**Verilog Code**

module phy\_prog #(

parameter DATA\_WIDTH = 32 // cannot change

)(

input clk,

input rst,

output reg o\_cmd\_ready,

input i\_cmd\_valid,

input [15 : 0] i\_cmd,

input [15 : 0] i\_cmd\_id,

input [39 : 0] i\_addr,

input [63 : 0] i\_data,

input [31 : 0] i\_cmd\_param,

output o\_wready,

input i\_wvalid,

input [DATA\_WIDTH-1 : 0] i\_wdata,

input i\_wlast,

input i\_keep\_wait,

output reg [1:0] o\_status,

output reg o\_res\_valid,

output reg [15 : 0] o\_res\_id,

output reg o\_rd\_st\_req,

output reg [23 : 0] o\_rd\_st\_addr,

output reg [15 : 0] o\_rd\_st\_id,

output reg [ 2 : 0] o\_rd\_st\_type,

output reg io\_busy,

output reg o\_ce\_n,

input i\_rb\_n,

output reg o\_we\_n,

output reg o\_cle,

output reg o\_ale,

output [ 3 : 0] o\_re,

output o\_dqs\_tri\_en,

input [ 3 : 0] i\_dqs,

output reg [ 3 : 0] o\_dqs,

output reg o\_dq\_tri\_en,

output reg [31 : 0] o\_dq

);

This is the declaration of the phy\_prog module. It defines the module parameters and inputs/outputs:

* **parameter DATA\_WIDTH = 32**: A parameter defining the width of the data bus. It's set to 32 bits and cannot be changed.

**Inputs:**

1. **clk**: Clock signal, driving the logic synchronously.
2. **rst**: Reset signal, when high, resets the entire FSM and all registers.
3. **i\_cmd\_valid**: Indicates if a valid command is present to be processed.
4. **i\_cmd**: 16-bit input representing the command to be issued to the NAND flash. This could be specific operations like writing, erasing, etc.
5. **i\_cmd\_id**: 16-bit command identifier used to identify the specific command being processed.
6. **i\_addr**: 40-bit address to specify where the data will be written in the NAND memory (row/column address).
7. **i\_data**: 64-bit data input that holds the data to be written to the NAND flash.
8. **i\_cmd\_param**: 32-bit command parameter that contains additional information about the command:
   * **[0]**: Has second command cycle? (1 - yes, 0 - no)
   * **[3:1]**: Number of addresses.
   * **[15:4]**: Busy time (wait time before proceeding).
   * **[30:16]**: Number of bytes of data.
   * **[31]**: Data source (0 - use i\_wdata, 1 - use i\_data).
9. **i\_wvalid**: Indicates if the write data is valid for the current cycle.
10. **i\_wdata**: 32-bit data input for write data.
11. **i\_wlast**: Indicates the last piece of data in a burst write operation.
12. **i\_keep\_wait**: Keeps the controller in a WAIT state.
13. **i\_rb\_n**: Input signal that indicates the ready/busy status of the NAND chip (low when busy, high when ready).
14. **i\_dqs**: Data strobe signal (used in DDR NAND interfaces).

**Outputs:**

1. **o\_cmd\_ready**: Indicates that the controller is ready to accept the next command.
2. **o\_wready**: Indicates that the controller is ready to receive write data.
3. **o\_status**: 2-bit status register that shows the current state of the controller (IDLE, BUSY, WAIT, READY).
4. **o\_res\_valid**: Output that signals when a result is valid (used after the operation is done).
5. **o\_res\_id**: Output that holds the ID of the command that was just executed.
6. **o\_rd\_st\_req**: Read status request signal that indicates the controller is requesting a status read operation.
7. **o\_rd\_st\_addr**: Address for the read status operation.
8. **o\_rd\_st\_id**: ID of the read status request.
9. **o\_rd\_st\_type**: Type of the status read request.
10. **io\_busy**: Indicates whether the controller is busy (used for external monitoring).
11. **o\_ce\_n**: Output signal to control the NAND chip enable (0 when active, 1 when inactive).
12. **o\_we\_n**: Output signal to control the NAND write enable (0 for write).
13. **o\_cle**: Command latch enable signal (used to latch command cycles).
14. **o\_ale**: Address latch enable signal (used to latch addresses).
15. **o\_re**: Read enable signal (used to control the NAND read operations).
16. **o\_dqs\_tri\_en**: Data strobe output enable signal (used for tri-stating in DDR NAND).
17. **o\_dqs**: Data strobe output signal.
18. **o\_dq\_tri\_en**: Data bus tri-state enable signal (controls the direction of the data bus).
19. **o\_dq**: 32-bit data bus for sending/receiving data to/from NAND.

