SYNTHESIZABLE SYNCHRONOUS-DRAM CONTROLLER CORE

Jeung Joon Lee
www.OpenCores.org

1.0 SYSTEM FEATURES

- Core supports 100MHz operation, PC100 compliant
- Flexible byte, word (16bit) and long word (32bit) accessing through the use of `mp_size[1:0]`
- Interfaces readily, without further modifications, to 2M x 32 SDRAMs such as:
  Samsung KM432S2030CT
  Fujitsu MB81F643242B
  and other compatibles.
- Flexible refresh-cycle generation, including "burst" refresh and normal refresh, and everything in between.
- The core performs the SDRAM initialization sequence transparently to the host, including Mode Register programming.
- During normal operation, the Mode Register can be updated by the host through the use of `sdram_mode_set_l`
- Built-in comprehensive synthesizable SDRAM tester.

2.0 GENERAL OVERVIEW

The synthesizable *Synchronous DRAM controller IP* is a complete design solution which allows virtually any type of microprocessor, microcontroller and DSP to interface to large capacity SDRAMs effortlessly.

All of the low-level SDRAM functions such as address demultiplexing, refresh generation and busy generation are handled by the IP transparently to the host. The non-trivial initialization sequence required by most SDRAMs is also performed transparently to the host upon powerup or system reset.

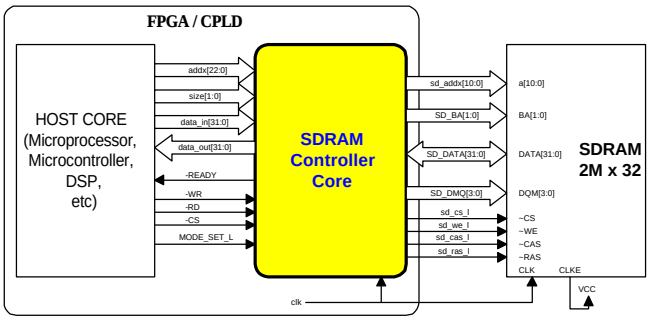

Figure 1 Typical interface block diagram

As with most DRAM controllers, this SDRAM controller provides to the host an SRAM-like bus interface. Figure 1 illustrates the typical connection block diagram. By embedding the SDRAM controller IP along with processor cores, a highly integrated System-on-Chip (SoC) designs are possible.

The host's write and read bus operation is completely asynchronous in relation to the SDRAM controller, much like the access to a SRAM. The SDRAM controller generates a busy output which the host can use as handshake control. This avoids the need of fixed wait states.

If a write or read request is made by the host while the SDRAM is being accessed, the host is placed in busy state until the pending access is completed. Similarly, if a request is made while a refresh operation is in progress, the host is busied until its completion.

Essentially, a write to the SDRAM controller represents a write to the SDRAM, while the read from the SDRAM controller represents a read from the SDRAM.

To write, the host drives the bus with a typical SRAM-like write operation. The SDRAM controller then generates the appropriate signals to transfer the host's data to the specified address of the SDRAM.

Figure 2 illustrates a typical host write to the SDRAM controller. Although a negative edge triggered host is illustrated, identical sequence of events would occur for a positive edge triggered host.

The handshaking for a typical write is as follows:

clock 1:
The host asserts -CS along with valid address, and size.

clock 2:
The host asserts -WR along with valid data

clock 3:
The host samples "busy" as low, and holds (extends) the bus

clock 4:
The host samples "busy" as high, so prepares to terminate the bus

clock 5:
The host deasserts -WR

clock 6:
The host deasserts -CS and terminates the bus cycle

Note that it is **not necessary** to have -CS and -WR be asserted and deasserted with any amount of delay. That is, `Ta` and `Tb` in figure 2 can be 0. However, `Tah` must be non-zero.

