7 SDRAM_CONTROL_4PORT Module

7.1 Description

IS42R16320D SDRAM is used to store image data. Data source (RAW2RGB module) and data consumer (VGA CONTROLLER module) communicate to SDRAM by SDRAM_CONTROL_4PORT module. This module contains a SDRAM controller. The 16-bit SDRAM data port is regarded as 4 virtual data ports (2 read, 2 write). The 30-bit input RBD data is buffered in two 16-bit-width write FIFOs which write data to SDRAM. In the meanwhile, to meet the read request of the data consumer (VGA CONTROLLER module), the RGB data stored in SDRAM previously is read out and buffered in two 16-bit-width read FIFOs. This module forms a complete frame buffer.

The structure of SDRAM_CONTROL_4PORT module is demonstrated below.

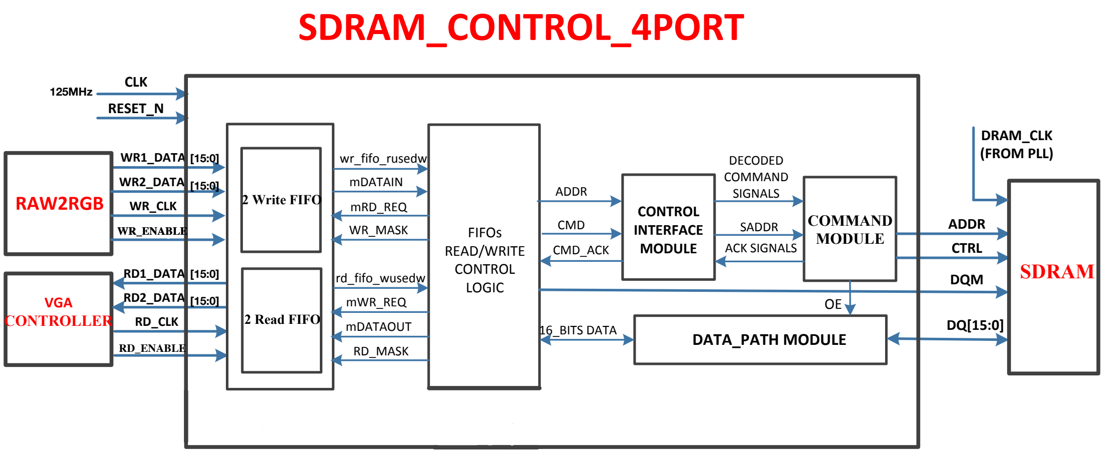

Figure 7-1 SDRAM_CONTROL_4PORT Module Structure

7.1.1 Data Flow

RGB format image data from RAW2RGB module is written to two write FIFOs in SDRAM_CONTROL module continuously. VGA CONTROLLER reads data from two read FIFOs continuously. The FIFOs READ/WRITE CONTROL LOGIC generates read/write request and address to the CONTROL INTERFACE sub module according to the state of four FIFOs. The CONTROL INTERFACE sub module handles not only the read/write request from FIFOs READ/WRITE CONTROL LOGIC, but also initial request or refresh request generated by its internal timer according to SDRAM timing sequence. The CONTROL INTERFACE sub module sends decoded command signals to the COMMAND sub module and command acknowledge signal to FIFOs READ/WRITE CONTROL LOGIC. The COMMAND module uses decoded command signals to generate operations conforming to SDRAM protocol and control the signals output to SDRAM. Controlled by OE signal generated by COMMAND sub module, the DATA_PATH sub module controls the direction of data line to make it simultaneous to the corresponding operations to SDRAM.

7.1.2 Using Four FIFOs

The input data is from RAW2RGB module and the output data is transferred to VGA CONTROLLER module. This means the read/write operations between external modules and SDRAM_CONTROLL_4PORT module need high continuity but comparatively low speed. On the other hand, the read/write between SDRAM_CONTROLL_4PORT module and SDRAM do not need continuity, and the speed is high. For the reasons above, asynchronous dual clock FIFOs are needed as input/output buffer. The data width of FIFOs is 16 bit, and the depth should be larger than one read/write burst length.

IS42R16320D SDRAM is used to store data. It has 4 banks, the data width of which is 16 bit. Since 30-bit data is needed for one pixel (R,G,B each need 10-bit data), the data of one pixel can't be write to/read from SDRAM in one operation. Also the 30-bit data can't share the same column/row address if it is stored in one bank of SDRAM, because the data width is only 16.

For convenience, we use two banks of SDRAM to store data of each pixel, as is demonstrated in the figure below. In this way, we build 4 virtual data ports for the 16-bit SDRAM data port (2 read, 2 write). We split the data of one pixel into two parts and store them in two banks separately, which means they can have the same column/row address. When the pixel data needs to be read out, these two parts are combined together to form a complete RGB-data.

Besides, to enhance the read/write speed, Full-Page Burst Mode is used in read/write transactions. For detail about Full-Page Burst Mode, please refer to Chapter 7.3.2

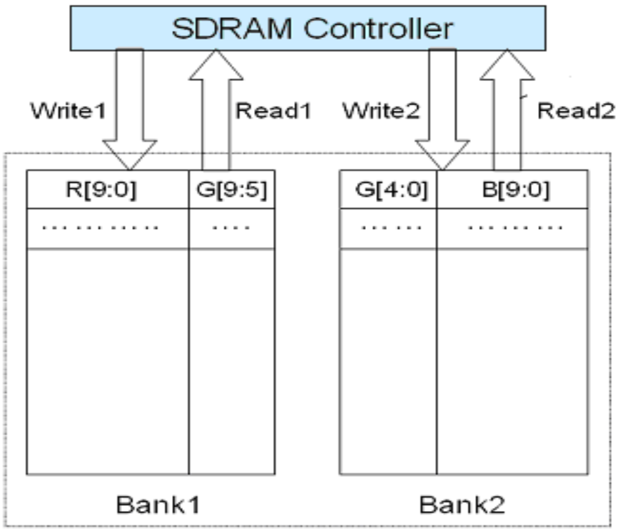

Figure 7-2 Four Port FIFOs Structure

7.2 Interface

Port	Direction	Width	Description
CLK	Input	1	Module clock
RESET_N	Input	1	Module reset active low, asynchronous
WR1_DATA	Input	`DSIZE	Write FIFO 1 Input data
WR_ENABLE	Input	1	Write request to Write FIFO 1
WR1_ADDR	Input	`ASIZE	Write start address of Write FIFO 1
WR1_MAX_ADDR	Input	`ASIZE	Write max address of Write FIFO 1
WR1_LENGTH	Input	9	Write length of Write FIFO 1
WR1_LOAD	Input	1	Write register load input of Write FIFO 1
WR1_CLK	Input	1	Write clock of Write FIFO 1
WR2_DATA	Input	`DSIZE	Write FIFO 1 Input data
~~WR2~~	~~Input~~	~~1~~	~~Write request to Write FIFO 2~~
WR2_ADDR	Input	`ASIZE	Write start address of Write FIFO 2
WR2_MAX_ADDR	Input	`ASIZE	Write max address of Write FIFO 2
WR2_LENGTH	Input	9	Write length of Write FIFO 2
WR2_LOAD	Input	1	Write register load input of Write FIFO 2
WR2_CLK	Input	1	Write clock of Write FIFO 2
RD1_DATA	Output	`DSIZE	Read FIFO 1 output data
RD_ENABLE	Input	1	Read request to Read FIFO 1
RD1_ADDR	Input	`ASIZE	Read start address of Read FIFO 1
RD1_MAX_ADDR	Input	`ASIZE	Read max address of Read FIFO 1
RD1_LENGTH	Input	9	Read length of Write FIFO 1
RD1_LOAD	Input	1	Read register load input of Read FIFO 1

