Inputs:

- `clk`: Clock signal, used to synchronize all actions in the design.
- `rst`: Reset signal, to reset the state machine and registers.
- `i_cmd_valid`: Input signal to indicate if a command is valid.
- `i_cmd`: 16-bit input command (first cycle and second cycle).
- `i_cmd_id`: 16-bit input command ID.
- `i_addr`: 40-bit input address (contains row and column address information).
- `i_cmd_param`: 32-bit input command parameter (additional control data for the command).
- `i_keep_wait`: Input signal indicating if the controller should remain in the WAIT state.
- `i_rready`: Input signal indicating if there is space available for reading data.
- `i_rb_n`: NAND ready/busy signal. Indicates if the NAND flash is ready for operations.
- `i_dqs`: Data strobe signal, used to synchronize data transfer.
- `i_dq`: 32-bit data input from the NAND flash.

Outputs:

- `o_cmd_ready`: 1-bit output indicating if the controller is ready to accept a new command.
- `o_status`: 2-bit status output, indicating IDLE, BUSY, WAIT, or READY state.
- `o_rvalid`: 1-bit output indicating if read data is valid.
- `o_rdata`: 32-bit output representing the read data from the NAND flash.
- `o_rlast`: Output signal indicating the last piece of data in a read operation.
- `o_rid`: 16-bit output representing the command ID for the read operation.
- `o_ruser`: 16-bit output representing user-defined metadata for the read operation.
- `io_busy`: 1-bit output indicating if the controller is busy.
- `o_ce_n`: Chip enable signal (active-low), used to select the NAND flash.
- `o_we_n`: Write enable signal, used to write data to the NAND flash.
- `o_cle`: Command latch enable, used to send commands to the NAND flash.
- `o_ale`: Address latch enable, used to send addresses to the NAND flash.
- `o_re`: 4-bit output representing read enable signals.
- `o_dqs_tri_en`: Tri-state enable signal for the data strobe, used to switch between input and output.
- `o_dq_tri_en`: Tri-state enable signal for the data bus, used to switch between input and output.
- `o_dq`: 32-bit output representing the data to be written to the NAND flash.

`localparam` Definitions

```
verilog
Copy code
localparam
    IDLE  = 10'b00_0000_0001,
    CMD1  = 10'b00_0000_0010,
    ADDR  = 10'b00_0000_0100,
    CMD2  = 10'b00_0000_1000,
    BUSY  = 10'b00_0001_0000,
    LOCK  = 10'b00_0010_0000,
    WAIT  = 10'b00_0100_0000,
    RPRE  = 10'b00_1000_0000,
    DATA  = 10'b01_0000_0000,
    RPST  = 10'b10_0000_0000;
```

Explaination:

- **Purpose**: `localparam` is used to define constants that are local to this module.
- These constants represent the states of a finite state machine (FSM).
    - `IDLE`: The controller is idle.
    - `CMD1`: The controller is in the first command phase.
    - `ADDR`: The controller is in the address phase, sending addresses to the NAND flash.
    - `CMD2`: The second command phase.
    - `BUSY`: The controller is waiting for the NAND flash to become ready.
    - `LOCK`: The controller is waiting for the `i_rb_n` signal to change.
    - `WAIT`: The controller is waiting for an external condition.
    - `RPRE`: The controller is preparing to read data.
    - `DATA`: The controller is reading data from the NAND flash.
    - `RPST`: The controller is finishing the read operation.

Registers

Verilog Code

```
reg  [ 9:0] state;
reg         has_cmd2;
reg  [ 2:0] addr_num;
reg  [11:0] busy_time;
reg  [14:0] data_num;
```

Explaination

- **Registers** store values over time in hardware. They hold state between clock cycles.
    - `state`: A 10-bit register representing the current state of the FSM.
    - `has_cmd2`: A 1-bit flag indicating if there is a second command.
    - `addr_num`: A 3-bit register indicating how many addresses need to be sent.
    - `busy_time`: A 12-bit register representing the number of cycles the controller should wait while the NAND flash is busy.
    - `data_num`: A 15-bit register representing the number of data bytes to be read.

