Application Note

Interfacing SDRAM to the MC68360 QUICC Device

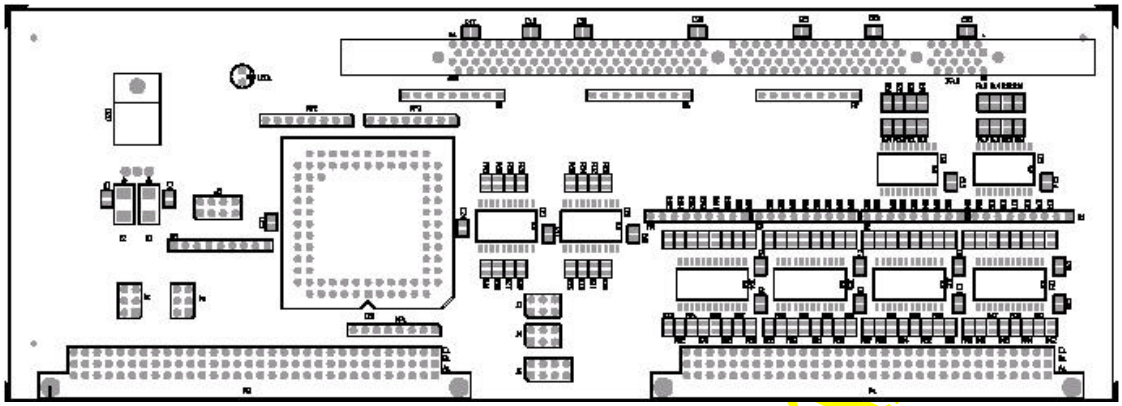
Michael Drozd
Freescale GmbH, Munich

1. Introduction

In the long term, Synchronous DRAMs (SDRAM) offer system designers at least two advantages over conventional Fast Page Mode or EDO DRAMs: a speed-upgrade and density road map to meet performance requirements of future processors, and a smooth transition to higher bus speeds without the need to implement major system re-designs. As SDRAMs reach price parity with conventional memories, SDRAM is making inroads into the PC main memory, space previously held by EDO/BEDO. This has implications for the embedded market place. Not only the performance, but also the price reductions will persuade designers to use SDRAM in preference to the previous architecture.

The intention of this document is to demonstrate a way to use SDRAM DIMM or stand-alone SDRAM devices in systems which originally do not provide SDRAM control signals. RAS-CAS signals generated by the memory controller for asynchronous RAM will be converted into RAS-CAS-WE-A10 signals basing on the JEDEC spec. Furthermore, the document describes a recommended setup for the memory controller and a setup procedure to provide the correct initialization of the SDRAM devices.

2. Adapter Board Layout



JUMPER:
J1: Clock selection 25/50MHz and selection of active clock edge
J2: Reset selection and MODE selection
J3: BA_IN1 address selection
J4: BA_IN0 address selection
J5: BA_IN2 selection or RAS_IN2 selection
J6: JTAG programming port
MP1: Measurement points for input signals
MP2,3: Measurement points for SDRAM control signals