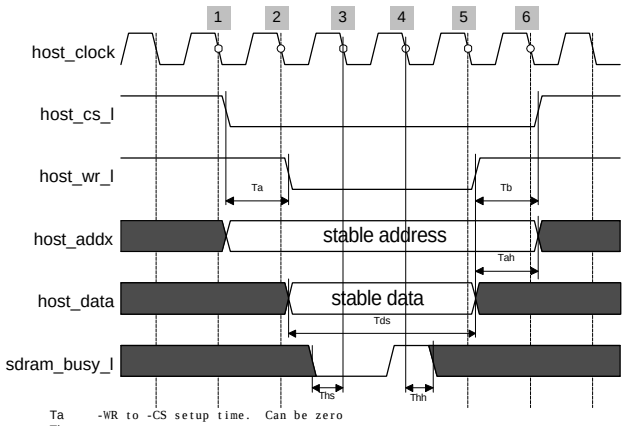

Figure 2 This timing diagram illustrates the interaction of the host and SDRAM controller during a write cycle. This plot shows a negative edge triggered host, but the sequence is identical for a positive edge triggered host.

Similarly, to read, the host drives the bus with a typical read operation. The SDRAM controller keeps the host in busy state until data is retrieved from the SDRAM and made available to the host.

Figure 3 illustrates a typical host read from the SDRAM controller. Although a negative edge triggered host is illustrated, identical sequence of events would occur for a positive edge triggered host.

The handshaking for a typical read is as follows::

clock 1:
The host asserts -CS, along with valid address

clock 2:
The host asserts -RD

clock 3:
The host samples "busy" as low, and holds (extends) the bus

clock 4:
The host samples "busy" as high and prepares the terminate the bus. The host can read the data at this cycle.

clock 5:
- The host deasserts -RD, and reads the data if it has not at previous cycle.
- The SDRAM controller acknowledges the deassertion of -RD by releasing the data bus.

clock 6:
The host deasserts -CS and terminates the bus cycle

As with writes, it is **not necessary** to have -CS and -WR be asserted and deasserted with any amount of delay. That is, Ta and Tb in figure 3 can be 0. However, Tah must be non-zero.

Figure 3 also shows two data buses. The top bus illustrates the case when the data bus is bi-directional. The bottom bus is the case when the bus is unidirectional. The bi-directional data bus is expected when the SDRAM core is to be part of a external data bus. The unidirectional data bus is expected when the SDRAM controller is to be embedded along with other cores in a SoC type of design. Note that most FPGA/CPLD do not support internal tri-state buses, and thus bi-directional buses needs to be converted into unidirectional buses.

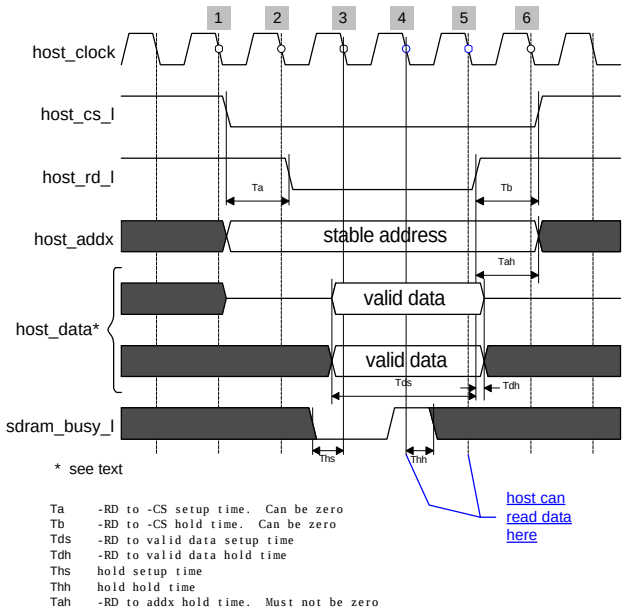

Figure 3 This plot illustrates a typical read bus event from the SDRAM controller

2.1 IP Signal Conventions

The signaling conventions used in the core are detailed below:

- Any active low signal is appended with "_l". For example, the active low system reset is denoted as sys_rst_l
- Any signals connecting to the host is preceded with "mp_". For example, the data bus connecting the controller to the host is denoted as mp_data[15:0].
- Any signals connecting to the SDRAM is preceded with "sd_". For example, the data bus connecting the controller to the SDRAM is denoted as sd_data[15:0].

2.2 IP Pinout

The pinout for the default IP (target the typical 2Mx32 SDRAM) consists of 2 system signals, 42 host side signals and 36 SDRAM side signals. The tables below describes the functions of the pins.

System Level Signals	Direction	Description
sys_rst_l	input	System level active low reset. Brings all state machines into the intial state. The reset is asynchronous.
Sys_clk	input	This is the main clock used to clock the state machine. An optional clock divider can be used internally.