RD1_CLK	Input	1	Read clock of Write FIFO 1
RD2_DATA	Output	`DSIZE	Read FIFO 2 output data
~~**RD2**~~	~~Input~~	~~1~~	~~Read request to Read FIFO 2~~
RD2_ADDR	Input	`ASIZE	Read start address of Read FIFO 2
RD2_MAX_ADDR	Input	`ASIZE	Read max address of Read FIFO 2
RD2_LENGTH	Input	9	Read length of Write FIFO 2
RD2_LOAD	Input	1	Read register load input of Read FIFO 2
RD2_CLK	Input	1	Read clock of Write FIFO 2
SA	Output	12	SDRAM address
BA	Output	2	SDRAM bank address
CS_N	Output	2	SDRAM chip selects Active low
CKE	Output	1	SDRAM clock enable
RAS_N	Output	1	SDRAM Row Address Strobe
CAS_N	Output	1	SDRAM Column Address Strobe
WE_N	Output	1	SDRAM write enable
DQ	Output	`DSIZE	SDRAM data bus
DQM	Output	`DSIZE/8-1	SDRAM data mask

* User can decide `ASIZE and `DSIZE.`ASIZE should no more than 23(the space of one SDRAM bank is 8M) and `DSIZE should no more than 16. In this system, `ASIZE is 23 and `DSIZE is 16.

Table 7-1 SDRAM_CONTROL_4PORT module Interface

7.3 Sub modules and Logic

7.3.1 FIFO

1. Description

Altera provides dual-clock FIFO (DCFIFO) mega functions. User can use the FIFO parameter editor launched from the MegaWizard Plug-In Manager in the Quartus II software to build the FIFO mega functions.

The FIFO functions are mostly applied in data buffering applications that comply with the first-in-first-out data flow in asynchronous clock domains. The read and write signals are synchronized to the **rdclk** and **wrclk** clocks respectively. The input and output ports of the DCFIFO is illustrated below.

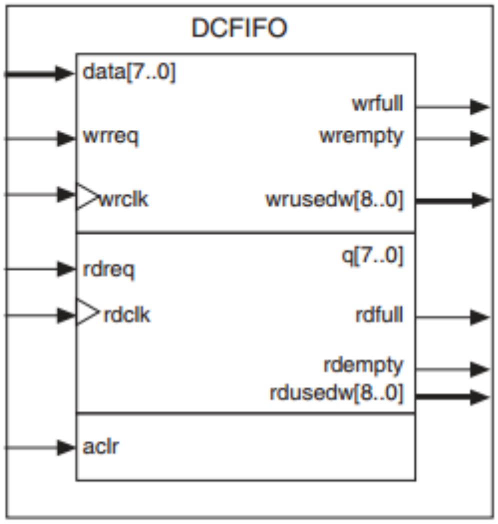

Figure 7-3 DCFIFO Structure

2. Interface

Port	Direction	Width	Description
wrclk	Input	1	Positive-edge-triggered clock. Use to synchronize the following ports: ■ data ■ wrreq ■ wrfull ■ wrusedw
rdclk	Input	1	Positive-edge-triggered clock. Use to synchronize the following ports: ■ q ■ rdreq ■ rdempty ■ rdusedw
data	Input	lpm_width	Holds the data to be written in the FIFO when the **wrreq** signal is asserted. If you manually instantiate the FIFO, ensure the port width is equal to the lpm_width parameter.

wrreq	Input	1	Assert this signal to request for a write operation.
rdreq	Input	1	Assert this signal to request for a read operation
aclr	Input	1	Assert this signal to clear all the output status ports. Asynchronous.
q	Output	lpm_width	Shows the data read from the read request operation. the width of the **q** port must be equal to the width of the data port.
wrfull	Output	1	When asserted, the FIFO is considered full. Do not perform write request operation when the FIFO is full.
rdempty	Output	1	When asserted, the FIFO is considered empty. Do not perform read request operation when the FIFO is empty
wrusedw	Output	lpm_widthu	Show the number of words stored in the FIFO. Ensure that the port width is equal to the lpm_widthu parameter if you manually instantiate the FIFO
rdusedw	Output	lpm_widthu	Show the number of words stored in the FIFO. Ensure that the port width is equal to the lpm_widthu parameter if you manually instantiate the FIFO

* Some other ports of the DCFIFO IP Core (wrempty, rdfull, almost_empty, almost_full, etc.) are not used.

Table 7-2 DCFIFO Interface

3. Main Parameter

Parameter	Type	Description
lpm_width	Integer	Specifies the width of the data and q ports
lpm_widthu	Integer	Specifies the width of the **rdusedw** and **wrusedw** ports
lpm_numwords	Integer	Specifies the depths of the FIFO you require. The value must be at least 4. The value assigned must comply with this equation, 2^{LPM_WIDTHU}

Table 7-3 DCFIFO Parameter

Attention: Default values of other parameter are commended to use. For more parameter, please refer to http://www.altera.com/literature/ug/ug_fifo.pdf, page 10.

4. Timing example

The figure below shows an example of the timing of DCFIFO. The write clock and read clock are of different frequency. This example is simulated by myself using Modelsim, which can help you get a better understand of DCFIFO. For more timing examples and explanation about DCFIFO, please refer to http://www.altera.com/literature/ug/ug_fifo.pdf

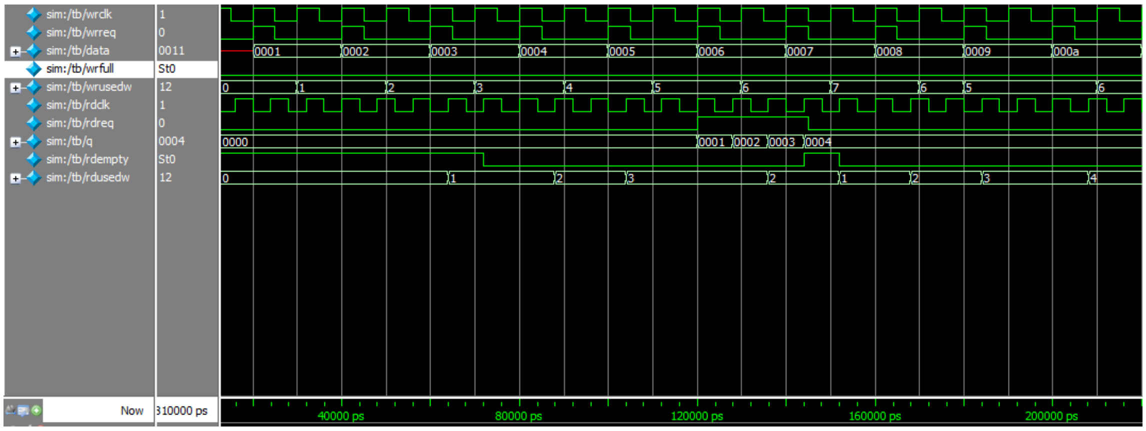

Figure 7-4 DCFIFO Timing Example

7.3.2 FIFOs Read/Write Control Logic

This module provides an interface for FIFOs and SDRAM to exchange data. Its main functions include:

1. Converting the state of FIFOs into read/write request signals for COMMAND_INTERFACE sub module. The read/write request to SDRAM is generated automatically.
2. Generate read/write address to SDRAM.
3. Switch among the four FIFOs deciding which one should be read or write.
4. Control the read/write operations of FIFO according to SDRAM timing sequence.

Signals related to the FIFOs read/write control logic are demonstrated below. Table 7-4 shows functions of some signals in the figure below.