3. SDRAM Adapter Board

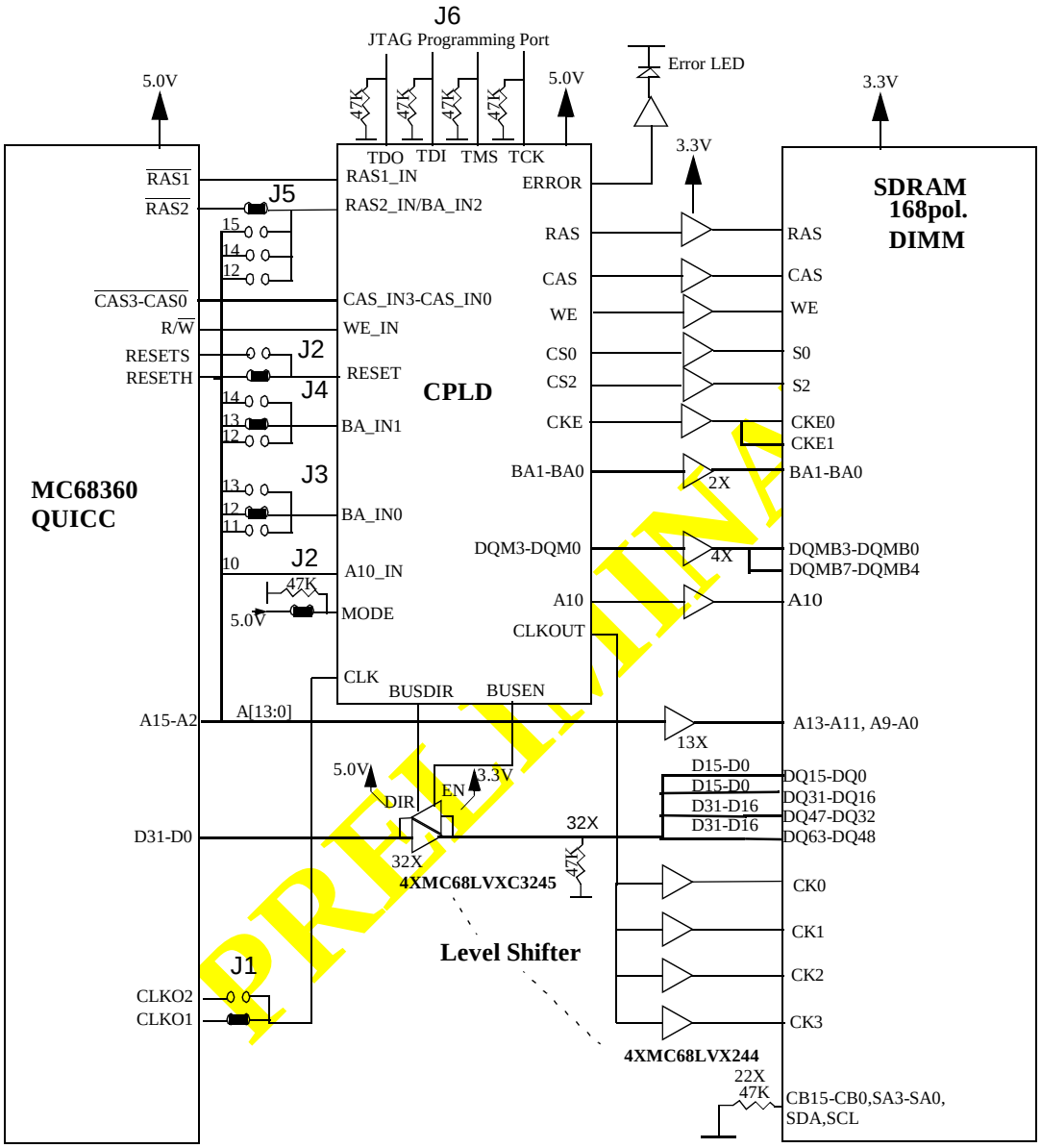


Figure 1. SDRAM Interface

The adapter board supports 2 different modes:

1) Mode = 0; No Jumper J2; Jumper J5 must select RAS2 for RAS2_IN. Two separate banks on RAS1 and RAS2.
2) Mode = 1;Jumper J2 in place; J5 selects A12, A14 or A15. One contiguous bank on RAS1.

4. SDRAM Features

4.1 Important SDRAM Timings

While SDRAM command information and data are always clocked in with the SDRAM CLK line, there are certain important timings that must be adhered to for SDRAM to function correctly. Those of special interest in this document are listed below. The example values are taken from the data sheet of a Micron MT48LC8M8A2-8E 100 Mhz device. Similar values can be found for other SDRAM devices. A CAS latency of two clocks is used in all diagrams.

Timing	Micron SDRAM Value	Description
t_{RCD}	30 ns	ACTIVE to READ or WRITE delay
t_{AC}	6 ns	Read Access Time (CAS Latency = 2 or 3)
--	2-3 clocks	READ to data valid (CAS latency)
--	0 clocks	READ/WRITE to READ/WRITE
--	0 clocks	READ to PRECHARGE
t_{RP}	20 ns	PRECHARGE to ACTIVATE
t_{REF}	64 ms	Refresh Period for 4096 ROWs * 15.625us = 64ms
t_{SETUP}	2 ns	Common name for worst case signal setup times to rising edge of CLK

Table 1. SDRAM Timings

4.2 SDRAM Commands

Name	CS	RAS	CAS	WE	addr	data	Note
INHIBIT	H	X	X	X			
NOP	L	H	H	H			
ACTIVATE	L	L	H	H	ROW/BANK		
READ	L	H	L	H	COLUMN/BANK/A10		
WRITE	L	H	L	L	COLUMN/BANK/A10	VALID	
BURST TERMINATE	L	H	H	L			
PRECHARGE (PRCG)	L	L	H	L	A10		1
AUTO REFRESH/SELF REFRESH	L	L	L	H			2
LOAD MODE REGISTER	L	L	L	L	OP-CODE		3

Table 2. SDRAM Command Encoding

Note: 1) If a10 HIGH both banks will be precharged. If a10 is LOW the selected bank will be precharged.

Note: 2) If clock is enabled by CLKE equals HIGH, AUTO REFRESH is enabled. If CLKE is LOW SELF REFRESH is enabled.

Note: 3) Address provides data for mode register.

5. Generation of Command Sequence from RAS-CAS Wave Forms

The Interface analyzes RAS-CAS wave forms generated by the 360er DRAM Controller for asynchronous DRAMs in order to generate SDRAM command sequences. The most critical timing is the read access. The 68360 DRAM Controller has to wait until the SDRAM can provide output data. Assuming that the 68360 runs >25MHz the timing requirements of an asynchronous DRAM results in a 4-2-2-2 wave form or 5-3-3-3 for slower implementation. The resulting SDRAM command sequence is generated requiring a 2 cycle CAS latency.:

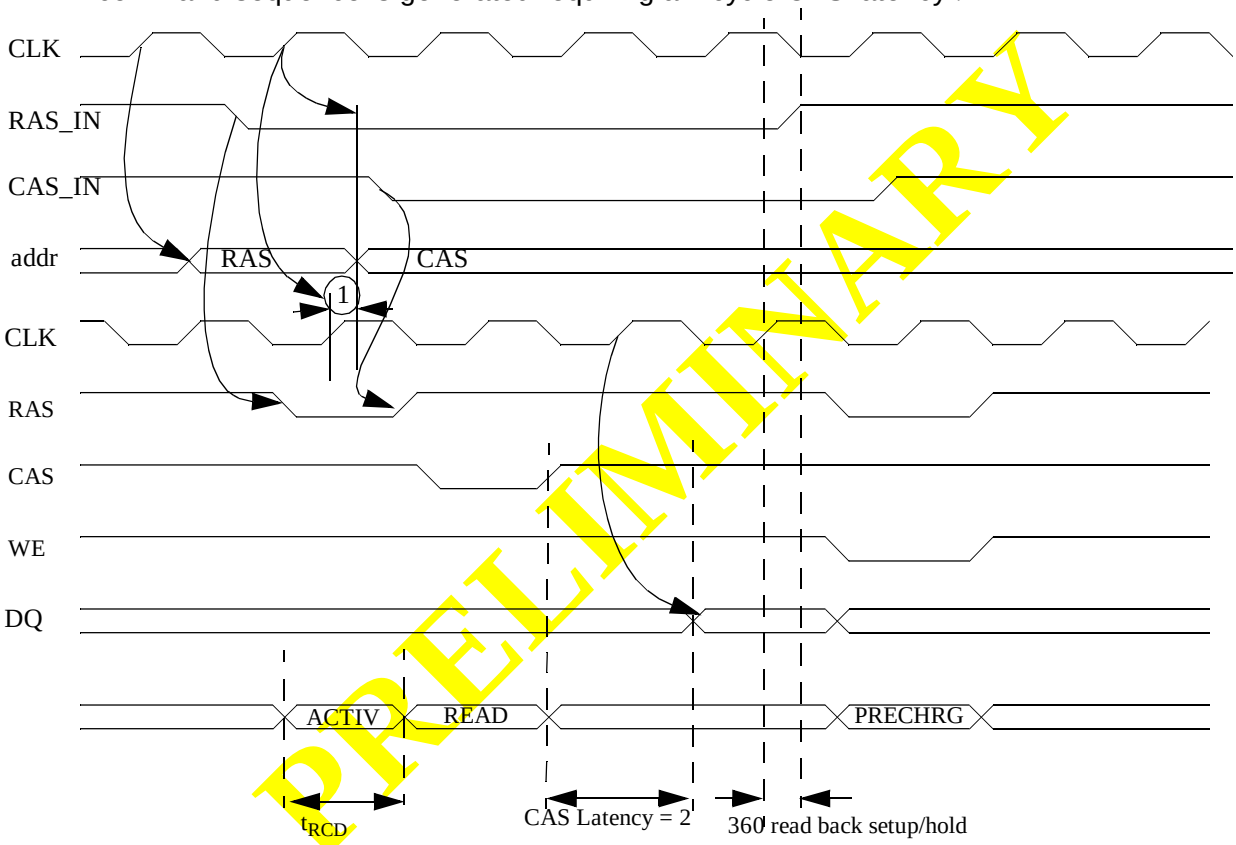


Figure 2. Single Read Access

During ACTIVATE the SDRAM requires the ROW address and the bank selection. The bank selection is the most significant row address. During READ or WRITE the column address must be valid on the address lines. The 360 DRAMC provides the addresses to the right time. No address latch is necessary and no additional address output bus is required for SDRAM.

For write access the SDRAM requires the input data during WRITE command. The DRAMC provides the data during CAS is low. No data latching is required. No additional data bus multiplexing is necessary

!

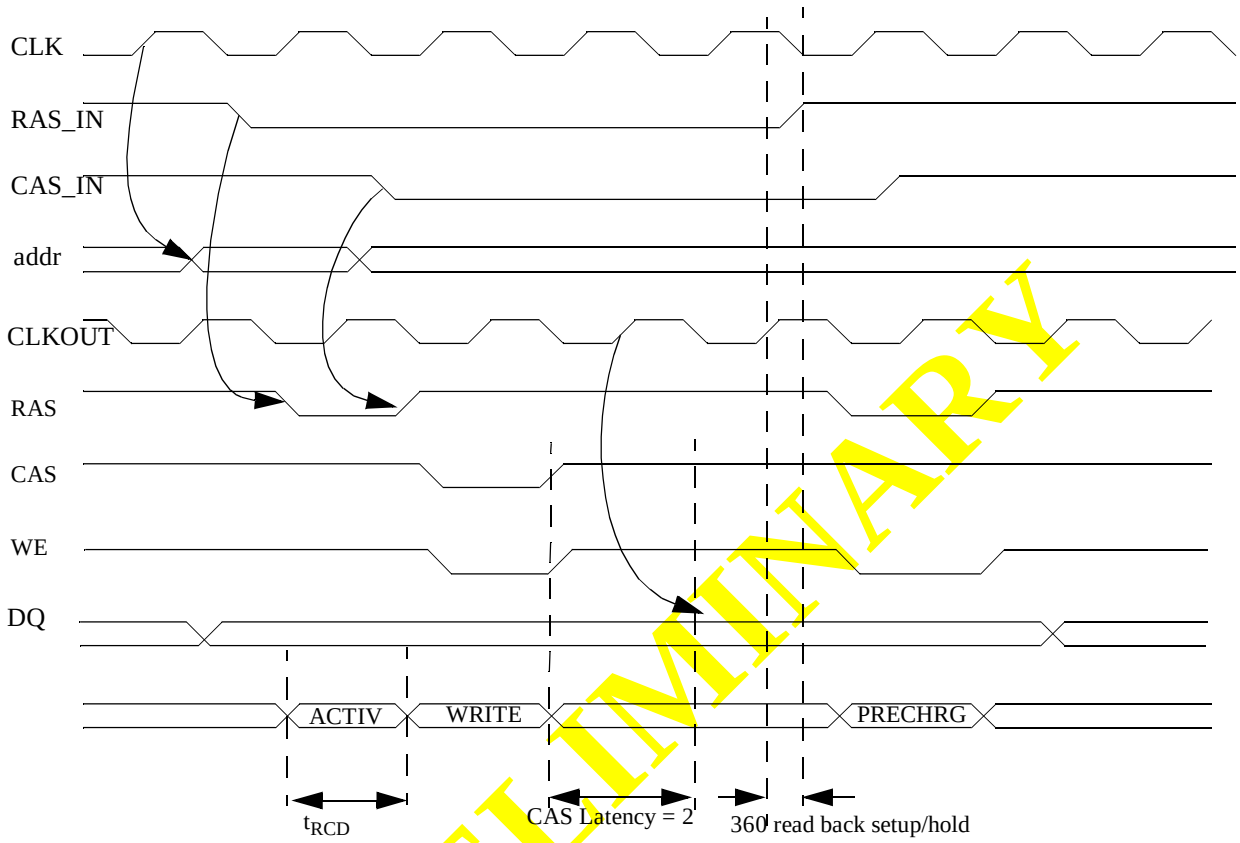


Figure 3. Single Write Access

The row is activated until the page is given up by the DRAMC by deasserting RAS. This enables the adapter to support Page Mode wave forms.