Host Side Signals	Direction	Description
mp_data[31:0]	bidi	Bidirectional data bus connected to the host (micro, DSP, etc).
mp_addx[22:0]	input	Linear address bus connected to the host. Upto 8Mbytes can be addressed with the default core.
mp_cs_l	input	Active low chip-select input from the host. This signal must go low on all reads or writes meant to the SDRAM. This is an asynchronous signal.
mp_rd_l	input	Active low signal which the host uses to indicate to the controller that the current bus transaction is a read from the SDRAM. This is an asynchronous signal.
mp_wr_l	input	Active low signal which the host uses to indicates to the controller that the current bus transaction is a write to the SDRAM
sdram_busy_l	output	This signal is used as a bus handshake control. After the host asserts -WR or -RD, it needs to sample this signal. Whenever it is low, the host needs to place the bus in a holds state, until it becomes high. If the bus cycle was a read, the host can read the data bus when the sdram_busy_l ha been sampled high.
sdram_mode_set	input	A write to the controller with this signal low will update the SDRAM's mode register.
mp_size[1:0]	input	Specifies the size of the data bus. 00 = 32 bit 01 = 8 bit 10 = 16 bit

SDRAM Side Signals	Direction	Description
sd_data[31:0]	bidi	This is the bidirectional data bus conneting the controller and the SDRAM.
sd_addx[10:0]	output	This is the multiplexed address bus connected to the SDRAM.

sd_wr_l	output	This is the active low write signal.
sd_cs_l	output	This is the active low SDRAM chip select signal.
sd_ras_l	output	This is the active low row address select signal.
sd_cas_l	output	This is the active low colum address select signal.
sd_dqm[3:0]	output	This is the active high byte mask signal. When high, no data can be read to written to the SDRAM byte.
sd_ba[1:0]	output	This is the bank select. It is also known as ADDX[13:12].
sd_clk	output	This is the clock driving the SDRAM.

2.3 SDRAM Controller IP Files

The SDRAM controller IP is composed of the following files:

```
SDRAM.V
HOSTCONT.V
SDRAMCNT.V
MICRO.V
INC.H
TST_INC.H
```

SDRAM.V is the top hierarchy module. It serves as a wrapper and instantiates the lower sub modules, HOSTCONT.V , SDRAMCNT.V and MICRO.V. Additionally, as an option, it contains a system clock divider.

SDRAMCNT.V is the main state machine which generates all of the control signals to the SDRAM. It also generates signals to control the HOSTCONT.V module. This module is responsible for essentially governing the SDRAM.

HOSTCNT.V contains the logic in the datapath connecting the host bus and the SDRAM bus. Most of the datapath logic receives the control signals from SDRAMCNT.V

MICRO.V is the optionally synthesizable tester. It essentially acts as a host to the SDRAM controller core. Based on the selected test, it will instruct the SDRAM controller to do a series of repeating write/read tests to the SDRAM.

INC.H contains a set of globals which determine the interval of refresh and the number of refreshes per interval. The user specifies the SDRAM clock speed in this file. The synthesizable test core also gets to be enabled here. Finally, the state machine's in SDRAMCNT.V is defined here.

TST_INC.H defines the type of test to perform when the test core is enabled.

2.4 Built-In Synthesizable SDRAM Tester

The SDRAM IP contains a set of synthesizable test cores which can aid the testing and development of embedded SDRAM controllers greatly. The test core is useful primarily on FPGA/CPLD platforms.

The test cores emulates a typical microprocesors' write and read bus cycles. When enabled, the tester becomes a host to the SDRAM controller. The tester/controller pair can then be used to test the performance and feature of a particular SDRAM chip quickly. This can be of particular use when the simulation model of the SDRAM is unavailable or if the simulation is not possible.

The test core is enabled at compile time through the use of `define statement. The header file tst_inc.h contains a set of define statements which when uncommented enables the test core. At the time of the writing of this documentation, three test cores are available:

(I) do_single_burst_write_read_test
This test core writes a specified amount of words to the SDRAM controller. It then delays for a specified amount of clocks, and proceeeds to read from the just written addresses of the SDRAM. This delay and read cycles indefinetely.