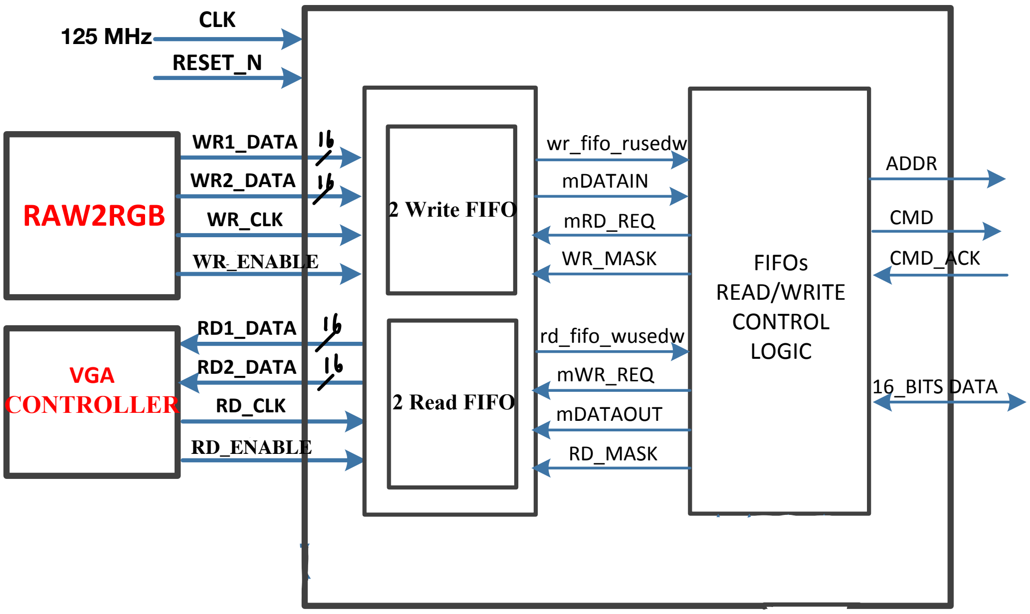

Figure 7-5 Signals Related to FIFO Read/Write Control Logic

1. FIFO interfaces

The input/output signals of the 4 FIFOs in SDRAM_CONTROL_4PORT module is demonstrated below.

	Write FIFO 1	Write FIFO 2	Read FIFO 1	Read FIFO 2
Wrclk	WR_CLK (from **RAW2RGB**)	WR_CLK (from **RAW2RGB**)	RD_CLK (from **RAW2RGB**)	RD_CLK (from **RAW2RGB**)
Rdclk	CLK (module clock)	CLK (module clock)	CLK (module clock)	CLK (module clock)
Data (input data)	WR1_DATA (from **RAW2RGB**)	WR2_DATA (from **RAW2RGB**)	mDATAOUT	mDATAOUT
Wrreq	WRITE (from **RAW2RGB**)	WRITE (from **RAW2RGB**)	RD_MASK[0]& mWR_REQ (Generated by FIFO CONTROL LOGIC)	RD_MASK[1]& mWR_REQ (Generated by FIFO CONTROL LOGIC)

Rdreq	WR_MASK[0]& mRD_REQ (Generated by FIFO CONTROL LOGIC)	WR_MASK[1]& mRD_REQ (Generated by FIFO CONTROL LOGIC)	READ (from **VGA CONTROLLER**)	READ (from **VGA CONTROLLER**)
Q (output data)	mDATAIN1*	mDATAIN2*	RD1_DATA (to **VGA CONTROLLER**)	RD2_DATA (to **VGA CONTROLLER**)

* mDATAIN is the input data of SDRAM memory.

If WR_MASK[0] is 1, mDATAIN = mDATAIN1; else if WR_MASK[1] is 1, mDATAIN = mDATAIN2

Table 7-4 Input and Output Signals of 4 FIFOs

2. Read/Write Request Generation

The RAW2RGB module, which can be considered as data source, keeps writing data to the two write FIFOs. The VGA CONTROLLER module as the "data consumer", keeps reading data to two read FIFOs. The states of the two write FIFOs are reflected by the **wr_fifo_rusedw1** and **wr_fifo_rusedw2** signals, which represents how much data in write FIFO1/2 can be read out to the SDRAM. The states of the two read FIFOs are reflected by the **rd_fifo_wusedw1** and **rd_fifo_wusedw2** signals, which represents how much data in read FIFO1/2 can be written to the VGA CONTROLLER.

When the amount of data in write FIFO1 reaches **rWR1_LENGTH**, which is a parameter configured by user, a read request **mRD_REQ** will be automatically generated to read data from Write FIFO 1 to the SDRAM, and WR_MASK[0] will be set to 1. The amount of data read for one burst is **rWR1_LENGTH** (16-bit data is transferred for one clock cycle in a burst). And also when data is read out from write FIFO 1, RAW2RGB module still keeps writing data to it.

Write FIFO 1 has the highest priority to exchange data with SDRAM. If the amount of data in write FIFO 1 is not enough for one burst and SDRAM data line is not occupied at the same time, write FIFO 2 will be considered. If the amount of data in write FIFO2 reaches **rWR2_LENGTH**, read request **mRD_REQ** will be generated to read data from Write FIFO 2 to the SDRAM, and WR_MASK[1] will be set to 1.

If both write FIFO 1 and 2 do not have enough data to be read out and SDRAM is not busy, read FIFO 1 will be considered. When the amount of data in read FIFO1 is less than **rRD1_LENGTH**, a write request **mWR_REQ** will be automatically generated to write data from SDRAM to the write FIFO1, and RD_MASK[0] will be set to 1. And also when data is written from SDRAM to read FIFO 1, VGA CONTROLLER module still keeps reading data to it.

Read FIFO 2 has the lowest priority to communicate with SDRAM. The circumstance is similar to what are mentioned above.

The code about this part is shown below.

```
// Write Side 1
if( (write_side_fifo_rusedw1 >= rWR1_LENGTH) && (rWR1_LENGTH!=0) )
begin
    mADDR   <= rWR1_ADDR;
    mLENGTH <= rWR1_LENGTH;
    WR_MASK <= 2'b01;
    RD_MASK <= 2'b00;
    mWR     <= 1;
    mRD     <= 0;
end
// Write Side 2
else if( (write_side_fifo_rusedw2 >= rWR2_LENGTH) && (rWR2_LENGTH!=0) )
begin
    mADDR   <= rWR2_ADDR;
    mLENGTH <= rWR2_LENGTH;
    WR_MASK <= 2'b10;
    RD_MASK <= 2'b00;
    mWR     <= 1;
    mRD     <= 0;
end
// Read Side 1
else if( (read_side_fifo_wusedw1 < rRD1_LENGTH) )
begin
    mADDR   <= rRD1_ADDR;
    mLENGTH <= rRD1_LENGTH;
    WR_MASK <= 2'b00;
    RD_MASK <= 2'b01;
    mWR     <= 0;
    mRD     <= 1;
end
// Read Side 2
else if( (read_side_fifo_wusedw2 < rRD2_LENGTH) )
begin
    mADDR   <= rRD2_ADDR;
    mLENGTH <= rRD2_LENGTH;
    WR_MASK <= 2'b00;
    RD_MASK <= 2'b10;
    mWR     <= 0;
```

Figure 7-6 Code Related to Read/Write Request Generation

3. Address Generation

The FIFOs read/write logic also generates the read/write address **mADDR** to the SDRAM.