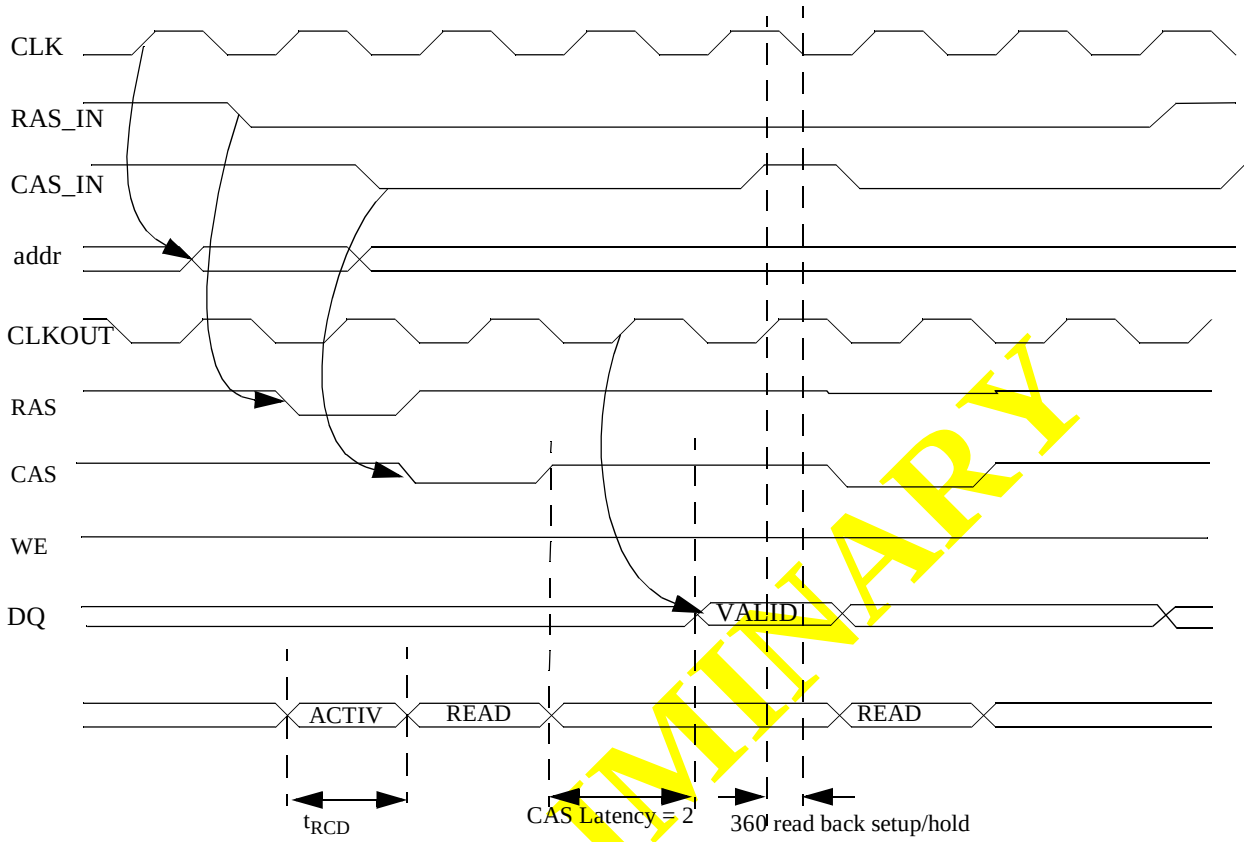


Figure 4. Page Mode Read Access

Please note that the burst mechanism of the SDRAM is disabled. Each read or write access requires a new CAS address.

6. Bank and A10 Multiplexing

Because of the special function of the A10 signal the adapter provides a special input. Also, the bank address must be stable during RAS (ACTIVATE) and during CAS address time (READ/ WRITE command). The bank address are connected to the most significant addresses. This means these address pins must be latched during RAS time in order to provide the signals also during CAS time. The dedicated bank address inputs of the adapter will be connected to address lines according to the size and configuration of the SDRAM devices. The address assignment can be done statically by board wiring or can be programmable by the adapter internally. The current version supports hard-wired solution. The programming of the A10 and bank address multiplexing can be done using the data bytes of the first 10 dummy writes for initialization (see next chapter).

7. Initialization of SDRAM

The first task with SDRAM is to program the power-up sequence. The JEDEC standard for power-up is the following:

1) maintain NOP inputs for 200 µs - the may be maintained after power-up by driving an inactive state.
2) Issue Precharge command to both banks (PRCG), and wait t_{RP}, the precharge time. For t_{RP}=24 ns, one wait clock (NOP) is required for 50 Mhz bus speed.
3) Issue at least 8 auto refresh commands (REF). Between refreshes, the refresh command period t_{RC} must be met. Based on a t_{RC} value of 80 ns, this requires two wait clocks for 33 Mhz, three wait clocks for 40 MHz, and four wait clocks for 50 Mhz.
4) Issue a mode register set (MRS) to initialize the mode register. This programs the burst length, CAS latency and write mode. After time t_{RSC} the first access can occur. For t_{RSC} = 2 clocks, one trailing wait clock is required.