(II) do_burst_write_read_test
This test core writes a specified amount of words to the SDRAM controller. It then delays for a specified amount of clocks, and then proceeds to read from the just written addresses of the SDRAM. The write-delay-read cycles indefinetely.

(III) do_read_write_test
This test core performs a single write, then delays for a specified clock ticks, then performs a single read from the just written address. The write-delay-read cycles indefinitely.

3.0 OVERVIEW OF BASIC SDRAM OPERATION

This section is not intended to discuss all of the detailed operation of a SDRAM, but does cover the essentials of a typical read, write and refresh operations. The reader should consult the specification manual for the particular memory for further details.

The majority of SDRAM employs a simple set of signal combinations called *commands* to carry out the basic IO. Most of these commands are one clock cycle in duration, and are clocked by the SDRAM on the rising edge. A proper setup and hold times must be observed. A sequence of these commands comprises the primitive operation of read, write and refresh. Figure 4 illustrates a typical set of command used by most SDRAMs.

The 1M x 16 SDRAM (IP default) is organized as 512K x 16 x 2 banks. The bank selection is done by a pin called BA (bank address) or sometimes also called A11. This IP refers to as BA, and from this point forward, A11 and BA will be used interchangeably.

The address organization of these SDRAMs are 11 rows by 8 columns. During the row address strobing, A[10:0] provides the row to select and during the column strobing A[7:0] provides the column to select (A[10:8] should be set to logic low).

· *Mode register set command*
This command is used to program the SDRAM's mode register. The mode register controls the operation of the SDRAM, including the CAS latency, burst type, burst length, test mode and other vendor specific options. Most SDRAMs do not initialize the mode register upon power up, thus it is critical this register be initialized prior to the normal use. The SDRAM controller initializes the mode register following every system reset with a default value. During the mode register programming, the SDRAM receives the data from A[10:0] and BA, rather than from the data bus.

· *Row address strobe and bank active command*
This command is used to select the bank and the row where the data access is to take place. The BA input selects the bank, while A[10:0] provides the row address.

- *Precharge*
The precharge command is used to begin the precharge operation to the selected bank(s). When A[10] is high, both banks are activated, while when A[10] is low, the bank selected by BA is activated.

· *Column address and write command*
This command is used to select the column of the selected bank where the write is to take place. It is important that the bank selected at the row address strobe be selected again. The address bus A[7:0] selects the column. A[10:8] should be logic low.