**Detailed Explanation of Signals**

**Command Handling**

* **i\_cmd\_valid**: When asserted, this signal indicates that a new command is available. The FSM reads the command (i\_cmd), address (i\_addr), and data (i\_data) when in the IDLE state.
* **o\_cmd\_ready**: The FSM asserts this signal when it is in the IDLE state, ready to accept a new command.

**Address Handling**

* **o\_ale**: This signal enables the Address Latch when the FSM is sending an address to the NAND flash.
* **i\_addr**: The address is provided as an input to the FSM and latched in the ADDRESS state.

**Data Transfer**

* **o\_wready**: The FSM asserts this signal during the DATA state to indicate that it is ready to accept external data for a write operation.
* **i\_wvalid**: This input indicates that the data on i\_wdata is valid. The FSM processes this data during the DATA state.
* **i\_wlast**: This input signal indicates the last data beat in a burst of data. When asserted, the FSM completes the data phase and transitions to the WAIT state.

**NAND Control Signals**

* **o\_ce\_n**: Chip Enable (active low), asserted when the FSM is communicating with the NAND flash.
* **o\_we\_n**: Write Enable (active low), asserted when the FSM is writing data to the NAND.
* **o\_cle**: Command Latch Enable, asserted during the COMMAND state to send a command to the NAND.
* **o\_ale**: Address Latch Enable, asserted during the ADDRESS state to send an address to the NAND.

**Local Parameters and Registers**

Verilog Code

localparam

IDLE = 10'b00\_0000\_0001,

CMD1 = 10'b00\_0000\_0010,

ADDR = 10'b00\_0000\_0100,

WPRE = 10'b00\_0000\_1000,

DATA = 10'b00\_0001\_0000,

WPST = 10'b00\_0010\_0000,

CMD2 = 10'b00\_0100\_0000,

BUSY = 10'b00\_1000\_0000,

LOCK = 10'b01\_0000\_0000,

WAIT = 10'b10\_0000\_0000;

localparam WARMUP\_DATA\_NUM = (PG\_WARMUP << 1);

These are **local parameters** that define the FSM states:

* **IDLE**: The idle state when the controller is waiting for a new command.
* **CMD1**: The first command cycle (the first byte of a command is latched).
* **ADDR**: The state where the address is latched into the NAND flash.
* **WPRE**: Prepares for data write.
* **DATA**: Data write state where data is transferred.
* **WPST**: Post-data write state for timing requirements.
* **CMD2**: The second command cycle if applicable.
* **BUSY**: Waits for the NAND flash to complete the operation.
* **LOCK**: Locks the state machine while waiting for the NAND ready/busy signal (i\_rb\_n).
* **WAIT**: Waits for the i\_rb\_n signal to go high.

WARMUP\_DATA\_NUM defines a warm-up value for the number of data cycles required before actual data transfer begins.

**Internal Registers and State Machine Logic**

Next, let’s discuss the internal state machine and the logic that makes the FSM work:

Verilog Code

reg [3:0] state, next\_state;

reg [15:0] cmd\_id;

reg [39:0] addr;

reg [63:0] data;

reg [31:0] cmd\_param;

reg [31:0] wdata;

reg wvalid, wlast;

reg [1:0] status;

reg res\_valid;

reg [15:0] res\_id;

reg rd\_st\_req;

reg [23:0] rd\_st\_addr;

reg [15:0] rd\_st\_id;

reg ce\_n, we\_n, cle, ale;

reg [31:0] dq;

reg dq\_tri\_en, dqs\_tri\_en;