The value of **mADDR** is assigned when a read/write request to the SDRAM is generated. If SDRAM reads data from write FIFO 1, **rWR1_ADDR**, which is the register write address of write FIFO 1, will be assigned to **mADDR**. Likewise, if SDRAM reads data from write FIFO 2, **rWR2_ADDR** will be assigned to **mADDR**. If SDRAM writes data to read FIFO 1 or 2, **rRD1_ADDR** or **rRD2_ADDR** will be assigned to **mADDR.**

The value of **rWR1_ADDR** is generated in the following way: when user input signal **WR1_LOAD** is set to 1 (i.e. the FIFO is reset), user input address **WR1_ADDR** will be assigned to **rWR1_ADDR**. Else when write FIFO 1 writes data to SDRAM, after a write transaction is finished, **rWR1_ADDR** will adds **rWR1_LENGTH** to its original value. If the current address is less than **rWR1_MAX_ADDR – rWR1_LENGTH**, the address will be reassigned to **WR1_ADDR**. Here **rWR1_MAX_ADDR** is the max address offset to **WR1_ADDR**.

The condition of **WR2_ADDR**, **RD1_ADDR**, **RD2_ADDR** is similar. The code about this part is shown below.

```
if(WR1_LOAD)
begin
    rWR1_ADDR   <=  WR1_ADDR;
    rWR1_LENGTH <=  WR1_LENGTH;
end
else if(mWR_DONE&WR_MASK[0])
begin
    if(rWR1_ADDR<rWR1_MAX_ADDR-rWR1_LENGTH)
        rWR1_ADDR   <=  rWR1_ADDR+rWR1_LENGTH;
    else
        rWR1_ADDR   <=  WR1_ADDR;
end
```

Figure 7-7 Code Related to Address Generation

Notice that **mADDR** is multiplex. When the FIFOs are reset, user-defined address (**WR1_ADDR**, **WR2_ADDR**, **RD1_ADDR** and **RD2_ADDR**) can be assigned to **mADDR**. During the normal read/write transaction, **mADDR** includes the column address, row address and bank address to SDRAM.

Attention:

In this design, Write FIFO 1 and read FIFO 1 use the same part of SDRAM since they have the same start address(**WR1_ADDR** = **RD1_ADDR** = 0) and maximum offset (**rWR1_MAX_ADDR = rRD1_MAX_ADDR** = 640*480). The maximum offset is 640*480, which is the resolution of one input image (please refer to RAW2RGB module for detail). So only the data amount of one image can be stored in SDRAM.

We know the actual storage space of one bank of the SDRAM is 8M, which is larger than 640*480. Why no more images are stored in the SDRAM? The reason is that the read frequency and write frequency of SDRAM are different, which is because the data source (RAW2RGB module) and data consumer (VGA CONTROLLER module) are in different clock domain. Thus the change of **rWR1_ADDR** and **rRD1_ADDR** are not simultaneous. If the read frequency of SDRAM is quicker than the write frequency, **rRD1_ADDR** will grow quicker than **WR1_ADDR**. Supposing infinite space is used in SDRAM to store data, which means the difference between **rRD1_ADDR** and **rWR1_ADDR** are getting larger. So the image data written to SDRAM and the image data read from SDRAM will not be simultaneous, and the time delay between input image and output image will be larger and larger.

So we set the maximum offset is 640*480 to store only one image in SDRAM. Since **rRD1_ADDR** and **rWR1_ADDR** are both circular, the difference between them will be no more than 640*480, which means the delay between input image and output image will be no more than the time of capturing one frame for camera. In this way, the output image can be regarded as "real-time". However, if we store the data amount of ten images in SDRAM, the maximum time delay between input and output will be the time of capturing ten frames for camera, which is longer.

4. Full-Page Burst Mode and timing sequence

When large amount or data is read or written, Full-Page Burst Mode is a good way to take full use of SDRAM performance and enhance read/write speed.

The full-page burst is used in conjunction with the BURST TERMINATE command to generate arbitrary burst lengths, which can be defined by user according to the actual application. The data in a row can be read or written consecutively in full-page mode. The address doesn't need to be configured to every data.

Read timing sequence

Before any READ or WRITE commands can be issued to a bank within the SDRAM, a row in that bank must be opened. This is accomplished via the ACTIVE command, which selects both the bank and the row to be activated. After opening a row (issuing an ACTIVE command), a READ or WRITE command may be issued to that row after several clock cycles determined by t_{RCD}.

The starting column and bank addresses are provided with the READ command. During READ bursts, the valid data-out element from the starting column address will be available following the CAS latency after the READ command. Each subsequent data-out element will be valid by the next positive clock edge.

Upon completion of a burst, assuming no other commands have been initiated, the DQs will go High-Z. A full-page burst will continue until terminated.

Full-page READ bursts can be truncated with the BURST TERMINATE command or PRECHARGE command to the same bank, and The BURST TERMINATE command or PRECHARGE command should be issued x cycles before the clock edge at which the last desired data element is valid, where x equals the CAS latency minus one. The diagram below shows Full-Page burst read terminated by BURST TERMINATE command for CAS latency=2. For more detail, please refer to IS42R16320D.pdf, Page 37

READ - FULL-PAGE BURST

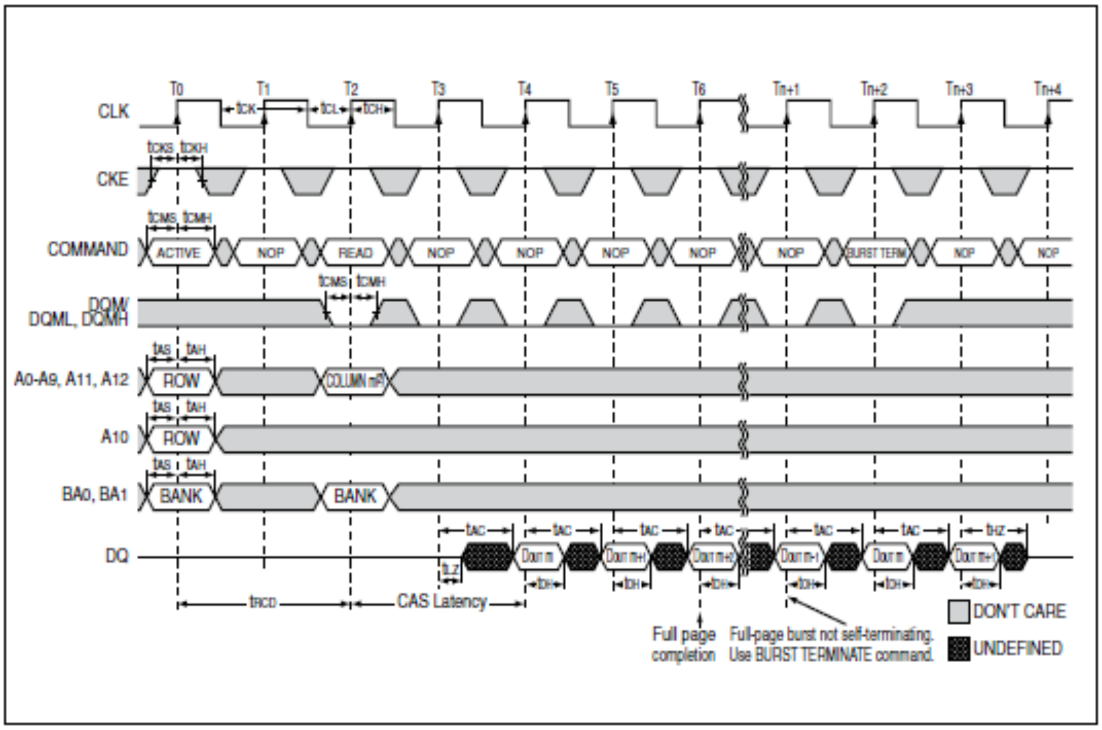

Figure 7-8 Read Full-Page Burst

The code about this part is shown below. ST is the controller status which records the timing of SDRAM. When the FIFOs control logic send a write/read command **CMD** to COMMAND_INTERFACE module and get acknowledge signal **CMDACK**, the read/write transaction to SDRAM begins.

```
case(ST)
0:  begin
        if({Pre_RD,mRD}==2'b01)
        begin
            Read    <=  1;
            Write   <=  0;
            CMD     <=  2'b01;
            ST      <=  1;
        end
        else if({Pre_WR,mWR}==2'b01)
        begin
            Read    <=  0;
            Write   <=  1;
            CMD     <=  2'b10;
            ST      <=  1;
        end
    end
1:  begin
        if(CMDACK==1)
        begin
            CMD<=2'b00;
            ST<=2;
        end
    end
default:
    begin
        if(ST!=SC_CL+SC_RCD+mLENGTH+1)
        ST<=ST+1;
        else
        ST<=0;
    end
endcase
```