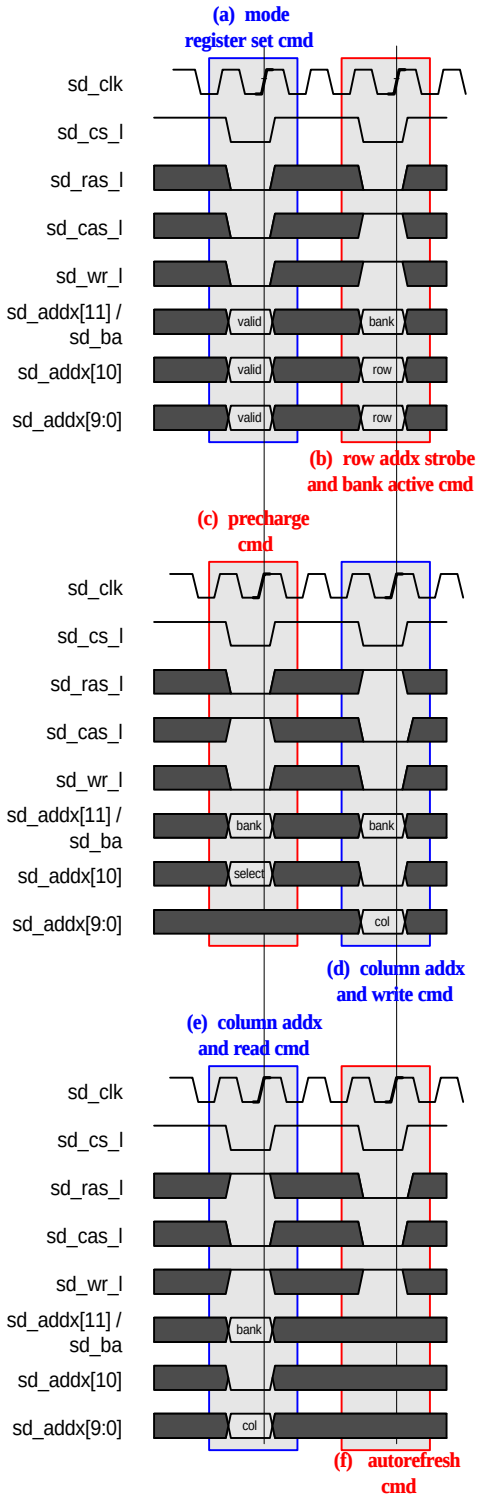

Figure 4 (a) - (f) This figure illustrates the basic commands most SDRAMs recognize. The proper combination of these command comprises the read, write and refresh operations.

· *Column address and read command*
This command is used to select the column of the selected bank where the read is to take place. The address bus A[7:0] selects the column. A[10:8] should be logic low.

· *Auto refresh command*
This command is used to start the internal refresh cycle. An internal row counter is adjusted to point to the next row.

3.1 Basic Operations

In order to take full advantage of the capabilities of the SDRAM, the specific application has to be known. This IP is meant to be as generic as possible, and thus the SDRAM controller IP only support the basic types of READ, WRITE and refresh operations. Some of the more advanced features such as, burst write and burst reads are not presently supported.

The following sections describes the basic operations of single word write, single word read and refresh.

3.1.1 Basic READ Operation

The basic read operation consists of reading a long word from the SDRAM. The read operation consists of the following command sequences:

1. row address strobe and bank active command,
2. wait to meet Trcd (1 NOP cycle),
3. column address and read command
4. wait to meet CAS latency
5. Precharge all bank command
6. wait to meet Trp
7. wait to meet Trc before a new row activate command

Trcd is row-col-delay parameter. It specifies the minimum time the SDRAM can recognize the column set command from the last row set command. For most PC100 SDRAM this is 20nS, therefore 1 Tclk must be inserted between the row and column set commands.

Once the SDRAM receives the column address strobe command, it makes the data available on its bus *CAS latency* cycles later. The CAS latency is programmed through the mode register. In theory this value can be programmed from 2 to 7 cycles, but most SDRAM vendors limit the choices to 2 or 3 cycles.

After the data has been read, the SDRAM controller issues a precharge-all command to ready for the next data access.

3.1.2 Basic WRITE Operation

The basic write operation consists of writing a byte/word or longword into the SDRAM. The write operation consists of the following command sequences:

1. row address strobe and bank active command
2. delay to meet Trcd (1 NOP cycle)
3. column address and write command
4. delay to meet Tras
5. precharge-all bank command
6. delay to meet Trp
7. delay to meet RC before another command

Similar to the case of single word read, the NOP operation is required between the row and the column strobing to meet the T_{rcd} requirement. Recall that most 100MHz SDRAM has this parameter as 20nS minimum.
The write operation inherently takes less cycles than the write because there are not CAS latencies involved. The SDRAM latched in the data on the rising edge of column strobe command. Figure 6 illustrates this concept.

