAS Output Logic only used for the CAS lines. This block also contains logic to drive the WE_N[3:0] signals.

COUTCELL – CAS Out Cello: Almost identical to the ROUTCELL with a slight difference on the reset logic.

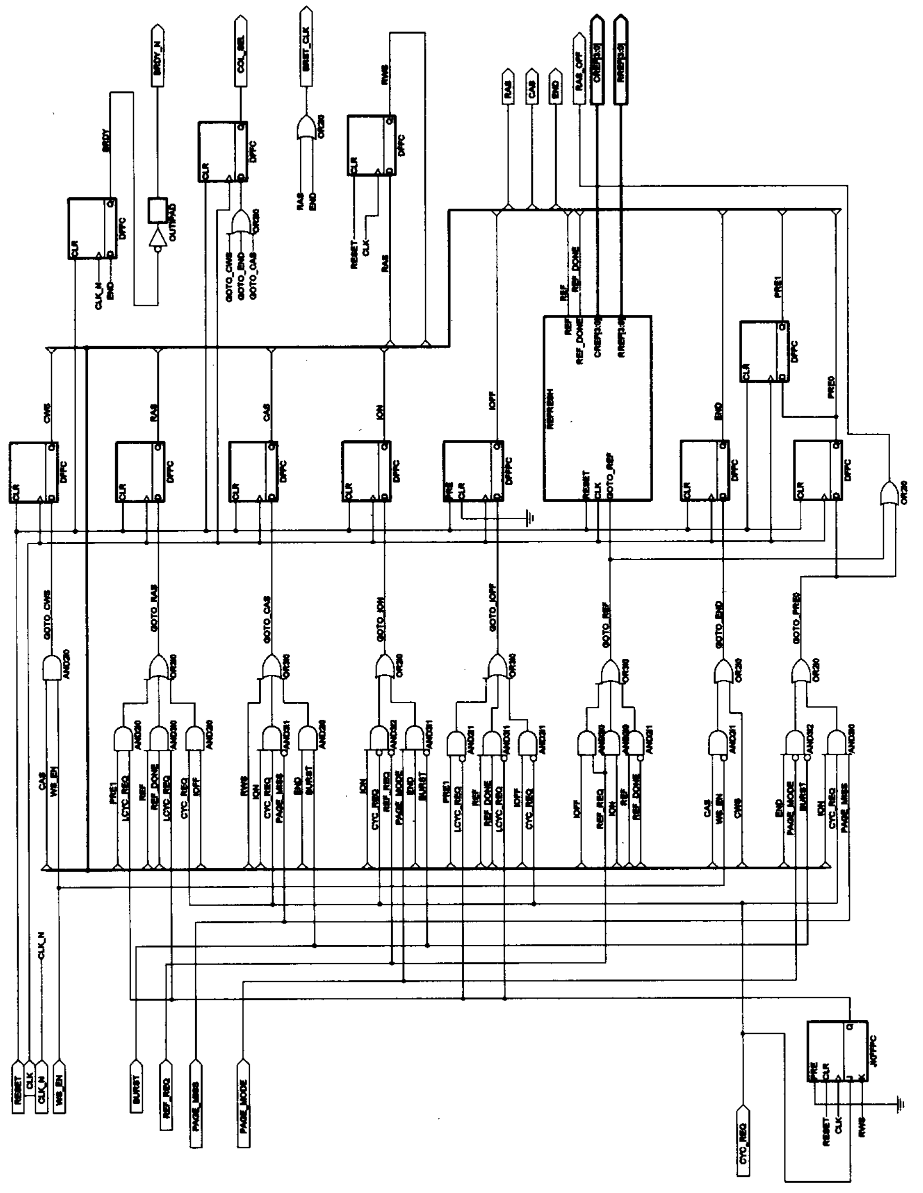


FIGURE 7 State Machine Logic Function

5.2 Interface Signals

TABLE 3
Processor Interface

SIGNAL	I/O	DESCRIPTION
A[25:2]	I	Processor Address lines: Address line inputs from the 486DX2. They are driven active on the rising edge of the clock and are guaranteed to be valid 14 ns later. The address is driven throughout the entire cycle.
ADS_N	I	Address Strobe: This signal is used to indicate the start of a cycle from the 486DX2. It has identical timing with the address. However, it will remain active for only one clock.
BE_N[3:0]	I	Byte Enables: Indicates which bytes of the current cycle are to be accessed. They have identical timing with the address.
BLAST_N	I	Last Burst Cycle: This signal indicates when the current cycle is the last of a burst cycle. Cycles that are not burst cycles will always have BLAST_N active.
CLK	I	33 MHz Clock: This is the clock input used to clock internal logic.
RESET	I	Reset Signal: Used to reset all logic internally.
BRDY_N	O	Burst Ready: This signal indicates the completion of the cycle. Since this device (DRAM Controller) supports burst cycles when requested, BRDY_N will always be used.
MxIO_N	I	Memory/Input/Output Status Signal: This signal indicates whether the current cycle is a memory cycle or an I/O cycle. It has identical timing to the address.
WxR_N	I	Write/Read Status Signal: This signal indicates whether the current cycle is a write cycle or a read cycle. It has identical timing to the address.
REFREQ	I	Refresh Request: This signal indicates a refresh cycle is desired by the external system. The assumptions here are that arbitration for the bus has already occurred outside this controller.
WS_EN	I	Wait State Enable: This signal is a programming option to configure the controller with a wait state during CAS active. When active, CAS will have approximately two clocks of access time. When inactive CAS will have approximately one clock of access time.
RANGE[3:0]	I	DRAM Range: These inputs indicate the overall size of the DRAM memory range. These four bits represent the number of 4 Meg blocks of memory.
BNK_EN[3:0]	I	Bank Enables: These four signals represent which banks are enabled.
SIZE_SEL[3:0]	I	Bank Size Selects: These four signals represent the size of each bank of memory. When inactive, the size of the bank is 4 megabytes. When active, the size is 16 megabytes.