Figure 7-9 Code Related to Read Timing

In a read transaction, after (t_{RCD} + t_{SC_CL}.) clock cycles the READ command is executed, input data from SDRAM is valid (tSC_CL = CAS latency) according to timing sequence in last page. After another **mLENGTH** clock cycles, which is the user-defined read length, the input data is invalid. The related code is shown below.

```
if(Read)
begin
    if(ST==SC_CL+SC_RCD+1)
    OUT_VALID   <=  1;
    else if(ST==SC_CL+SC_RCD+mLENGTH+1)
    begin
        OUT_VALID   <=  0;
        Read        <=  0;
        mRD_DONE    <=  1;
    end
end
else
mRD_DONE    <=  0;
```

Figure 7-10 Code Related to Read Timing Sequence

Write timing sequence

An ACTIVE command must be issued before the WRITE command, which is similar to the read timing sequence. The WRITE command may be issued after several clock cycles determined by tRCD.

The starting column and bank addresses are provided with the WRITE command. During WRITE bursts, the first valid data-in element will be registered coincident with the WRITE command. Subsequent data elements will be registered on each successive positive clock edge. Upon completion of a fixed-length burst, assuming no other commands have been initiated, the DQs will remain High-Z and any additional input data will be ignored. A full-page burst will continue until terminated.

Full-page WRITE bursts can be truncated with the BURST TERMINATE command or PRECHARGE command to the same bank. When truncating a WRITE burst with the BURST RERMINATE, the input data applied coincident with the BURST TERMINATE command will be ignored. The last data written (provided that DQM is LOW at that time) will be the input data applied one clock previous to the BURST TERMINATE command. The diagram below shows Full-Page burst write terminated by BURST TERMINATE command. For more detail, please refer to **IS42R16320D.pdf, Page 44**

WRITE - FULL PAGE BURST

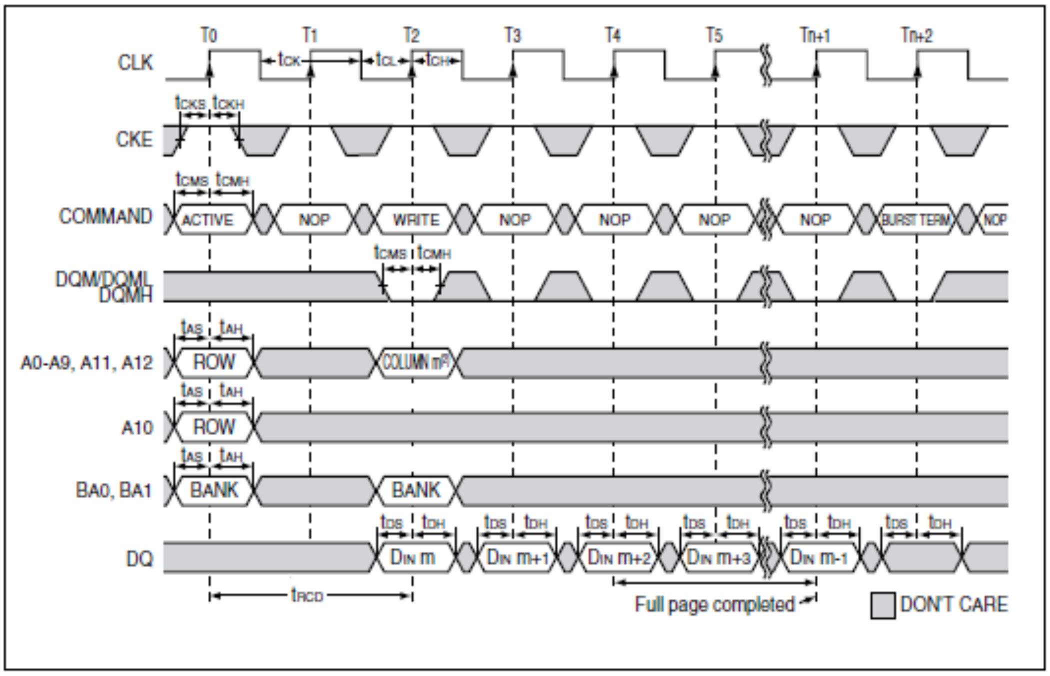

Figure 7-11 Write Full-Page Burst

The code about this part is shown below. In a write transaction, after tRCD clock cycles after the WRITE command is executed, output data to SDRAM is valid according to timing sequence in last page. After another **mLENGTH** clock cycles, which is the user-defined read length, the output data is invalid. The related code is shown below.

```
    if(Write)
    begin
        if(ST==SC_CL-1)
        IN_REQ  <=  1;
        else if(ST==SC_CL+mLENGTH-1)
        IN_REQ  <=  0;
        else if(ST==SC_CL+SC_RCD+mLENGTH)
        begin
            Write   <=  0;
            mWR_DONE<=  1;
        end
    end
    else
    mWR_DONE<=  0;
```

Figure 7-12 Code Related to Write Timing

7.3.3 COMMAND_INTERFACE Sub Module

1. Description

This module judges the input CMD signal from FIFOs CONTROL LOGIC and output the corresponding decoded command to COMMAND module. There are also an initial timer and a SDRAM refresh timer embedded in the module. When the SDRAM is powered up and initialized, a predefined manner must be complied. In this case, the decoded command to COMMAND module is generated according to the value of initial timer. Also Because the SDRAM needs to be refreshed automatically when it works, a REFRESH decoded command is generated at regular interval according to the refresh timer.

2. Interface

Port	Direction	Width	Description
CLK	Input	1	Module clock
RESET_N	Input	1	Module reset active low, asynchronous
CMD	Input	3	Command input
ADDR	Input	`ASIZE	Address
REF_ACK	Input	1	Refresh request acknowledge
CM_ACK	Input	1	Command acknowledge(from COMMAND module)
NOP	Output	1	Decoded NOP command
READA	Output	1	Decoded READA command
WRITEA	Output	1	Decoded WRITEA command

REFRESH	Output	1	Decoded REFRESH command
PRECHARGE	Output	1	Decoded PRECHARGE command
LOAD_MODE	Output	1	Decoded LOAD_MODE command
SADDR	Output	`ASIZE	Registered version of **ADDR**
REF_REQ	Output	1	Refresh request
INIT_REQ	Output	1	Initial request
CMD_REQ	Output	1	Command acknowledge (to FIFOs CONTROL LOGIC)

Table 7-5 CONTROL_INTERFACE sub module Interface

3. Function

Generate WRITEA and READA command

This module generates **WRITEA** and **READA** command to the COMMAND module according to the input **CMD** signal from FIFOs CONTROL LOGIC.