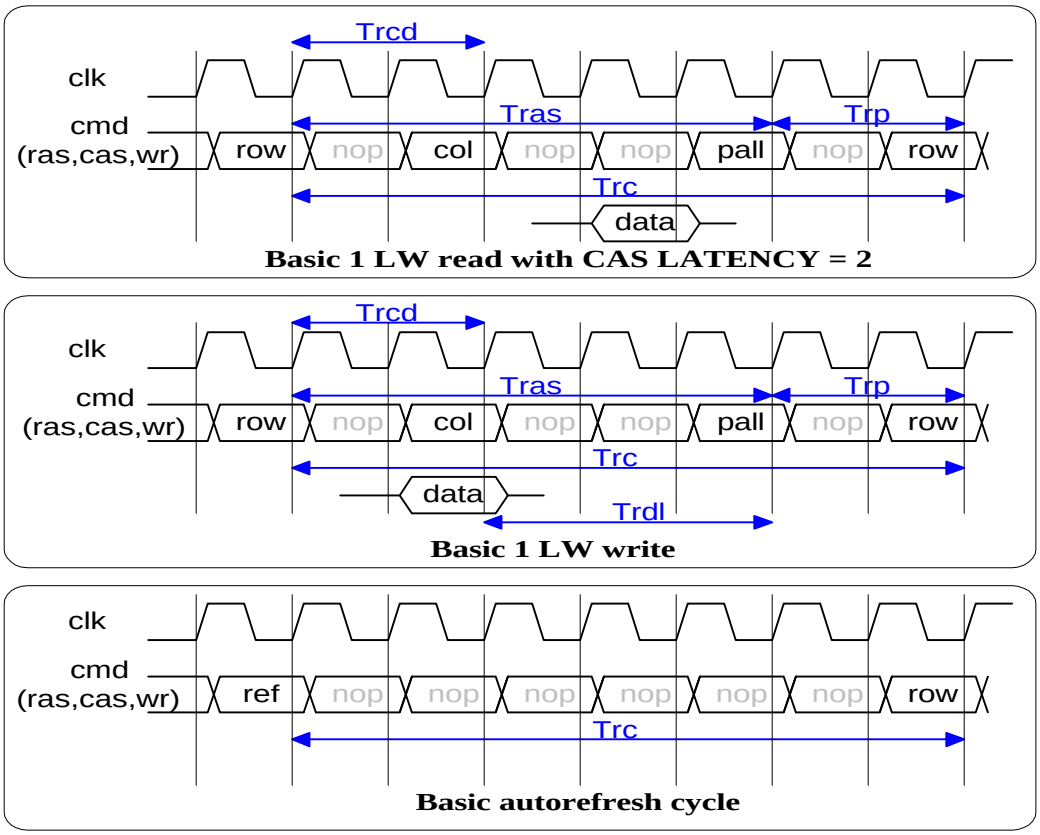

Figure 5 The basic read, write and refresh operations

3.1.3 Refresh Operation

The refresh operation consists of initiating the internal auto-refresh cycle of the SDRAM, and consists of the following command sequence:

1. N number of autorefresh commands
2. delay to meet Trc before issuing another command

N represents the number of rows to be refreshed. Following the autorefresh command, the SDRAM can not accept any new commands until T_{rc} later. This parameter varies by SDRAM vendor, and thus must be referenced by case basis, but typically ranges from 60 to 90nS for a 100MHz rated SDRAM.

The number of autorefresh cycles, N, to be issued depends on the type of refresh desired. As previously mentioned, in a typical 2M x 32 SDRAM (IP default), there are 2048 rows per bank. And the typical refresh interval is 32mS. For an evenly distributed refresh type, this represents 1 refresh operation every 15.6uS. Alternatively, a "burst" type of refresh can perform all 2048 refresh operations every 32mS. The latter approach is highly advantageous for very slow hosts. Thus N can take a value from 1 to 2048.

The refresh interval is set by defining the refresh frequency in `inc.h`

```
// Refresh Frequency in Hz.
//   For burst  refresh use 33Hz    (30mS)
//   For normal refresh use 66666Hz (15uS)
```

```
`define Frefresh                        66666
```

The number of refreshes performed is defined by the parameter below in inc.h

```
// The number of refreshes done during normal refresh
    cycle.
// Set this to be 2048 for "burst"   refreshes, and
// set this to be 1    for "regular" refreshes
`define auto_ref_cntr_limit             1
```

3.2 SDRAM Mode Register and Initialization Sequence

Upon powerup, the SDRAM's mode register must be initialized for proper operation. This is due to the fact that most SDRAMs do not initialize the mode register following a reset. However, before the register can be initialized, the SDRAM must be subjected to an initialization sequence.

Notice that following power up (or reset), a repeated number of autorefresh command is necessary. The particulars of the powerup sequencing is also highly dependent on the SDRAM maker, so consult the specification. In general, however, the sequence consists of the following:

1. After a stable power (Vcc) has been reached, the SDRAM should see a stable clock for about 200uS. During this time, no valid command should be issued.
2. Both banks must be precharged (precharge command)
3. one or more auto-refresh commands must be issued.
4. The mode register can now be initialized.