The initialization is software transparent. After reset the adapter ignores all SDRAM access until the first byte write access will be performed into the DRAM area. The first write access generates the precharge command independent from address and data. The write access itself is ignored. The 2nd byte write access generates the first autorefresh commend. The 3rd byte write generates the 2nd autorefresh and so on. The 10th write byte access generates the load mode register command. This write access must have a CAS address of the value required as data byte for the SDRAM mode register. The physical CAS address can be programmed as offset address. The physical address depends on the actual data width of the connected SDRAM.

Table 3. Mode Register Setting

	required mode register data	address offset for 8bit data bus	address offset for 16bit data bus	address offset for 32bit data bus
CAS Latency = 2	$0220	$0220	$0440	$0880
CAS Latency = 3	$0230	$0230	$0460	$08C0

The first 4 byte write accesses are used to program the bank select mux and A10 select mux.(refer to previous chapter). In fast page mode the activated page of the dummy write accesses must be given up before the SDRAM can start working. This requires a 11th dummy write access to the SDRAM into a different page after the first four write accesses. This is the first write access which has an effect. The previous 10 accesses have been suppressed by the SDRAM adapter.

An example initialization sequence is given in the following assembler source:

```
LEA    DC0_B_Addr+$880,A0      ; load mode config 32bit data bus, CAS latency = 2
MOVE.B      #$03,D0            ; could be A10 input multiplexing byte
MOVE.B      D0,(A0)            ; write byte 1, PRECHARGE BOTH BANKS
MOVE.B       #$03,D0           ; could be BA0 input multiplexing byte
MOVE.B      D0,(A0)            ; write byte 2, AUTO REFRESH
MOVE.B      #$03,D0            ; could be BA1 input multiplexing byte
MOVE.B      D0,(A0)            ; write byte 3, AUTOREFRESH
MOVE.B      #$03,D0            ; ignored
MOVE.B      D0,(A0)            ; write byte 4, AUTOREFRESH
MOVE.B      #$03,D0            ; ignored
MOVE.B      D0,(A0)            ; write byte 5, AUTOREFRESH
MOVE.B      #$03,D0            ; ignored
MOVE.B      D0,(A0)            ; write byte 6, AUTOREFRESH
MOVE.B      #$03,D0            ; ignored
MOVE.B      D0,(A0)            ; write byte 7, AUTOREFRESH
MOVE.B      #$03,D0            ; ignored
MOVE.B      D0,(A0)            ; write byte 8, AUTOREFRESH
MOVE.B      #$03,D0            ; ignored
MOVE.B       D0,(A0)           ; write byte 9, AUTOREFRESH
MOVE.B      #$03,D0            ; ignored
MOVE.B      D0,(A0)            ; write byte 4, LOAD MODE REGISTER

LEA         DC0_B_Addr+$400,A0 ; load different page address
MOVE.B      D0,(A0)            ; give up page, no dummy write!
```

8. Refresh

SDRAM distinguish between two possible modes of refreshing rows, AUTOREFRESH and SELFREFRESH. The row address is generated internally in both modes. If the SDSRAM is disabled by CKE=0 and the SDRAM is in power down mode, the rows will be refreshed by SELFREFRESH mechanism. During normal mode the rows will be refreshed whenever an AUTOREFRESH command is issued. This avoids conflicts with other RAM accesses.

The Adapter does not support the power down mode. The AUTOREFRESH command is generated after a CAS-RAS refresh wave form. The row address is ignored.

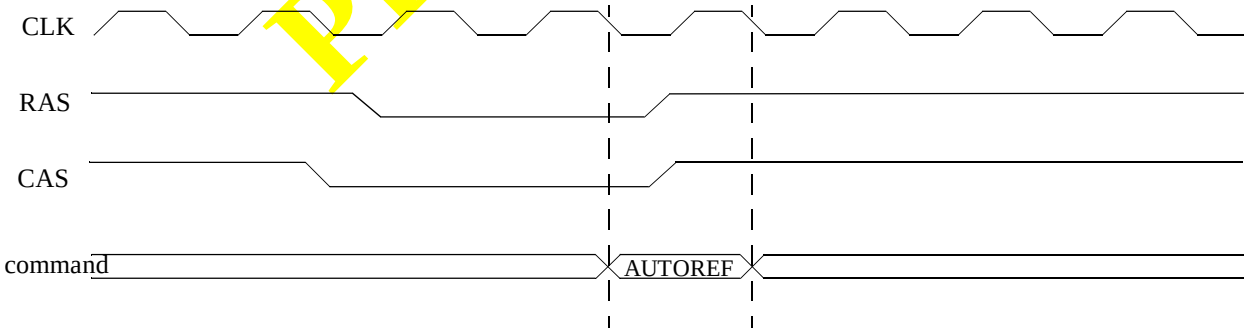


Figure 5. Refresh

9. Recommended 68360 DRAM Controller Register Setup

The GMR register defines the refresh period some timing parameter and address multiplexing. For a 16M SDRAM banks using 4 devices with 8bit bus width the following setting is recommended:

GMR

RCNT7	RCNT6	RCNT5	RCNT4	RCNT3	RCNT2	RCNT1	RCNT0	RFEN	RCYC1	RCYC0	PGS2	PGS1	PGS0	DPS1	DPS0
0	0	0	1	0	1	1	1	1	0	0	0	0	1	0	0
WBT40	WBTQ	SYNC	EMWS	OPAR	PBEE	TSS40	NCS	DWQ	DW40	GAMX	-	-	-	-	-
0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0

The Option Register OR defines the address mask and the length of the RAS/CAS assertion time TCYC[3-0]. For fast implementation the read data from SDRAM can be latched after 4 clocks which requires a RAS enlargement of 1. In some cases a cycle enlargement of 2 cycles is required. Page Mode can be enabled The example shows a 16M SDRAM bank.