**Internal Registers:**

* **state, next\_state [3:0]**: These represent the **current state** and the **next state** of the FSM. The FSM transitions from one state to another based on the inputs and the current state.
* **cmd\_id [15:0]**: Holds the **command ID** currently being processed. This ID comes from i\_cmd\_id when i\_cmd\_valid is asserted.
* **addr [39:0]**: Stores the **address** received from i\_addr. This is latched at the start of the operation when i\_cmd\_valid is high.
* **data [63:0]**: Holds the **data** to be written to the NAND flash. This comes from i\_data during a write operation.
* **cmd\_param [31:0]**: This stores additional **parameters** for the command, such as the number of addresses or data cycles, which are latched from i\_cmd\_param.
* **wdata [31:0]**: This is the **write data** received from i\_wdata, used during write operations.
* **wvalid, wlast**: These hold the status of the **write data** validity and the **last data beat** during a write transaction. These signals mirror i\_wvalid and i\_wlast.
* **status [1:0]**: This register holds the current **status** of the FSM (IDLE, BUSY, WAIT, or READY).
* **res\_valid**: This flag is set when the result of a command is ready to be returned to the external system.
* **res\_id [15:0]**: Holds the **ID** of the command whose result is ready. This ID is latched from i\_cmd\_id during command execution.
* **rd\_st\_req**: This signal is asserted when the FSM needs to **read the status** from the NAND flash after executing a command.
* **rd\_st\_addr [23:0]**: The **address** associated with the read status request. Typically, this is the page/block address where the command was applied.
* **rd\_st\_id [15:0]**: The ID associated with the **read status request**, which helps to track which operation's status is being queried.
* **ce\_n, we\_n, cle, ale**: These are the internal control signals for the NAND flash memory. They control **chip enable**, **write enable**, **command latch enable**, and **address latch enable**.
* **dq [31:0]**: The data register used to drive the **DQ bus** during write operations or command/address transfers.
* **dq\_tri\_en, dqs\_tri\_en**: These signals control the tri-state buffers for the **DQ** and **DQS** lines, determining whether they are driven by the FSM or in high impedance mode.

This Verilog code represents a finite state machine (FSM) that transitions between various states depending on inputs and current conditions. Below is a line-by-line detailed explanation of the code and its process:

#### **Sequential Logic Block (Triggered by Clock and Reset)**

Verilog Code

always @(posedge i\_clk or negedge i\_nrst) begin

if (!i\_nrst) begin

state <= IDLE;

end else begin

state <= next\_state;

end

end

* **always @(posedge i\_clk or negedge i\_nrst)**: This is a sequential block that gets triggered on the **rising edge** of the clock signal i\_clk or when the reset signal i\_nrst is **asserted low** (active-low reset).
  + **posedge i\_clk**: Indicates the block is executed on the rising edge of the clock, ensuring the FSM is synchronized to the system clock.
  + **negedge i\_nrst**: The FSM is also sensitive to the falling edge of i\_nrst, meaning when i\_nrst is low, the system will reset.
* **if (!i\_nrst)**: This checks if the reset signal i\_nrst is active (low).
  + **If reset is asserted**, the FSM sets the state to IDLE. This is the default or initial state the FSM enters after a reset to ensure the system starts from a known state.
* **else**: If i\_nrst is not low (i.e., the system is not in reset), the FSM transitions to the next\_state, which is determined by the combinational logic in the next block.

**Combinational Logic Block (Next State Logic)**

Verilog Code

always @(\*) begin

next\_state = state;

case (state)

IDLE: begin

if (i\_cmd\_valid) begin

next\_state = COMMAND;

end

end

* **always @(\*)**: This is a combinational logic block that updates next\_state based on the current state and the conditions inside the case statement.
  + The (\*) sensitivity list ensures that the block is triggered when any of its inputs change, making it purely combinational.
* **next\_state = state;**: This is a default assignment. The FSM assumes it will stay in the current state unless some specific condition changes (which is checked in the case statement).
* **case (state)**: This is a case statement that controls the behavior of the FSM based on its current state. The different states, like IDLE, COMMAND, ADDRESS, etc., represent different operational phases of the FSM.

**IDLE State**

Verilog Code

IDLE: begin

if (i\_cmd\_valid) begin

next\_state = COMMAND;

end

end

* **IDLE:**: This is the case where the FSM is in the IDLE state, which is the default state after reset or when the FSM is not processing any commands.
* **if (i\_cmd\_valid)**: This checks if the signal i\_cmd\_valid is high. The i\_cmd\_valid signal indicates that a new command is ready to be processed.
  + **If i\_cmd\_valid is asserted**, the FSM transitions to the COMMAND state. This signifies that the FSM has accepted the command and is ready to begin processing it.

**COMMAND State**

Verilog Code

COMMAND: begin

if (i\_keep\_wait) begin

next\_state = WAIT;

end else begin

next\_state = ADDRESS;

end

end

* **COMMAND:**: This state is where the FSM processes the incoming command.
* **if (i\_keep\_wait)**: This checks if the i\_keep\_wait signal is high. i\_keep\_wait is typically used to signal that the FSM needs to pause or wait before proceeding. This may be required for synchronization or other timing constraints.
  + **If i\_keep\_wait is high**, the FSM transitions to the WAIT state, which indicates it will wait until the NAND flash is ready to proceed (e.g., after sending a command or address).
* **else begin next\_state = ADDRESS;**: If i\_keep\_wait is not asserted, the FSM moves directly to the ADDRESS state, where it will begin sending the address for the command.