PAGE_MODE	I	Enable Page Mode: This programming option is used to enable page mode. Because of the timing restrictions, worst case design indicates a critical path in the page mode detection logic. However, at lower frequencies or with fast parts, page mode may still be used. When inactive, the critical path is removed from the logic.

TABLE 4 DRAM Interface

SIGNAL	I/O	DESCRIPTION
RAS_N[3:0]	O	Row Address Strobes: These signals are used to strobe in the row address for each of the banks.
CAS_N[3:0]	O	Column Address Strobes: These signals are used to strobe in the column address for each of the banks.
WE_N[3:0]	O	Byte Write Enables: These four signals represent write enables for each of the bytes in a DWORD.
MA[10:0]	O	DRAM Row/Col Memory Address: These 11 outputs are used to present both the row and column to the DRAM.

5.3 Cycle Detection Logic

Cycles in the design are started by the address being within the range programmed on the RANGE[3:0] bits, ADS_N going active, and MXIO_N indicating memory. When these three events occur the signal CYC_REQ goes active indicating to the state machine to start a cycle.

The first attempt at designing this used a macro for the TTL range comparator 74684. Since this was an 8-bit comparator, four of the comparisons were redundant and were tied inactive. The output of the comparator was then combinatorially ANDed with ADS_N and MXIO_N. The use of the extra logic needed for 8-bit comparison and the following level of decode made this path painfully slow. It was not possible to detect the cycle by the next clock edge after ADS_N had gone active.

By reducing the comparator to a 4-bit range comparator — and combining the ADS_N and MXIO_N logic into the comparator, it was possible to speed this path up from 18 ns to 8 ns, thus making it functional. Paying close attention to which gates fit directly in one level of a logic cell also helped. An inverter was needed in the ADS_N path in order to combine it with the comparator logic. However, since ADS_N will be active well before the range comparison is complete, it is not in the critical path.

5.4 Page Compare Logic

This page hit/miss logic is similar to the range compare, except that it must be performed over 12 address lines and it only needs to match the value. However, because of the need to compare 12 lines down to one output, this path is very slow. Under worst case conditions for the QL12x16-0 this path is simulated to 27.6 ns.