Steps 3 and 4 can be in any order. The parameter which sets the number of auto-refreshes in the step 3 above is in inc.h file as:

```
// The number of refreshses done at power up.
// This varies by vendors.  Be sure to referer your DRAM
// vendor spec.  It is set to 3 by default
`define power_up_ref_cntr_limit         3
```

As previously mentioned, during the mode-register programming, the SDRAM receives the initialization data through the address bus and not through the data bus. Following the powerup, the SDRAM controller IP initializes the mode register with a default value of 12'h020, which corresponds to:

```
// DEFAULT MODE-REGISTER values
// The below is programmed to the mode regsiter at
// powerup
`define default_mode_reg_BURST_LENGHT   3'b000
`define defulat_mode_reg_BURST_TYPE     1'b0
`define default_mode_reg_CAS_LATENCY    3'b010
```

And the above data is written to the Mode Register during the step 4 outlined above.

Refer to figure 6 for the meaning of the mode register bit fields. Most SDRAMs follows the bit field definition illustrated, but make sure to consult the specific SDRAM vendor specs as they might contain subtle differences.

4.0 CUSTOMIZING THE CORE TO WORK FOR YOU

The SDRAM controller core has been designed so it can be as customizable as possible. As the core matures, it's customizing capability will also grow.

Most of the user adjustable parameters and features are found in the include file called inc.h, and are heavily commented.

Enabling the built-in tester

As previously mentioned, the SDRAM controller core contains a synthesizable tester, which acts as a host to the SDRAM controller and issues a series of write/read operations to the SDRAM. This is a handy feature if you want to verify that the SDRAM controller core works with the SDRAM.

```
// Uncomment below to use the microprocessor bus simulator
// This will allow the synthesis of a small tester, which
sends
// a series of data sequences to the SDRAM controller,
then reads
// back and verifies that the data is correct.
// Once you enable this option, choose the test mode in
// the file "tst_inc.h"
// ====================
//`define simulate_mp
```

Defining the type of host data bus

The data bus to/from the host can be a single 32 bit bidirectional bus or it can be 2 separate 32 bit unidirectional bus. The former case is best used when the SDRAM controller core is to be embedded alone in a FPGA/CPLD and talks to an external host. This obviously saves a lot of valuable pins.

The latter case is to be used when the host (micro, DSP, etc) is to be embedded along with the SDRAM controller on the same FPGA/CPLD. Most FPGA/CPLD's do not support internal bidirectional buses, and may cause fitter problems.

```
// Comment the below for bidirectional bus, and UNcomment
for unidirectional
`define    databus_is_unidirectional
```

Enabling Data Bus Aligner

When the data bus aligner is enabled, all data which are not long words (32bit) will be aligned to the low order bus. For example, when the host writes a word (16bit) to the SDRAM, it needs to present the data to the SDRAM controller on the low order 16 bits of the data bus. Similarly, when the host request a word of data from the SDRAM controller,. It will receive the data aligned to the lower 16 bits of the data bus. The data aligner is handy, if an 8 bit host is being interfaced to the 2Mx32 SDRAM, and all of the data space want to be addressed. Enabling the data aligned may however, slow down the system performance due to mux delays.

When the aligner is disabled, the data read/written will be expected to be in the correct byte order location. For example, when the host writes a byte of data to the SDRAM at address ending in 0x2, the SDRAM controller will transfer whatever data at data[23:16] to the SDRAM. Similarly, when the host request a byte of data at address location ending in 0x3, it will get the data at bit location [31:24].

```
//`define align_data_bus
```

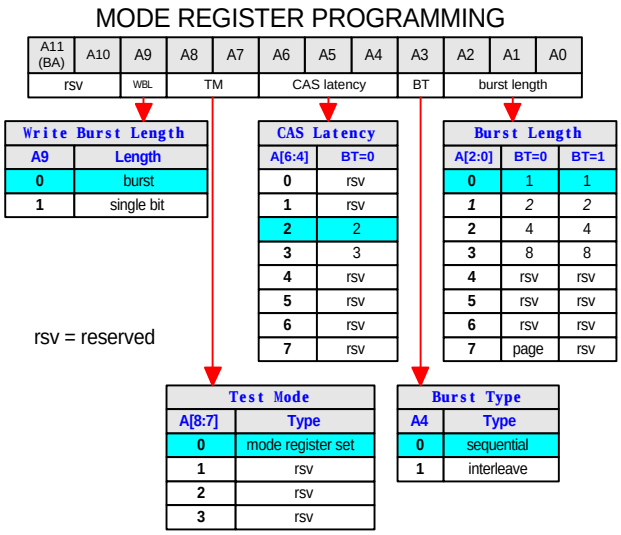

Figure 6 This plot illustrates the mode register definition for Samsung KM416S1120 SDRAM. Other SDRAMs are similar. The highlighted values is what the SDRAM controller initializes at powerup.