**ADDRESS State**

Verilog Code

ADDRESS: begin

if (/\* address complete \*/) begin

if (/\* write operation \*/) begin

next\_state = DATA;

end else begin

next\_state = WAIT;

end

end

end

* **ADDRESS:**: In this state, the FSM is responsible for sending the address associated with the current command to the NAND flash. The address is typically needed for operations like read, write, and erase.
* **if (/\* address complete \*/)**: This condition checks if the address has been fully sent to the NAND flash. The actual condition will depend on the address transmission mechanism and how many address cycles are required.
  + Once the address is sent, the FSM determines whether it needs to proceed with a write operation or wait.
* **if (/\* write operation \*/)**: This condition checks if the current command is a write operation (as opposed to a read or other command). The actual condition here would check a bit in the command signal.
  + **If it’s a write operation**, the FSM transitions to the DATA state to send the data.
  + **Else**, it transitions to the WAIT state, indicating that it needs to wait for the NAND flash to be ready.

**DATA State**

Verilog Code

DATA: begin

if (i\_wlast) begin

next\_state = WAIT;

end

end

* **DATA:**: This state is used when the FSM needs to send or receive data (typically for write or read operations). In this state, data is transmitted or received over a bus to/from the NAND flash.
* **if (i\_wlast)**: This condition checks if i\_wlast is high. The i\_wlast signal indicates that the last piece of data in the current transaction has been sent (in the case of a write) or received (in the case of a read).
  + **Once the last data has been transferred**, the FSM transitions to the WAIT state, indicating it is waiting for the NAND to finish the operation.

**WAIT State**

Verilog Code

WAIT: begin

if (i\_rb\_n) begin

next\_state = RESULT;

end

end

* **WAIT:**: In this state, the FSM is waiting for the NAND flash to be ready. This is common in NAND flash systems because many operations (like program, erase, or read) take time, and the FSM must wait for the NAND to signal it is ready to proceed.
* **if (i\_rb\_n)**: This condition checks the value of the i\_rb\_n signal. The i\_rb\_n signal (Ready/Busy) is typically used in NAND flash systems to indicate whether the flash memory is busy (low) or ready (high).
  + **If i\_rb\_n is high**, the FSM transitions to the RESULT state, meaning that the operation is complete, and the result can be processed.

**RESULT State**

Verilog Code

RESULT: begin

next\_state = IDLE;

end

* **RESULT:**: In this state, the FSM handles the result of the completed NAND operation. This may involve generating status signals, notifying the system that the operation is complete, and preparing for the next command.
* **next\_state = IDLE;**: Once the result has been processed, the FSM transitions back to the IDLE state, waiting for the next command to be issued.

**Default Case**

Verilog Code

default: begin

next\_state = IDLE;

end

* **default:**: This is a safeguard to ensure that if the state somehow enters an undefined or invalid state, the FSM transitions back to IDLE to recover and wait for the next valid command.

**Block 1: flag Register**

verilog

Copy code

always@(posedge clk or posedge rst)

if(rst) begin

flag <= 1'h0;

end else if((state == WPST) & has\_cmd2)begin

flag <= 1'h1;

end else if((state == CMD2) & is\_latch\_edge) begin

flag <= 1'h0;

end

* **always@(posedge clk or posedge rst)**: This block is triggered by either the positive edge of the clock (clk) or the positive edge of the reset (rst).
* **if (rst)**: When rst (reset) is high, the flag is set to 1'h0 (reset state).
* **else if((state == WPST) & has\_cmd2)**: If the current state is WPST (probably a Write Post state) and there is a second command (has\_cmd2 is true), the flag is set to 1'h1.
* **else if((state == CMD2) & is\_latch\_edge)**: If the current state is CMD2 (processing a second command) and is\_latch\_edge is true (probably indicating some timing edge has been reached), the flag is reset to 1'h0.

**Purpose:**

The flag signal is used to track the presence of a second command (has\_cmd2) and is reset when the second command processing (CMD2) is completed.

**Block 2: flag\_r Register**

verilog

Copy code

always@(posedge clk or posedge rst)

if(rst) begin

flag\_r <= 1'h0;

end else begin

flag\_r <= flag;

end

* **always@(posedge clk or posedge rst)**: Another sequential block triggered by the clock or reset.
* **if (rst)**: When reset is asserted, the flag\_r register is reset to 1'h0.
* **else**: On every clock cycle, flag\_r is updated with the value of flag.

**Purpose:**

flag\_r is a delayed version of flag. It holds the previous value of flag, possibly for timing or synchronization purposes in the FSM.