The delay of 27.6 ns was well above the allotment of 15 ns for page hit/miss logic. However, under best case conditions this path is within margin. For this reason, an option was added to keep the page mode access. This option is controlled by enabling the input PAGE_MODE. When active, the state machine will assume the page compare logic is fast enough and will use the output to determine state transitions. If PAGE_MODE is inactive, then the PAGE_MISS signal will be masked off in the state machine.

Design Tip

If page mode is strongly desired under all conditions at 33 MHz, then the state machine could be modified to support this. By designing in a one clock delay at the start of the cycle in which no activity is performed, the page hit/miss logic has an additional 30 ns to complete. This of course, increases all cycles by one clock, but if the cycle is a page hit, the RAS and RWS state are bypassed — saving two clocks. There is still a net gain of one clock.

5.5 Bank Decode Logic

The bank decode logic consists of a static section and a dynamic section. The static section decodes the choices programmed on the SIZE_SEL[3:0] pins and the BNK_EN[3:0] pins. The programming on these pins comes down to five possible cases. The delay on this logic is irrelevant since it is static.

The dynamic section takes the five case signals and combines them with the address to determine the correct bank. The delay through this logic must be less than 30 ns in order to be valid when RAN_N is activated. The worst case simulation came to 15 ns.

Design Tip

If more pins are needed for a different implementation, the size and enable pins could be replaced with the case signals directly, which would reduce eight pins to five pins.

5.6 Row/Column Address Mux

The row/column address mux consists of several of 2-to-1 multiplexers that select the appropriate address bits to drive for the row and column. The selection between row and column is made using the COL_SEL signal that comes from the state machine.

Tables 5 and 6 show the various DRAM configurations that the multiplexer is capable of supporting. However, only two of the address mux modes are supported directly in the bank decoder.

The 1M depth and the 4M depth give bank sizes of 4M and 16M respectively, which are supported in the bank decoder. The other address mux modes could be used provided only one bank was populated. The bank decoder could also be redesigned to support different decoding combinations. Both MA0 and MA1 require additional levels of muxing for row address bits. MA0 must support A11 and A21, while MA1 must support A12 and A23.

TABLE 5 Row Address Map

DRAM DEPTH	ROW/ COLUMN	10	9	8	7	6	5	4	3	2	1	0
256K	9x9	22	20	19	18	17	16	15	14	13	12	11
512K	10x9	22	20	19	18	17	16	15	14	13	12	11
1M	10x10	22	20	19	18	17	16	15	14	13	12	21
2M	11x10	22	20	19	18	17	16	15	14	13	12	21
4M	11x11	22	20	19	18	17	16	15	14	13	23	21

TABLE 6 Column Address Map

DRAM DEPTH	ROW/ COLUMN	10	9	8	7	6	5	4	3	2	1	0
256K	9x9	12	11	10	9	8	7	6	5	4	3	2
512K	10x9	12	11	10	9	8	7	6	5	4	3	2
1M	10x10	12	11	10	9	8	7	6	5	4	3	2
2M	11x10	12	11	10	9	8	7	6	5	4	3	2
4M	11x11	12	11	10	9	8	7	6	5	4	3	2

Burst cycles on the 486 require external logic to generate A[3:2]. Depending on the first address, the subsequent values may either be an incremented address or decremented address. The BURST_CA block takes care of detecting the first address, latching the address, and generating subsequent CA[3:2] bits of the burst cycle. Table 7 shows the burst address sequence.

TABLE 7 486DX2 Burst Address Sequence

BURST ADDRESS SEQUENCE A[3:2]				
FIRST	SECOND	THIRD	FOURTH	COMMENTS
00	01	10	11	Increment
01	00	11	10	Decrement
10	11	00	01	Increment
11	10	01	00	Decrement