OR

TCYC3	TCYC2	TCYC1	TCYC0	AM27	AM26	AM25	AM24	AM23	AM22	AM21	AM20	AM19	AM18	AM17	AM16
0	0	1	0	1	1	1	1	0	0	0	0	0	0	0	0
AM16	AM14	AM13	AM12	AM11	FCM3	FCM2	FCM1	FCM0	BCYC1	BCYC0	-	PGME	SPS1	SPS0	DSSEL
0	0	0	0	0	0	0	0	0	0	0	0	0/1	0	0	1

The base register BR will be set at the end. It holds the base address which is system specific and the Valid bit to enable the bank.

BR

BA31	BA30	BA29	BA28	BA27	BA26	BA25	BA24	BA23	BA22	BA21	BA20	BA19	BA18	BA17	BA16
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
BA16	BA14	BA13	BA12	BA11	FC3	FC2	FC1	FC0	TRLXC	BACK40	CSNT40	CSNTC	PAREN	WP	V
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

10. Critical Timing for CPLD Synthesis

The most critical timing is the refresh wave form. It is not possible to change the CAS-to-RAS delay of refresh. The state machine must recognize the CAS-before-RAS signal event. This causes hard setup requirements for RAS_IN1-RAS_IN2 and CAS_IN0-CAS_IN3 during CPLD synthesis. The setup time must be equal and < 7ns.

The all-over delay from 360 through CPLD, SDRAM back to the 360 during read access can be larger than the cycle time. This access delay can be compensated by TCYC bits in the OR register.

The next critical timing is the race between positive edge of the delayed CLKOUT and the RAS address. The address hold delay must be larger than the CLK->CLKOUT propagation delay (see (1) in Figure 2). The 5V-3.3V address level shifter helps to delay the address in this case. The CLK->CLKOUT delay must be in the range of all other output delays RAS, CAS, WE, DQMB, etc).

11. Conclusion

The purpose of this application note is to advise users with existing MC68360 “QUICC” designs on how to interface adapt these designs to interface to SDRAM, which is now a more readily available and cost effective memory solution in comparison to DRAM. However, users should be aware that the performance of interfacing SDRAM to the MC68360 is limited because the MC68360 memory controller does not perform bursting.

This application note documents a hardware example of how to interface SDRAM to the MC68360 integrated communication processor, using simple hardware which has already been implemented on an adaptor board which interfaces directly to the MC68360 QUADS board.

References

It is important that users familiarize themselves with the following references for a better understanding of the terminology and general MPC8260 programming model.

- *MC68360 QUICC™ User’s Manual*; MC68360UM/AD Rev .1
- *MC68360 QUICC™ QUADS Hardware User’s Manual Rev 1*