If **CMD** is 3'b001, **READA** command is generated.
If **CMD** is 3'b010, **WRITEA** command is generated.
If **CMD** is 3'b000, **NOP** command is generated.

Generate refresh command

Since the storage unit of SDRAM is actually capacitor which tends to discharge, SDRAM must be auto refreshed at regular interval to maintain the data stored. According to the datasheet of SDRAM IS42R16320D, The SDRAM must be refreshed 8k times in 64ms. Because the clock frequency of the module is 125MHz, therefore 64ms means 8M clock cycles. Thus the interval between two auto refresh behaviors is 8M/8k = 1k clock cycles.

The code in this module about auto refresh is shown below. The parameter REF_PER is defined in the file **SDRAM_PARAM.h**. Its value is 1024 for the reason mentioned above. A timer is used to calculate the passing clock cycles since last refresh. If the value of timer is 0, a refresh request is generated. And when a refresh acknowledge is received, the value of timer will be reset to REF_PER.

```
// refresh timer
always @(posedge CLK or negedge RESET_N) begin
        if (RESET_N == 0)
        begin
                timer           <= 0;
                REF_REQ         <= 0;
        end
        else
        begin
                if (REF_ACK == 1)
                begin
                    timer <= REF_PER;
                    REF_REQ <=0;
                end
                else if (INIT_REQ == 1)
                begin
                    timer <= REF_PER+200;
                    REF_REQ <=0;
                end
                else
                    timer <= timer - 1'b1;

                if (timer==0)
                    REF_REQ     <= 1;

        end
end
```

Figure 7-13 Code Related to Refresh Request Generation

Generate Initial manner

When the SDRAM is powered up and initialized, a predefined manner must be complied. The values of REFRESH command, PRECHARGE command and LOAD_MODE command are decided according to the initialized timing sequence, which is illustrated below. For more details, please refer to [IS42R16320D.pdf, Page 22-24]

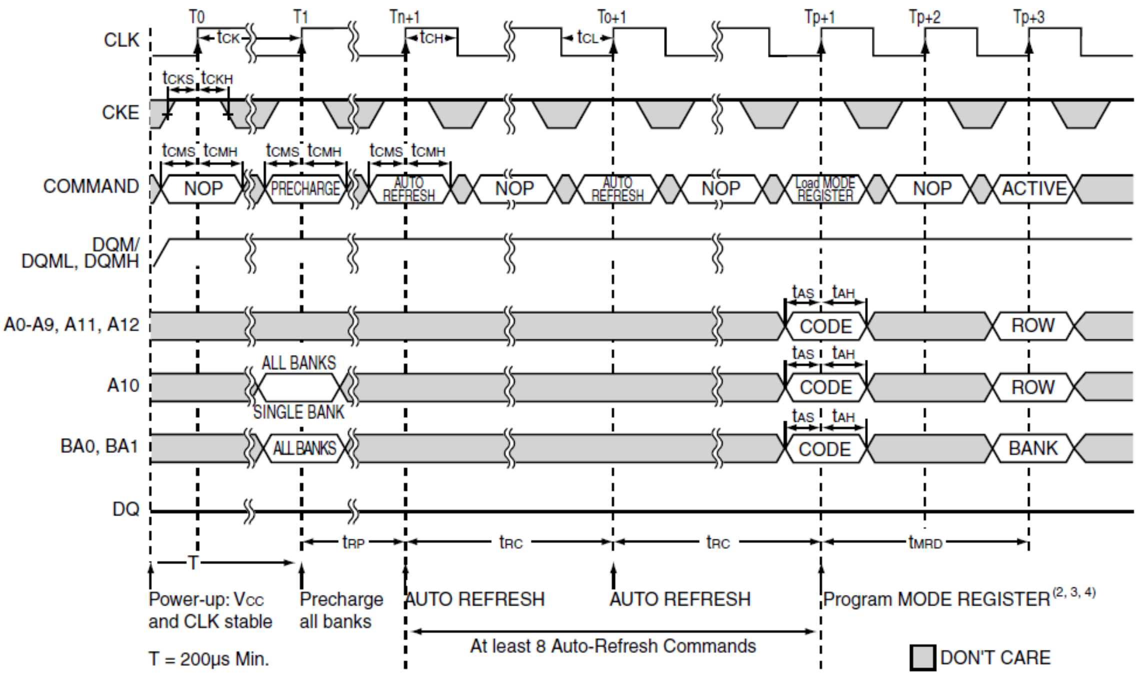

Figure 7-14 Initialization Timing Sequence

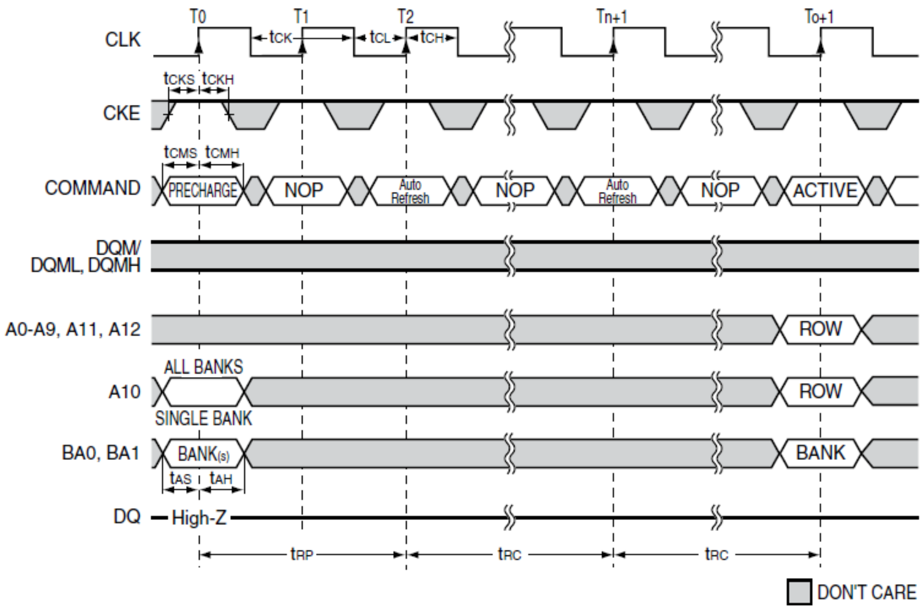

Figure 7-15 Auto Refresh Cycles in Initializaiton

A 100us delay is required prior to issuing any command other than a COMMAND INHIBIT or a NOP. The COMMAND INHIBIT or NOP may be applied during the 100us period and should continue at least through the end of the period. With at least one COMMAND INHIBIT or NOP command having been applied, a PRECHARGE command should be applied once the 100us delay has been satisfied. All banks must be precharged. This will leave all banks in an idle state after which at least eight AUTO REFRESH cycles must be performed. After the AUTO REFRESH cycles are complete, the SDRAM is then ready for mode register programming. The mode register should be loaded prior to applying any operational command because it will power up in an unknown state.