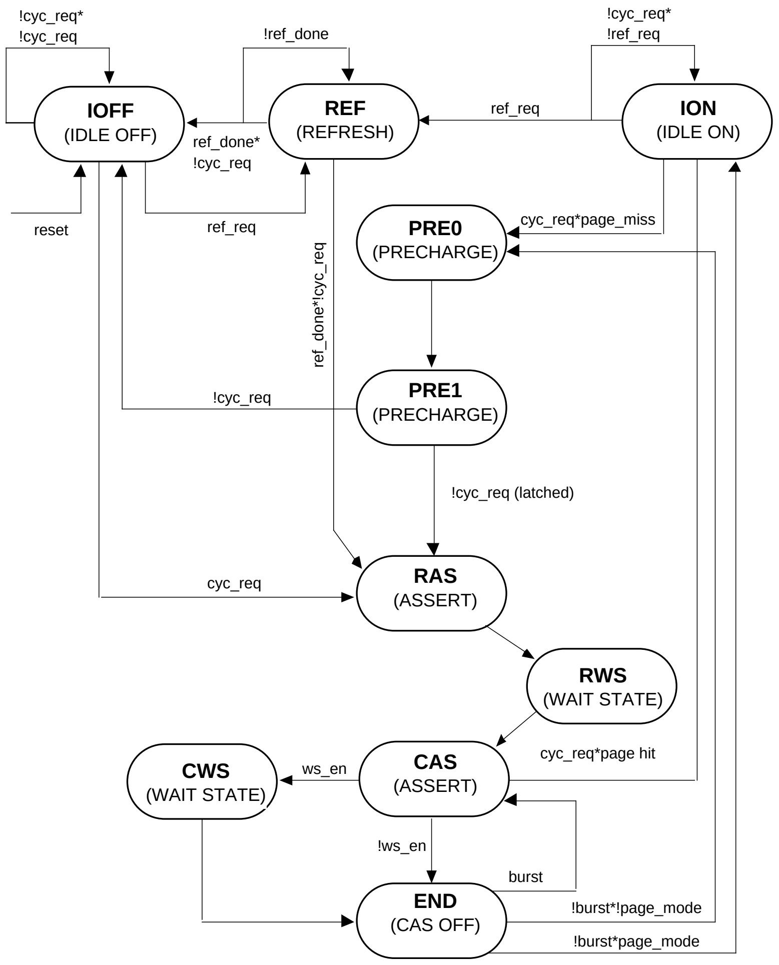
5.7 State Machine

The state machine contains all the control logic for the DRAM controller. In most cases a state output may be used directly for the desired control function. However, some control functions require additional logic to combine several states to create a control signal.

The state machine resets to the IOFF state. In this state RAS is inactive. Requests to leave this state may consist of either a refresh request or a processor cycle request. When a processor cycle request is made, a transition to the RAS state will occur where RAS is activated. The cycle will then proceed through RWS, CAS (where it asserts CAS) and END. The end state is the last state of a single data transaction and is used to deactivate CAS. If burst mode is indicated by the processor, then following cycles may transition directly back to the CAS state to reactivate CAS.

Refresh cycles occur in the REF state. A separate state machine controls the actual refresh cycles. The main state machine will remain in the REF state until a REF_DONE sinal is received from the refresh state machine. At this point all the RAS_N outputs are precharged, thus the machine waits for the next cycle request in the IOFF state.

If page mode is not enabled (and not bursting), transitions from the END state will reset the RAS_N output. If page mode is enabled, then RAS will not be reset and a transition to the idle on state will be made (ION). Departure from ION will occur for refresh request and processor cycle request. For the latter, a transition to either PRE0 or CAS will be made depending upon the page miss status. Page miss cycles from the ION state will require precharging in PRE0 and PRE1 states. Figure 8 shows the state transition table.



Normal Cycle
(Including Burst)

FIGURE 8
DRAM State Machine

The states and Cycle State Machine Equations are defined as follows:

IOFF – Idle RAS Off State: This state is an idle state with RAS inactive. This state is entered at reset and from the refresh state when no cycles are pending. This state may also be entered from the PRE1 state if page_mode is not enabled and no cycle is pending.

```
IOFF = IOFF•/CYC_REQ
+ REF•REF_DONE•/CYC_REQ
```

ION – Idle RAS Active State: This state is identical to the IOFF state except that one of the RAS_N outputs is active. This state is only used if page_mode is enabled. If a refresh request occurs while in this state, then a transition to the REF state is made. The REF state makes sure the lines get precharged before starting the refresh cycle. When a regular cycle request is made, a transition to either the precharge states or the CAS state is made depending upon whether the page address matches the previous cycle page address.

```
ION = ION•/CYC_REQ•/REF_REQ
+ END•/BURST•PAGE_MODE
```