Appendix A - Verified Verilog Code for CPLD

```
/*
SDRAM Adapter
by Michael Drozd
R12781@email.sps.mot.com
+49-89-92103-245

-----------------------------------------------
HISTORY:

rev 4:
CAS assignment mirrowed in read/write task

rev 3:
only 1 cycle between data access and refresh
as RAS precharge possible

rev 2:
RAS can go high before cas goes high during
read/write no page-mode
-----------------------------------------------

Adapter special for MC68360 (QUICC).
This CPLD is implemented on the SDRAM board which
connects to the QUADS evalboard.

It generates the signals for SDRAM DIMM module.
2 banks controlled by 2 CS.
23bit wide data.
16M, 64M modules and bigger are supported.

Control signals for bidir. 5.0V-3.3V levelshifter of the
databus are generated (BUSEN,BUSDIR).


The adapter does not support bank interleaving and powerdown.


*/

`timescale 1ns / 100ps

module ada (
// sdram_port
CAS,
RAS,
WE,
BA1,
BA0,
```

```
A10,
CKE,
CS0,
CS2,
DQM3,
DQM2,
DQM1,
DQM0,
CLKOUT,
ERROR, // ERROR LED must not be alight
// levelshifter port
BUSEN,  // 1=disable, 0=enable
BUSDIR,// 1=read SDRAM, 0=write SDRAM
//dramc_port
CAS_IN3,
CAS_IN2,
CAS_IN1,
CAS_IN0,
RAS_IN2,
RAS_IN1,
WE_IN,
A10_IN,
BA_IN1,
BA_IN0,
CLK,
RESET,
MODE// MODE=0 2 bank mode; MODE = 1 single bank mode
);

output RAS;
output CAS;
output WE;
output CKE;
output CS0,CS2;
output BA1,BA0;
output A10;
output DQM3,DQM2,DQM1,DQM0;
output CLKOUT;
output ERROR;
output BUSEN,BUSDIR;


input CLK;
input MODE;
input RESET;
input WE_IN;
input A10_IN;
input RAS_IN2, RAS_IN1;
input CAS_IN3;
input CAS_IN2;
input CAS_IN1;
```

```
input CAS_IN0;
input BA_IN1, BA_IN0;

wire rasx,casx;
wire ba0x,ba1x,resb,a10x;
wire [3:0] cas_inx;
wire clkx,clk_barx;
wire ras_in1x,ras_in2x;
wire resbx;

reg RAS;
reg CAS;
reg WE;
reg CKE;
reg DQM3;
reg DQM2,DQM1,DQM0;
reg BA1,BA0;
reg A10;
reg CS0,CS2;

reg[2:0] state;
reg [1:0] no_init;
reg resb1,resb2;
reg[1:0] bank;
reg[1:0] active_cs;
reg[3:0] init_count;
reg error_flag;
reg ras_reg,cas_reg;

assign ras_in1x = MODE ? ( RAS_IN2 | RAS_IN1) : RAS_IN1;
assign ras_in2x = MODE ? (!RAS_IN2 | RAS_IN1) : RAS_IN2;

assign rasx = ras_in2x & ras_in1x;
assign casx = CAS_IN3 & CAS_IN2 & CAS_IN1 & CAS_IN0;
assign cas_inx = {CAS_IN3,CAS_IN2,CAS_IN1,CAS_IN0};
assign CLKOUT = CLK;
assign resbx = RESET;
assign ba1x = BA_IN1;
assign ba0x = BA_IN0;
assign a10x = A10_IN;


assign BUSEN = rasx;
assign BUSDIR = WE_IN;
assign ERROR = error_flag;


task activate;
   input        a10x;
   input        ba0x;
   input        ba1x;
```

```
   begin
      CKE   = 1;
      RAS = 0;
      CAS = 1;
      WE  = 1;
      DQM3   = 1;
      DQM1   = 1;
      DQM2   = 1;
      DQM0   = 1;
      BA0    = ba0x;
      BA1    = ba1x;
      A10    = a10x;
CS0 = ras_in1x;
CS2 = ras_in2x;
   end
endtask

task auto_refresh;
   begin
      CKE   = 1;
      RAS = 0;
      CAS = 0;
      WE  = 1;
      DQM3   = 1;
      DQM1   = 1;
      DQM2   = 1;
      DQM0   = 1;
      BA1    = 0;
      BA0    = 0;
      A10    = 0;
CS0 = 1'b0;
CS2 = 1'b0;
   end
endtask

task burst_term;
   begin
      CKE   = 1;
      RAS = 1;
      CAS = 1;
      WE  = 0;
      DQM3   = 1;
      DQM1   = 1;
      DQM2   = 1;
      DQM0   = 1;
      BA1    = 0;
      BA0    = 0;
      A10    = 0;
CS0 = active_cs[0];
CS2 = active_cs[1];
   end
```

```
endtask

task load_mode_reg;
   input[1:0] bank;
   begin
      CKE   = 1;
      RAS = 0;
      CAS = 0;
      WE  = 0;
      DQM3   = 1;
      DQM1   = 1;
      DQM2   = 1;
      DQM0   = 1;
      BA1    = bank[1];
      BA0    = bank[0];
      A10    = 0;
CS0 = 1'b0;
CS2 = 1'b0;
   end
endtask

task nop;
   begin
      CKE   = 1;
      RAS = 1;
      CAS = 1;
      WE  = 1;
      DQM3   = 1;
      DQM1   = 1;
      DQM2   = 1;
      DQM0   = 1;
      BA1    = 0;
      BA0    = 0;
      A10    = 0;
CS0 = active_cs[0];
CS2 = active_cs[1];
   end
endtask

task idle;
   begin
      CKE   = 1;
      RAS = 1;
      CAS = 1;
      WE  = 1;
      DQM3   = 1;
      DQM1   = 1;
      DQM2   = 1;
      DQM0   = 1;
      BA1    = 0;
      BA0    = 0;
```

```
      A10    = 0;
CS0 = 1'b1;
CS2 = 1'b1;
   end
endtask

task precharge;
   input[1:0] bank;
   begin
      CKE   = 1;
      RAS = 0;
      CAS = 1;
      WE  = 0;
      DQM3   = 1;
      DQM1   = 1;
      DQM2   = 1;
      DQM0   = 1;
      BA1    = bank[1];
      BA0    = bank[0];
A10   = 0;
CS0 = active_cs[0];
CS2 = active_cs[1];
   end
endtask

task precharge_both_bank;
   begin
      CKE   = 1;
      RAS = 0;
      CAS = 1;
      WE  = 0;
      DQM3   = 1;
      DQM1   = 1;
      DQM2   = 1;
      DQM0   = 1;
      BA1    = 0;
      BA0    = 0;
A10   = 1;
CS0 = 1'b0;
CS2 = 1'b0;
   end
endtask

task read_write;
   input     WE_IN;
   input  [3 : 0] cas_inx;
   input[1:0]     bank;
   begin
      CKE   = 1;