The code about the generation of decoded command during the initialization is shown below.

```
always @(posedge CLK or negedge RESET_N) begin
        if (RESET_N == 0)
        begin
                init_timer      <= 0;
                REFRESH         <= 0;
                PRECHARGE       <= 0;
                LOAD_MODE       <= 0;
                INIT_REQ        <= 0;
        end
        else
        begin
                if (init_timer < (INIT_PER+201))
                    init_timer  <= init_timer+1;

                if (init_timer < INIT_PER)
                begin
                    REFRESH     <=0;
                    PRECHARGE   <=0;
                    LOAD_MODE   <=0;
                    INIT_REQ    <=1;
                end
                else if(init_timer == (INIT_PER+20))
                begin
                    REFRESH     <=0;
                    PRECHARGE   <=1;
                    LOAD_MODE   <=0;
                    INIT_REQ    <=0;
                end
```

PRECHARGE COMMAND

8 AUTOREFRESH COMMAND

LOAD_MODE COMMAND

```
                else if(    (init_timer == (INIT_PER+40))   ||
                            (init_timer == (INIT_PER+60))   ||
                            (init_timer == (INIT_PER+80))   ||
                            (init_timer == (INIT_PER+100))  ||
                            (init_timer == (INIT_PER+120))  ||
                            (init_timer == (INIT_PER+140))  ||
                            (init_timer == (INIT_PER+160))  ||
                            (init_timer == (INIT_PER+180))  )
                begin
                    REFRESH     <=1;
                    PRECHARGE   <=0;
                    LOAD_MODE   <=0;
                    INIT_REQ    <=0;
                end
                else if(init_timer == (INIT_PER+200))
                begin
                    REFRESH     <=0;
                    PRECHARGE   <=0;
                    LOAD_MODE   <=1;
                    INIT_REQ    <=0;
                end
                else
                begin
                    REFRESH     <=0;
                    PRECHARGE   <=0;
                    LOAD_MODE   <=0;
                    INIT_REQ    <=0;
                end
        end
end
```

Figure 7-16 Code Related to Initial Manner Generation

Attention:

1. The clock cycles between each auto refresh is 20 cycles, which is equal to t_{RC} of the SDRAM.
2. The parameter INIT_PER in the code is configured in file **SDRAM_PARAM.h**. The value of **INIT_PER** is 25000. Since the module clock frequency of SDRAM_CONTROL_4PORT module is 125MHz, thus 25000 clock cycles is 200us, which is the minimum time required before the first precharge command is generated.

7.3.4 COMMAND sub module

1. Description

This module uses decoded command signals from CONTROL_INTERFACE sub module to generate operations conforming to SDRAM protocol and control the signals output to SDRAM. This module also generates **OE** signal for data path module to control it. Besides, non-multiplex input address ADDR is transferred into multiplex address for SDRAM and sent to **SA** and **BA** according to the time.

Attention: the code of this module is not suggested to be modified. You do not need to understand the internal logic of this module, and the following part can be omitted.

2. Interface

Port	Direction	Width	Description
CLK	Input	1	Module clock
RESET_N	Input	1	Module reset active low, asynchronous

SADDR	Input	`ASIZE	Multiplex Address
NOP	Input	1	Decoded NOP command
READA	Input	1	Decoded READA command
WRITEA	Input	1	Decoded WRITEA command
REFRESH	Input	1	Decoded REFRESH command
PRECHARGE	Input	1	Decoded PRECHARGE command
LOAD_MODE	Input	1	Decoded LOAD_MODE command
REF_REQ	Input	1	Refresh request
INIT_REQ	Input	1	Initial request
PM_STOP	Input	1	Page mode stop
REF_ACK	Output	1	Refresh request acknowledge
CMD_ACK	Output	1	Command acknowledge
OE	Output	1	OE signal for DATA_PATH module
SA	Output	12	SDRAM address
BA	Output	2	SDRAM bank address
CS_N	Output	2	SDRAM chip selects Active low
CKE	Output	1	SDRAM clock enable
RAS_N	Output	1	SDRAM Row Address Strobe Command
CAS_N	Output	1	SDRAM Column Address Strobe Command
WE_N	Output	1	SDRAM write enable

Table 7-6 COMMAND sub module Interface

3. Function

Generate operations to SDRAM

Operations are generated according to the input decoded command signals. The logic is generated below. Notice that REFRESH command has higher priority than READA/WRITEA command.

```
if ((REF_REQ == 1 | REFRESH == 1) & command_done == 0 & do_refresh == 0 & rp_done == 0          // Refresh
        & do_reada == 0 & do_writea == 0)
        do_refresh <= 1;
else
        do_refresh <= 0;

if ((READA == 1) & (command_done == 0) & (do_reada == 0) & (rp_done == 0) & (REF_REQ == 0))    // READA
begin
        do_reada <= 1;
        ex_read <= 1;
end
else
        do_reada <= 0;

if ((WRITEA == 1) & (command_done == 0) & (do_writea == 0) & (rp_done == 0) & (REF_REQ == 0))  // WRITEA
begin
        do_writea <= 1;
        ex_write <= 1;
end
else
        do_writea <= 0;

if ((PRECHARGE == 1) & (command_done == 0) & (do_precharge == 0))   // PRECHARGE
        do_precharge <= 1;
else
        do_precharge <= 0;

if ((LOAD_MODE == 1) & (command_done == 0) & (do_load_mode == 0))   // LOADMODE
        do_load_mode <= 1;
else
        do_load_mode <= 0;
```

Figure 7-17 Code Related to Operation Generation

What needs to be noticed is that WRITEA and READA command actually imply an ACTIVE command first, which is demonstrated in the state diagram below. Thus when the module receives WRITEA or READA command, which means the interval signal do_wirtea or do_reada is 1, ACTIVE command must be executed first, and then write or read operation are enabled after a time delay according to CAS configuration of SDRAM.