PRE0 – RAS Precharge State 0: This state is the first of two when precharging the RAS_N outputs.

```
PRE0 = ION •CYC_REQ•PAGE_MISS
+ END•/BURST•/PAGE_MODE
```

PRE1 – RAS Precharge State 1: This state is the second of two when precharging the RAS_N outputs. Transitions out of this state will always be to the RAS state when page_mode is enabled. However, if page_mode is not enabled, then the precharged states may not have been entered as a result of a cycle request (as in the case of page_mode), and therefore, the state of cyc_req must be checked to determine between IOFF and RAS.

```
PRE1 = PRE0
```

RAS – Assert RAS Active State: This state is used to activate the correct RAS_N output. It can be entered from three different states. RAS may be entered from IOFF directly whenever a cycle request is made. In this scenario the RAS_N outputs are already precharged. Likewise, if a request comes near the completion of the refresh cycle, then a direct transition from REF to RAS is possible. The REF state assures that all RAS_N outputs are precharged when leaving. The third entry point to RAS occurs from the PRE1 state when a request is pending.

```
RAS = PRE1•LCYC_REQ
+ REF•REF_DONE•LCYC_REQ
+ IOFF•CYC_REQ
```

RWS – RAS Wait State: This state is needed to meet RAS active time specs. It will always be entered directly from the RAS state and transition to the CAS state.

```
RWS = RAS
```

CAS – Assert CAS Active State: This state is entered from one of three different states. It may be entered directly from RWS for a page miss cycle, or from ION for a page hit cycle. Additionally, the CAS state may be entered from the END state when the 486 is bursting bus cycles. This situation is similar to a page hit cycle with no idle clocks and no need to detect the hit/miss. Although this cycle is essentially a page hit cycle, it will be operational in both page mode and non page mode since it does not require detecting the page hit/miss status.

```
CAS = RWS
+ ION•CYC_REQ•PAGE_HIT
+ END•BURST
```

CWS – CAS Active Wait State (also called Wait State 0): This state is similar to the RWS state in that it is used to extend the CAS access time. This state is entered from the CAS state whenever the wait state enable pin (WS_EN) is active. When in use, the CAS active time will be on the order of two clocks, and burst and page hit cycles will be approximately three clocks.

```
CWS = CAS•WS_EN
```

REF – Refresh Active State: This state is entered when a refresh cycle is needed. It can be entered from the ION state or the IOFF state. It is assumed in the design that arbitration for the bus takes place outside the chip, and that the ref_req signal is mutually exclusive with the bus cycle ADS_N going active. Once in the REF state, the machine will remain until refresh is complete. Once the refresh cycle is complete, all RAS_N outputs are precharged and ready to access a new bank. The machine can then directly move to the RAS state if a cycle has begun. If no cycle is pending, then a transition to the IOFF state is made so that the RAS_N outputs will remain precharged.

```
REF = REF•/REF_DONE
+ ION•REF_REQ (MUTUALLY EXCLUSIVE WITH CYC_REQ)
+ IOFF•REF_REQ (MUTUALLY EXCLUSIVE WITH CYC_REQ)
```

END – Last CAS Active State: This state is the last state of the bus cycle. It is primarily used to reset the CAS_N outputs, and sometimes the RAS_N outputs if needed. It is used to activate the correct CAS_N output.

```
END = CAS•/WS_EN
+ CWS
```

Additionally, there are several output signals based on different transitions that are defined as follows:

BRDY_N – Burst Ready Signal: This signal is simply a delayed version of the END state. Since the ready signal to the 486DX2 must straddle the rising edge of the clock, it is best to clock the ready with the falling edge of the clock.

COL_SEL – Column Select Signal: This signal is used in the address multiplexer to select between row and column. When active, it selects the column as the name implies. The default for this signal is inactive. It is driven active during the CAS, CWS, and END states.

BURST_CLK – Burst Clock Signal: This signal is used to change the column address bits used for successive cycles in a burst transaction. This signal will go active during the RAS state to latch in the first value. It will then go active on each END state to clock the next value.

RAS_OFF – RAS Off Signal: This signal is used to reset the RAS outputs at the end of a cycle when not in page mode, and also during idle when in page mode. It goes active when the next state of the state machine will be PRE0 or REF.