**Block 3: io\_busy Signal**

verilog

Copy code

always@(posedge clk or posedge rst)

if(rst) begin

io\_busy <= 1'h0;

end else if(((state == IDLE) & (~i\_cmd\_valid)) || (state == LOCK) || (state == WAIT))begin

io\_busy <= 1'h0;

end else begin

io\_busy <= 1'h1;

end

* **io\_busy**: This signal indicates whether the system is busy processing an operation.
* **if (rst)**: On reset, io\_busy is cleared (set to 1'h0).
* **else if(...)**: If the system is in the IDLE, LOCK, or WAIT state and no command is valid (~i\_cmd\_valid), io\_busy remains low (not busy).
* **else**: If the conditions for being idle are not met, io\_busy is set to 1'h1 (indicating that the system is busy).

**Purpose:**

io\_busy is used to indicate whether the system is currently processing an operation or waiting for new commands.

**Block 4: o\_we\_n Signal (Write Enable)**

verilog

Copy code

always@(posedge clk or posedge rst)

if(rst) begin

o\_we\_n <= 1'h1;

end else if(((state == CMD1) || (state == ADDR) || (state == CMD2)) && is\_we\_edge)begin

o\_we\_n <= ~o\_we\_n;

end

* **o\_we\_n**: This signal controls the write enable (WE) for external devices, such as memory.
* **if (rst)**: On reset, o\_we\_n is set to 1'h1 (indicating no write operation).
* **else if(...)**: If the current state is CMD1, ADDR, or CMD2 and the signal is\_we\_edge is true (likely indicating the correct timing edge for writing), o\_we\_n is toggled (~o\_we\_n).

**Purpose:**

o\_we\_n manages the write enable control signal for the memory. It toggles during the appropriate states (CMD1, ADDR, CMD2) when the FSM is ready to perform a write operation.

**Block 5: o\_cle Signal (Command Latch Enable)**

verilog

Copy code

always@(posedge clk or posedge rst)

if(rst) begin

o\_cle <= 1'h0;

end else if((state == CMD1) || (state == CMD2) || flag || (i\_cmd\_valid && (i\_cmd[7:0] == 8'h85)))begin

o\_cle <= 1'h1;

end else begin

o\_cle <= 1'h0;

end

* **o\_cle**: This signal controls the command latch enable (CLE) for memory, which tells the memory when a command is being issued.
* **if (rst)**: On reset, o\_cle is set to 1'h0.
* **else if(...)**: The signal is set to 1'h1 (indicating a command is being latched) if the state is CMD1 or CMD2, or if flag is set, or if i\_cmd\_valid is true and the command (i\_cmd[7:0]) equals 8'h85.

**Purpose:**

o\_cle enables the memory command latch during specific states when a command needs to be sent to the memory (e.g., during CMD1, CMD2).

**Block 6: o\_ale Signal (Address Latch Enable)**

verilog

Copy code

always@(posedge clk or posedge rst)

if(rst) begin

o\_ale <= 1'h0;

end else if(state == ADDR)begin

o\_ale <= 1'h1;

end else begin

o\_ale <= 1'h0;

end

* **o\_ale**: This signal controls the address latch enable (ALE), which tells the memory when an address is being sent.
* **if (rst)**: On reset, o\_ale is set to 1'h0.
* **else if (state == ADDR)**: When the FSM is in the ADDR state (address phase), o\_ale is set to 1'h1.

**Purpose:**

o\_ale activates during the address phase (ADDR state) to indicate that the address should be latched by the memory.

**Block 7: state\_r Register (State Register)**

verilog

Copy code

always@(posedge clk or posedge rst)

if(rst) begin

state\_r <= 9'h0;

end else begin

state\_r <= state;

end

* **state\_r**: This register stores the current state of the FSM with one clock cycle delay.
* **if (rst)**: On reset, state\_r is cleared to 9'h0.
* **else**: The state\_r register captures the current state of the FSM (state) on every clock cycle.

**Purpose:**

state\_r holds the previous state of the FSM. It is used to compare the current and previous states to track transitions or store state history.

**Block 8: Write Enable (o\_we\_n) Control**

Verilog Code

always@(posedge clk or posedge rst)

if(rst) begin

o\_we\_n <= 1'h1;

end else if(((state == CMD1) || (state == ADDR) || (state == CMD2)) && is\_we\_edge) begin

o\_we\_n <= ~o\_we\_n;

end

**Purpose:**

* This block manages the write enable signal o\_we\_n. The o\_we\_n signal is active-low, meaning that when it is 0, a write operation occurs.