      RAS = 1;
      CAS = 0;
```

```
      WE  = WE_IN;
      DQM3   = cas_inx[0];// most significant byte
      DQM2   = cas_inx[1];
      DQM1   = cas_inx[2];
      DQM0   = cas_inx[3];// least significant byte
      BA1    = bank[1];
      BA0    = bank[0];
      A10    = 0;
CS0 = active_cs[0];
CS2 = active_cs[1];
   end
endtask

task ram_disable;
   begin
      CKE   = 1;
      RAS = 1;
      CAS = 1;
      WE  = 1;
      DQM3   = 1;
      DQM1   = 1;
      DQM2   = 1;
      DQM0   = 1;
      BA1    = 0;
      BA0    = 0;
      A10    = 0;
CS0 = 1'b1;
CS2 = 1'b1;
   end
endtask

task error_monitor;
   begin
      CKE   = 1'b0;
      RAS = ras_reg;
      CAS = cas_reg;
      WE  = 1'b0;
      DQM3   = init_count[3];
      DQM1   = init_count[2];
      DQM2   = init_count[1];
      DQM0   = init_count[0];
      BA1    = no_init[1];
      BA0    = no_init[0];
      A10    = 1'b0;
CS0 = 1'b1;
CS2 = 1'b1;
   end
endtask


task x_default;
```

```
   begin
      CKE   = 1'bx;
      RAS = 1'bx;
      CAS = 1'bx;
      WE  = 1'bx;
      DQM3   = 1'bx;
      DQM1   = 1'bx;
      DQM2   = 1'bx;
      DQM0   = 1'bx;
      BA1    = 1'bx;
      BA0    = 1'bx;
      A10    = 1'bx;
 CS0 = 1'bx;
CS2 = 1'bx;
   end
endtask
always @(posedge CLK)
begin
resb2 = resb1;
resb1 = resbx;
end

always @(posedge CLK) // latching bank address like the sdram itself
begin
if((resb2 == 0)||(resbx==0))
begin
bank = 2'b0;
end
else if((state==0)&&(rasx==0)&&(casx==1))
begin
bank[0] = ba0x;
bank[1] = ba1x;
end
end

assign clk_barx = ~CLK ;

always @(posedge clk_barx)
begin
if((resb2 == 0)||(resbx==0))
begin
state = 0;
init_count = 0;
no_init = 1;
error_flag = 0;
active_cs = 2'b11;
end
else if(error_flag == 1)
begin
state = 0;
active_cs = 2'b11;
```

```
end
else if((state == 0)&&(rasx==0)&&(casx==1))
begin
state = 1;// activate wait
active_cs[1] = ras_in2x;
active_cs[0] = ras_in1x;
end
else if((state == 1)&&(rasx==0)&&(casx==1))
begin
state = 1;// read/write
end
else if((state == 1)&&(rasx==0)&&(casx==0))
begin
state = 2;// read/write wait
if(no_init==1)
begin
if(init_count==9)
begin
init_count=0;
no_init=2;
end
else
begin
init_count = init_count+1;
end
end
end
//else if((state == 2)&&(rasx==0)&&(casx==0))
else if((state == 2)&&(casx==0))// 360er special
begin
state = 2;// read/write wait
end
else if((state == 2)&&(rasx==0)&&(casx==1))
begin
state = 1;// activate wait
end
else if(((state == 1)||(state == 2))&&(rasx==1)&&(casx==1))
begin
state = 6;// precharge
end
else if((state == 6)&&(rasx==1)&&(casx==1))
begin
state = 7;// precharge wait
end
else if((state == 6)&&(rasx==1)&&(casx==0))
begin
state = 3;// refresh pre
active_cs = 2'b11;
if(no_init==2)
begin
no_init=0;
```

```
endend
else if((state == 7)&&(rasx==1)&&(casx==1))
begin
state = 0;// idle
active_cs = 2'b11;
if(no_init==2)
begin
no_init=0;
end
end
else if((state == 7)&&(rasx==1)&&(casx==0))
begin
state = 3;// refresh pre
active_cs = 2'b11;
if(no_init==2)
begin
no_init=0;
end
end
else if((no_init==0)&&(state == 0)&&(rasx==1)&&(casx==0)) // refreshes are possible before DCTR is set, refresh
only if cs is asserted
begin
state = 3;// refresh pre
end
else if((state == 3)&&(rasx==1)&&(casx==0))
begin
state = 3;// refresh pre wait
end
else if((state==3)&&(rasx==0)&&(casx==0))
begin
state = 4;// refresh
end
else if((state==3)&&(rasx==1)&&(casx==1))
begin
state = 0;
end
else if((state == 4)&&(rasx==0)&&(casx==1))
begin
state = 5;// refresh wait
end
else if((state == 5)&&(rasx==0)&&(casx==1))
begin
state = 5;//  refresh wait
end
else if((state == 5)&&(rasx==1)&&(casx==1))
begin
state = 0;// idle
end
else if((state == 0)&&(rasx==1))
begin
state = 0;
```

```
end
else if(no_init==0)
begin
error_flag = 1;
ras_reg = rasx;
cas_reg = casx;
end
end



always @(state or cas_inx or rasx or casx or error_flag or no_init or bank or a10x or WE_IN or init_count or
active_cs or ras_in1x or ras_in2x or ba1x or ba0x or ras_reg or cas_reg)
begin
if(error_flag == 1)
begin
error_monitor;
end
else if(no_init > 0)
begin
if((state==1)&&(rasx==0)&&(casx==0)&&(WE_IN==0))
begin
case(init_count)
0 : begin
   precharge_both_bank;
   end
1 : begin
   auto_refresh;
   end
2 : begin
   auto_refresh;
   end
3 : begin
   auto_refresh;
   end
4 : begin
   auto_refresh;
   end
5 : begin
   auto_refresh;
   end
6 : begin
   auto_refresh;
   end
7 : begin
   auto_refresh;
   end
8 : begin
   auto_refresh;
   end
9 : load_mode_reg(bank);
```

```
default : x_default;
endcase
end
else
nop;
end
else
case (state)
0 : if((rasx==0)&&(casx==1))
activate(a10x,ba0x,ba1x);
else
idle;
1 : if(casx==0)
read_write(WE_IN,cas_inx,bank);
else
nop;
2 : nop;
3 : nop;
4 : auto_refresh;
5 : nop;
6 : precharge(bank);
7 : nop;
default : x_default;
endcase
end


endmodule
```

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

freescale™ semiconductor