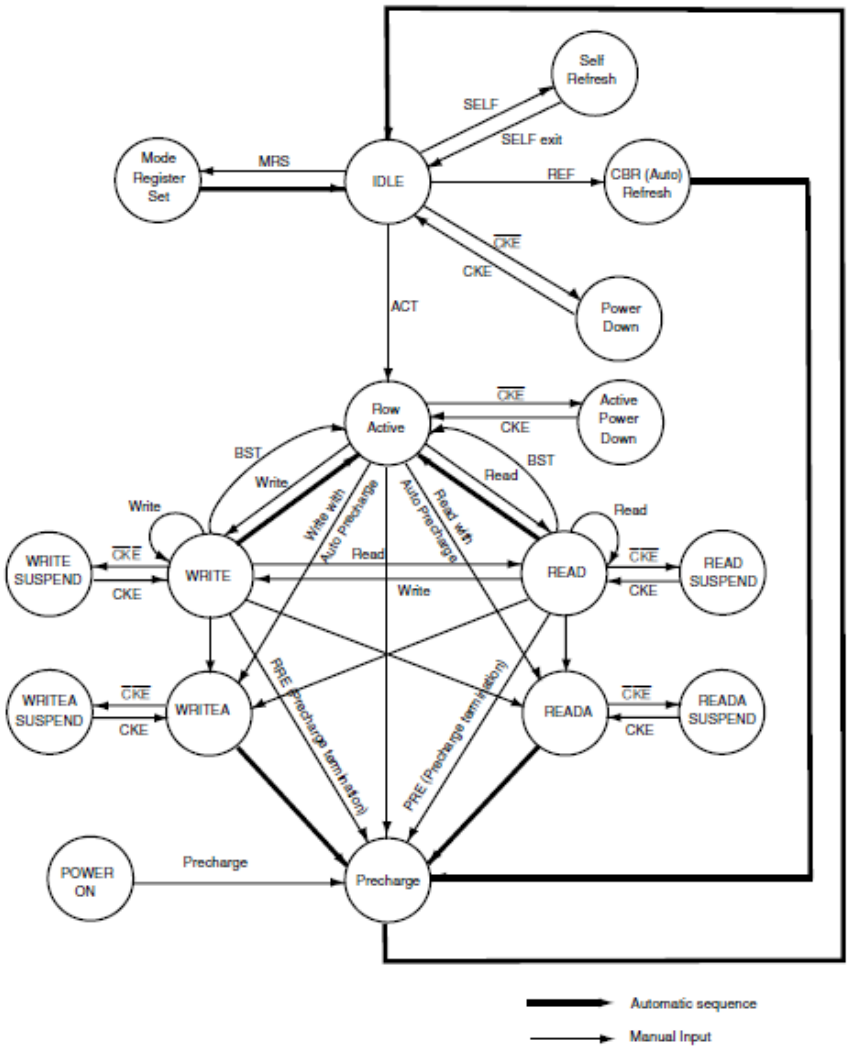

Figure 7-18 SDRAM State Diagram

The code below tracks the time between the ACTIVE command and the subsequent WRITE or READ command. The shift register is set using the configuration register setting SC_RCD. The shift register is loaded with a single '1' with the position within the register dependent on SC_RCD. When the '1' is shifted out of the register it sets so_rw which triggers a write or reada command.

```
always @(posedge CLK or negedge RESET_N)
begin
        if (RESET_N == 0)
        begin
                rw_shift <= 0;
                do_rw    <= 0;
        end

        else
        begin

                if ((do_reada == 1) | (do_writea == 1))
                begin
                        if (SC_RCD == 1)                          // Set the shift register
                                do_rw <= 1;
                        else if (SC_RCD == 2)
                                rw_shift <= 1;
                        else if (SC_RCD == 3)
                                rw_shift <= 2;
                end
                else
                begin
                        rw_shift <= (rw_shift>>1);
                        do_rw    <= rw_shift[0];
                end
        end
end
```


```
always @(posedge CLK or negedge RESET_N)
begin
        if (RESET_N == 0)
        begin
                rw_shift <= 0;
                do_rw    <= 0;
        end

        else
        begin

                if ((do_reada == 1) | (do_writea == 1))
                begin
                        if (SC_RCD == 1)                          // Set the shift register
                                do_rw <= 1;
                        else if (SC_RCD == 2)
                                rw_shift <= 1;
                        else if (SC_RCD == 3)
                                rw_shift <= 2;
                end
                else
                begin
                        rw_shift <= (rw_shift>>1);
                        do_rw    <= rw_shift[0];
                end
        end
end
```


Figure 7-19 Code Related to Timing Tracking After ACTIVE Command

OE signal

OE is a control signal of DATA_PATH module. OE is 1 when write operation is executed, and OE is 0 when read operation is executed. For normal burst write (SC_PM=0) the duration of OE is dependent on the configured burst length. For page mode accesses (SC_PM=1) the OE signal is turned on at the start of the write command and is left on until a PRECHARGE (page burst terminate) is detected.

Address

Non-multiplex address **SADDR** is transferred into multiplex address for SDRAM and sent to **SA** and **BA** according to the time. The way how ADDR is divided into bank address, row address and column address is shown in the code below:

```
// assignment of the row address bits from SADDR
assign    rowaddr    = SADDR[`ROWSTART + `ROWSIZE - 1: `ROWSTART];

  // assignment of the column address bits
assign    coladdr    = SADDR[`COLSTART + `COLSIZE - 1:`COLSTART];

 // assignment of the bank address bits
assign    bankaddr   = SADDR[`BANKSTART + `BANKSIZE - 1:`BANKSTART];
```

Figure 7-20 Code Related to Address Assignment

The parameter ROWSTART, ROWSIZE, COLSTART, COLSIZE, BANKSTART, BANKSIZE are defined in the file **Sdram_Param.h**.

Acknowledge signal

When a REFRESH request generated by the internal refresh timer circuit in the COTROL_INTERFACE module is received, this module will generate a refresh acknowledge signal REF_ACK to the CONTROL_INTERFACE module. When other kinds of command is received, this module will generate a command acknowledge signal CMD_ACK to the CONTROL_INTERFACE module. The code about this part is shown below.

```
always @(posedge CLK or negedge RESET_N)
begin

        if (RESET_N == 0)
        begin
                CM_ACK   <= 0;
                REF_ACK  <= 0;
        end

        else
        begin
                if (do_refresh == 1 & REF_REQ == 1)                    // Internal refresh timer refresh request
                        REF_ACK <= 1;
                else if ((do_refresh == 1) | (do_reada == 1) | (do_writea == 1) | (do_precharge == 1)   // externa  commands
                         | (do_load_mode))
                        CM_ACK <= 1;
                else
                begin
                        REF_ACK <= 0;
                        CM_ACK  <= 0;
                end
        end
end
```

Figure 7-21 Code Related to Acknowledge Signals Generation

RAS_N, CAS_N, and WE_N signal

These signals are generated according to the issued command and the table below. For more details, please refer to [IS42R16320D.pdf, Page 9]

COMMAND TRUTH TABLE

Function	CKE n – 1	n	CS	RAS	CAS	WE	BA1	BA0	A10	A12, A11 A9 - A0
Device deselect (DESL)	H	×	H	×	×	×	×	×	×	×
No operation (NOP)	H	×	L	H	H	H	×	×	×	×
Burst stop (BST)	H	×	L	H	H	L	×	×	×	×
Read	H	×	L	H	L	H	V	V	L	V
Read with auto precharge	H	×	L	H	L	H	V	V	H	V
Write	H	×	L	H	L	L	V	V	L	V
Write with auto precharge	H	×	L	H	L	L	V	V	H	V
Bank activate (ACT)	H	×	L	L	H	H	V	V	V	V
Precharge select bank (PRE)	H	×	L	L	H	L	V	V	L	×
Precharge all banks (PALL)	H	×	L	L	H	L	×	×	H	×
CBR Auto-Refresh (REF)	H	H	L	L	L	H	×	×	×	×
Self-Refresh (SELF)	H	L	L	L	L	H	×	×	×	×
Mode register set (MRS)	H	×	L	L	L	L	L	L	L	V

Note: H=V_{IH}, L=V_{IL} x= V_{IH} or V_{IL}, V = Valid Data.

Figure 7-22 Command Truth Table

The code of this part is shown below.

```
//Generate the appropriate logic levels on RAS_N, CAS_N, and WE_N
//depending on the issued command.
//
            if ( do_refresh==1 ) begin                          // Refresh: S=00, RAS=0, CAS=0, WE=1
                RAS_N <= 0;
                CAS_N <= 0;
                WE_N  <= 1;
            end
            else if ((do_precharge==1) & ((oe4 == 1) | (rw_flag == 1))) begin      // burst terminate if write is active
                RAS_N <= 1;
                CAS_N <= 1;
                WE_N  <= 0;
            end
            else if (do_precharge==1) begin                     // Precharge All: S=00, RAS=0, CAS=1, WE=0
                RAS_N <= 0;
                CAS_N <= 1;
                WE_N  <= 0;
            end
            else if (do_load_mode==1) begin                     // Mode Write: S=00, RAS=0, CAS=0, WE=0
                RAS_N <= 0;
                CAS_N <= 0;
                WE_N  <= 0;
            end

            else if (do_reada == 1 | do_writea == 1) begin  // Activate: S=01 or 10, RAS=0, CAS=1, WE=1
                RAS_N <= 0;
                CAS_N <= 1;
                WE_N  <= 1;
            end
            else if (do_rw == 1) begin                          // Read/Write: S=01 or 10, RAS=1, CAS=0, WE=0 or 1
                RAS_N <= 1;
                CAS_N <= 0;
                WE_N  <= rw_flag;
            end
            else if (do_initial ==1) begin
                RAS_N <= 1;
                CAS_N <= 1;
                WE_N  <= 1;
            end
            else begin                                          // No Operation: RAS=1, CAS=1, WE=1
                RAS_N <= 1;
                CAS_N <= 1;
                WE_N  <= 1;
            end
```

Figure 7-23 Code Related to RAS_N, CAS_N, WE_N Signals Assignment