**Detailed Explanation:**

* **Reset Condition:** On reset (rst), o\_we\_n is set to 1'h1, indicating that the write operation is disabled.
* **Write Control:** When the state is either CMD1, ADDR, or CMD2, and a write edge (is\_we\_edge) is detected, the signal o\_we\_n is toggled (i.e., flipped between 0 and 1). This ensures that during specific command or address states, the write operation occurs at the appropriate time.

**Block 9: Command Latch Enable (o\_cle)**

Verilog Code

always@(posedge clk or posedge rst)

if(rst) begin

o\_cle <= 1'h0;

end else if((state == CMD1) || (state == CMD2) || flag || (i\_cmd\_valid && (i\_cmd[7:0] == 8'h85))) begin

o\_cle <= 1'h1;

end else begin

o\_cle <= 1'h0;

end

**Purpose:**

* The o\_cle signal is used to indicate that a command is being latched (written to a command register). It becomes active when a command is present.

**Detailed Explanation:**

* **Reset Condition:** On reset, o\_cle is set to 0, meaning that the command latch is inactive.
* **Active Conditions:** o\_cle is set to 1 in the following cases:
  + During the CMD1 or CMD2 states, indicating that the system is currently handling a command.
  + If the flag signal is set, which may indicate that another command is expected.
  + If a command is valid (i\_cmd\_valid) and its opcode (i\_cmd[7:0]) is 8'h85, which seems to be a specific case where the command latch needs to be enabled.
* **Default:** In other states, o\_cle is deactivated (0).

**Block 10: Address Latch Enable (o\_ale)**

Verilog Code

always@(posedge clk or posedge rst)

if(rst) begin

o\_ale <= 1'h0;

end else if(state == ADDR) begin

o\_ale <= 1'h1;

end else begin

o\_ale <= 1'h0;

end

**Purpose:**

* The o\_ale signal is used to indicate that an address is being latched (written to an address register). It is only active during the address state.

**Detailed Explanation:**

* **Reset Condition:** On reset, o\_ale is set to 0, meaning that the address latch is inactive.
* **Active Condition:** If the state is ADDR, the system is handling an address, so o\_ale is set to 1.
* **Default:** In other states, the address latch is deactivated (0).

**Block 11: State Register (state\_r)**

Verilog Code

always@(posedge clk or posedge rst)

if(rst) begin

state\_r <= 9'h0;

end else begin

state\_r <= state;

end

**Purpose:**

* The state\_r register stores the previous state of the system.

**Detailed Explanation:**

* **Reset Condition:** On reset, state\_r is cleared to 9'h0, representing an idle or reset state.
* **State Latching:** On each clock cycle, state\_r is updated to the current state (state), effectively keeping track of the previous state. This is useful for operations that need to reference both the current and previous state.

**Block 12: Result Valid and Result ID (o\_res\_valid, o\_res\_id)**

Verilog Code

always@(posedge clk or posedge rst)

if(rst) begin

o\_res\_valid <= 1'h0;

o\_res\_id <= 16'h0;

end else if(((state\_r == BUSY) || (state\_r == WPST)) && (state == IDLE)) begin

o\_res\_valid <= 1'h1;

o\_res\_id <= i\_cmd\_id;

end else begin

o\_res\_valid <= 1'h0;

// o\_res\_id <= 16'h0;

end

**Purpose:**

* These signals handle the result validation and return the result ID after a command has completed.

**Detailed Explanation:**

* **Reset Condition:** On reset, o\_res\_valid and o\_res\_id are cleared, indicating no valid result and no result ID.
* **Result Generation:** If the previous state (state\_r) was BUSY or WPST and the current state is IDLE, this means that a command has completed, so o\_res\_valid is set to 1 (indicating a valid result), and o\_res\_id is set to the current command ID (i\_cmd\_id).
* **Default:** In other cases, o\_res\_valid is cleared to 0, signaling that no valid result is available at that time.

**Block 13: Read Status Request (o\_rd\_st\_req, o\_rd\_st\_type, o\_rd\_st\_addr, o\_rd\_st\_id)**

Verilog Code

always@(posedge clk or posedge rst)

if(rst) begin

o\_rd\_st\_req <= 1'h0;

o\_rd\_st\_type <= 3'h0;

o\_rd\_st\_addr <= 24'h0;

o\_rd\_st\_id <= 16'h0;

end else if((state == WAIT) && (i\_rb\_n == 1'h1) && (~i\_keep\_wait)) begin

o\_rd\_st\_req <= 1'h1;

o\_rd\_st\_type <= {2'b0, busy\_time[10]};

o\_rd\_st\_addr <= i\_addr[39:16];

o\_rd\_st\_id <= i\_cmd\_id;

end else begin

o\_rd\_st\_req <= 1'h0;

o\_rd\_st\_type <= 3'h0;

o\_rd\_st\_addr <= 24'h0;

o\_rd\_st\_id <= 16'h0;

end

**Purpose:**

* These signals handle the request to read status and provide relevant details like address and ID for the request.