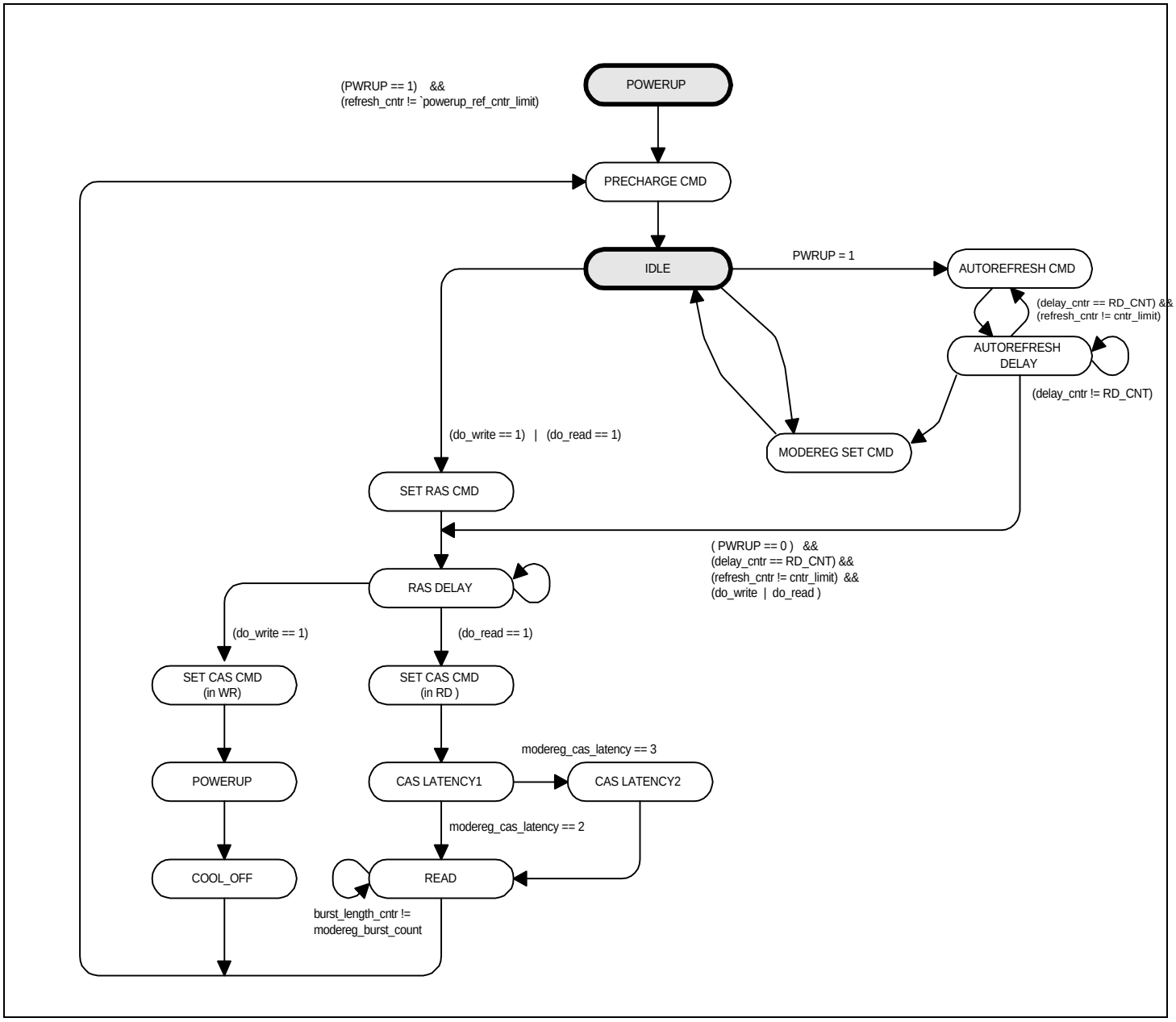

Figure 7 This is the state flow of the SDRAM controller state machine