CAS-Before-RAS Refresh

The refresh cycles are controlled by the refresh state machine contained in the REFRESH block. This block consists primarily of a shift register triggered by entering the REF state of the cycle state machine.

The type of refresh cycle employed by the controller is known as CAS-before-RAS. This type of refresh cycle is lowest in power and requires the least amount of external logic. By activating CAS first and then RAS, the DRAM detects that a refresh cycle is intended and therefore uses an internal refresh row to address the row. Additionally, the DRAM will know to not drive the output buffers of the DRAM.

Activating refresh to all banks simultaneously can cause huge power surges and results in a difficult system design. The controller presented here staggers the accesses to each bank by one clock — thus keeping power surges to a minimum.

The control signals to enable refresh for each RAS and CAS of each bank are created by detecting different points in the shift register. For example, the bank 0 CAS will go active when the second latch in the shift register goes active, and will go inactive when the fifth latch goes active. The logic is then shifted for each of the following banks. At the end of the shift register a signal goes active for one clock to indicate the completion of the refresh cycle. At this point the last bank has been precharged for one clock and will have been precharged for two clocks by the time the cycle state machine begins a cycle.

5.8 Output Logic

The output logic consists of logic required to clock RAS, CAS, and WE. The implementation used here uses a generous amount of latches (which are plentiful in the QL 12x16) to clock out RAS and CAS on phase 2 clock edges while deactivating on phase 1. Two different cells, COUTCELL and ROUTCELL, are repeated multiple times and wired to the appropriate bank enables to achieve this.

The basic outcell (ROUTCELL and COUTCELL) uses a simple D flip-flop to start activation on phase 2 (by clocking with CLK_N). A second JK flip-flop is used to continue driving the output for the rest of the cycle. While the D FF remains active for only one clock, the JK FF will remain active until the reset signal comes along. The outputs of these two latches and the refresh signal are ORed together to create the final output signal.

Design Tip

Potential areas for improvement would be reconfiguring the logic such that the NOR gate could be removed. This could be accomplished by using one JK FF clocked on phase 2 which would have separate set and reset signals for refresh cycles combined with the set and reset of processor cycles. The only downside would be the deactivation of the signal in phase 2 instead of phase 1. While this would be fine for RAS, CAS would have a problem on page mode and burst cycles meeting.

6.0 PHYSICAL VIEW OF COMPLETE DESIGN

Figure 9 shows a typical view of how the controller design is implemented in the QL12x16 FPGA logic cell architecture. It occupies 122 out of the 192 available logic cells and 73 of the 76 I/O pins.