**Detailed Explanation:**

* **Reset Condition:** On reset, all the request signals (o\_rd\_st\_req, o\_rd\_st\_type, o\_rd\_st\_addr, and o\_rd\_st\_id) are cleared.
* **Status Request Generation:** When the state is WAIT, the i\_rb\_n signal is asserted (1'h1), and i\_keep\_wait is not active, a read status request (o\_rd\_st\_req) is generated. The type of request is encoded in o\_rd\_st\_type, which uses the busy\_time[10] signal. The request address is derived from the upper 24 bits of the i\_addr signal, and the command ID is assigned to o\_rd\_st\_id.
* **Default:** In all other states, no status request is generated, and all outputs are reset.

**Block 14: Chip Enable (o\_ce\_n) Control**

Verilog Code

always@(posedge clk or posedge rst)

if(rst) begin

o\_ce\_n <= 1'h1;

end else if(((state == IDLE) & (~i\_cmd\_valid)) || (state == LOCK) || (state == WAIT)) begin

o\_ce\_n <= 1'h1;

end else begin

o\_ce\_n <= 1'h0;

end

**Purpose:**

* This block manages the chip enable signal (o\_ce\_n), which is active-low. It controls when the chip is enabled or disabled based on the state.

**Detailed Explanation:**

* **Reset Condition:** On reset, o\_ce\_n is set to 1'h1, disabling the chip.
* **Chip Enable Control:** If the state is IDLE and no command is valid (~i\_cmd\_valid), or the state is LOCK or WAIT, the chip is disabled (1'h1). Otherwise, the chip is enabled (1'h0), allowing operations to proceed.
* **Default:** The chip remains enabled in all other cases.

**Block 16: Data Strobe (o\_dqs)**

Verilog Code

always@(posedge clk or posedge rst)

if(rst) begin

o\_dqs <= 4'hf;

end else if((state == DATA) & (~data\_src) & (~i\_wvalid)) begin

o\_dqs <= {4{o\_dqs[3]}};

end else if(state == DATA) begin

o\_dqs <= 4'h5;

end else if((state\_r == WPRE) || (state == WPST) || (state\_r == WPST) || (flag | flag\_r)) begin

o\_dqs <= 4'h0;

end else begin

o\_dqs <= 4'hf;

end

**Purpose:**

* The o\_dqs signal controls the data strobe signal, which synchronizes the timing for data reads or writes. This logic ensures correct timing behavior for data transfers.

**Detailed Explanation:**

* **Reset Condition:** On reset, o\_dqs is set to 4'hf, which implies that the data strobe lines are disabled or inactive.
* **DATA State (With ~data\_src and ~i\_wvalid):** If the system is in the DATA state, and the data\_src signal is not active (~data\_src), and there is no valid write (~i\_wvalid), the o\_dqs is set to repeat the current o\_dqs[3] value across all 4 bits. This ensures consistent strobe behavior during this condition.
* **DATA State (General):** When the system is in the DATA state (and neither of the above conditions applies), the o\_dqs signal is set to 4'h5, a specific pattern to indicate the data strobe is active for data transfer.
* **WPRE, WPST, and Flag Conditions:** In case the state is WPRE (Write Precharge), WPST (Write Post), or if the flag or flag\_r is set, the data strobe is disabled by setting o\_dqs to 4'h0.
* **Default Case:** In all other cases, the data strobe is disabled and remains in its default inactive state (4'hf).

**Block 17: Data Bus Tri-State Enable (o\_dq\_tri\_en)**

Verilog Code

always@(posedge clk or posedge rst)

if(rst) begin

o\_dq\_tri\_en <= 1'h1;

end else if( (state == CMD1) || (state == ADDR) || (state == CMD2) || (state == DATA) ) begin

o\_dq\_tri\_en <= 1'h0; // output

end else begin

o\_dq\_tri\_en <= 1'h1;

end

**Purpose:**

* This block controls whether the data bus (o\_dq) is tri-stated or actively driven. In tri-state mode, the bus is effectively disconnected, allowing other devices to use it.

**Detailed Explanation:**

* **Reset Condition:** On reset, o\_dq\_tri\_en is set to 1'h1, meaning the data bus is in a tri-state mode (disconnected).
* **Active Output Conditions:** When the state is CMD1, ADDR, CMD2, or DATA, the data bus is actively driven (o\_dq\_tri\_en = 1'h0), meaning the output data is valid and can be driven to external devices.
* **Default:** In other states, the data bus is tri-stated by setting o\_dq\_tri\_en back to 1'h1.

**Block 18: Data Address Calculation (dq\_addr) and Data Preparation (dq\_data)**

Verilog Code

wire [7:0] dq\_addr;

assign dq\_addr = (i\_addr >> {addr\_cnt, 3'h0}) & 8'hff;

reg [31:0] dq\_data;

always @\*

if(data\_cnt < WARMUP\_DATA\_NUM) begin

dq\_data = i\_data[31:0];

end else begin

dq\_data = (i\_data >> ({data\_cnt, 3'h0} - (WARMUP\_DATA\_NUM << 3))) & 32'hffff\_ffff;

end

**Purpose:**

* The dq\_addr and dq\_data signals are used to calculate the appropriate address and prepare the data to be sent on the data bus during the DATA state.

**Detailed Explanation:**

* **Data Address (dq\_addr):** This 8-bit address is calculated by right-shifting the input address (i\_addr) by the value addr\_cnt (multiplied by 8 using {addr\_cnt, 3'h0}), and then masking it with 8'hff to ensure only 8 bits are taken. This address is used during the ADDR state to specify the memory address being accessed.
* **Data Preparation (dq\_data):**
  + If the data\_cnt (data counter) is less than the predefined WARMUP\_DATA\_NUM, the first 32 bits of i\_data are selected and assigned to dq\_data.
  + Otherwise, the i\_data is shifted by the amount (data\_cnt \* 8 - WARMUP\_DATA\_NUM \* 8) and masked with 32'hffff\_ffff to ensure that only the required portion of data is extracted for transmission.

This logic prepares the correct data based on the current data counter and how many cycles the data transfer has been ongoing.

**Block 19: Data Output (o\_dq)**

Verilog Code

always@(posedge clk or posedge rst)

if(rst) begin

o\_dq <= 32'h0;

end else if(state == CMD1) begin

o\_dq <= {4{i\_cmd[7:0]}};

end else if(state == CMD2) begin

o\_dq <= {4{i\_cmd[15:8]}};

end else if(state == ADDR) begin

o\_dq <= {4{dq\_addr}};

end else if((state == DATA) & data\_src) begin

o\_dq <= dq\_data;

end else if((state == DATA) & (~data\_src) & (~i\_wvalid)) begin

o\_dq <= {4{i\_wdata[31:24]}};

end else if((state == DATA) & (~data\_src)) begin

o\_dq <= i\_wdata;

end else begin

o\_dq <= 32'h0;

end

**Purpose:**

* The o\_dq signal represents the output data bus. This block manages what data gets placed onto the data bus depending on the current state and input conditions.

**Detailed Explanation:**

* **Reset Condition:** On reset, o\_dq is cleared to 32'h0, meaning no data is placed on the bus.
* **CMD1 and CMD2 States:**
  + During the CMD1 state, the lower byte of the command (i\_cmd[7:0]) is repeated 4 times and placed on the o\_dq bus.
  + During the CMD2 state, the upper byte of the command (i\_cmd[15:8]) is repeated 4 times and placed on the bus.
* **ADDR State:** In the ADDR state, the calculated address (dq\_addr, prepared in Block 18) is repeated 4 times and placed on the o\_dq bus.
* **DATA State (With data\_src Active):** If data\_src is active during the DATA state, the prepared data (dq\_data) is placed onto the data bus.
* **DATA State (Without data\_src, Write Data Mode):** If data\_src is not active, and there is no valid write (~i\_wvalid), the upper byte of i\_wdata is repeated 4 times and placed on the bus. If i\_wvalid is true, the full i\_wdata (32 bits) is placed on the bus.
* **Default Case:** In all other cases, the data bus is cleared (32'h0).

**Block 20: Write Ready (o\_wready)**

Verilog Code

assign o\_wready = (state == DATA) & (data\_cnt >= WARMUP\_DATA\_NUM);

**Purpose:**

* The o\_wready signal indicates when the system is ready to accept new write data.

**Detailed Explanation:**

* The system is ready for a write operation when the current state is DATA and the data\_cnt is greater than or equal to the WARMUP\_DATA\_NUM. This ensures that the write pipeline is warmed up and ready to accept data for processing.