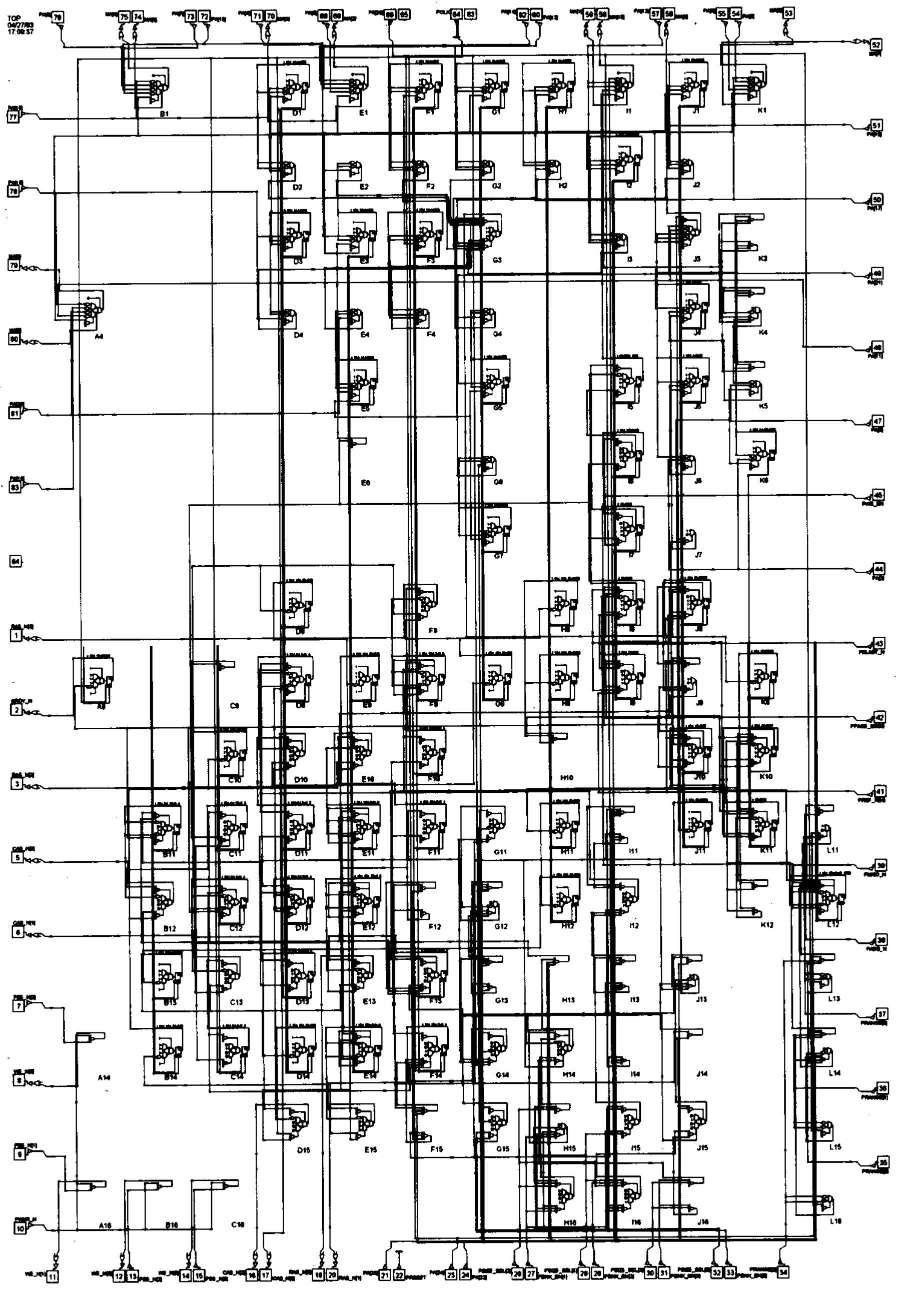


FIGURE 9 Pinout and Cell Interconnect

7.0 QUICKLOGIC TOOLS AND FPGA ARCHITECTURE

The QuickLogic FPGA development tools are completely integrated under Microsoft Windows making use of hierarchical links between all aspects of the design with a familiar user interface.

The QuickLogic development tools were extremely advantageous in the development of this application note. The schedule available for completing this design was extremely tight. The tight schedule, coupled with the fact of not ever having used the tools initially created a feeling of extreme apprehension. However, the intuitive nature and ease of use of the interface made it possible to be up to speed within hours.

Design Entry

The design entry phase consisted of using the Engineering Capture System. This schematic entry package was well suited for hierarchical design, making it possible to easily design generic cells which could be repeated multiple times at a higher level. The hierarchical nature of this tool also made it very easy to observe all connections within the design.

The built-in symbol generator and error checker made creation of symbols very straightforward and painless. At any time when the design of a cell changed, which happened continuously, all that was needed was a quick click on the CREATE-SYMBOL option to replace the old symbol. If at any time there were errors between the symbol and the schematic, the error checker would point them out.

Design Verification

I generally used the design simulator after placing and routing to perform both logic verification and timing verification simultaneously. The simple process of back annotation made it easy to verify the design with post layout delay information. The waveform tool made it easy to enter stimulus to the design, and observe the output during simulation.

FPGA Architecture

The FPGA architecture made it possible to achieve a fast decode circuit with minimal clock to output skews — both of which were extremely important in this design. The direct outputs from each of the first-level gates allowed for minimal propagation delay in sections of logic that did not require the entire logic block. Additionally, input selection into the last mux of the logic block made it possible to implement the address mux much faster than standard AND-OR architecture. The end result was a complete high-speed DRAM controller design that was accomplished in several weeks — versus several months with custom designs.