- **State Transitions and Behavior**
- The rest of the code describes the behavior of the controller based on the state it is in. This is done using `always` blocks, which define how registers and outputs change on each clock cycle.

always @(posedge clk or posedge rst)

Verilog Code:

```
always @(posedge clk or posedge rst) begin
    if (rst) begin
        o_cmd_ready    <= 1'b1;
        io_busy        <= 1'b0;
        o_status       <= 2'b00;
        state          <= IDLE;
        has_cmd2       <= 1'b0;
        addr_num       <= 3'b0;
        busy_time      <= 12'b0;
        data_num       <= 15'b0;
    end else begin
        case (state)
            IDLE: begin
                o_cmd_ready <= 1'b1;
                io_busy     <= 1'b0;
                o_status    <= 2'b00;

                if (i_cmd_valid) begin
                    state    <= CMD1;
                    o_cmd_ready <= 1'b0;
                    has_cmd2 <= (i_cmd_param[0] == 1'b1);
                    addr_num <= i_cmd_param[3:1];
                    data_num <= i_cmd_param[18:4];
                end
            end
```

Reset and Initialization:

- This block is sensitive to the **positive edge** of the clock (`posedge clk`) and the reset (`rst`).
    - If `rst` is active (`rst == 1`), it resets all the registers to their initial values:
        - `o_cmd_ready`: Command ready signal is set to `1`, indicating the controller is ready to accept a new command.
        - `io_busy`: Busy flag is set to `0` (controller is not busy).
        - `o_status`: Status register is set to `00` (IDLE state).
        - `state`: The state machine moves to `IDLE`.
        - `has_cmd2`: Flag indicating if there is a second command is set to `0`.
        - `addr_num`: Number of addresses to send is set to `0`.
        - `busy_time`: The busy timer is reset to `0`.
        - `data_num`: The number of data bytes to read is reset.

IDLE State:

- When the reset is not active (`rst == 0`), the state machine begins in the `IDLE` state.
  - In this state, the controller is waiting for a command.
  - `o_cmd_ready` is set to `1`, meaning the controller is ready to accept a new command.
  - `io_busy` is `0` because the controller is not performing any operations.
  - If `i_cmd_valid` is `1`, indicating a new command has been issued, the state machine transitions to the `CMD1` state.
    - `o_cmd_ready` is set to `0`, indicating the controller is busy handling the command.
    - `has_cmd2`: Set to `1` if the least significant bit of `i_cmd_param` is `1`, indicating that the command has a second part.
    - `addr_num`: This is the number of address cycles (from bits [3:1] of `i_cmd_param`).
    - `data_num`: This is the number of data cycles (from bits [18:4] of `i_cmd_param`).

CMD1 State: Sending First Command

Verilog Code:

```
CMD1: begin
    o_cle <= 1'b1;
    o_ale <= 1'b0;
    o_dq  <= {16'b0, i_cmd};
    o_dq_tri_en <= 1'b1;

    if (i_keep_wait) begin
        state <= WAIT;
    end else if (!i_rb_n) begin
        state <= CMD2;
        o_cle <= 1'b0;
    end else begin
        state <= ADDR;
    end
end
```

- In the `CMD1` state:
  - `o_cle` (Command Latch Enable) is set to `1`, enabling the command latch so the NAND flash can receive a command.
  - `o_ale` (Address Latch Enable) is `0`, meaning we are not sending addresses in this state.
  - `o_dq`: The output data bus is loaded with the command (`i_cmd`) and padded with zeros in the upper 16 bits (`{16'b0, i_cmd}`).
  - `o_dq_tri_en` is set to `1`, enabling the output on the data bus.
- The next state is determined based on the conditions:
  - If `i_keep_wait` is high, the controller moves to the `WAIT` state.

- If `i_rb_n` (Ready/Busy signal) is low, indicating the NAND flash is not ready, the controller moves to the `CMD2` state.
- Otherwise, it moves to the `ADDR` state, where it sends addresses to the NAND flash.

ADDR State: Sending Address

```
Verilog Code

ADDR: begin
    o_ale <= 1'b1;
    o_cle <= 1'b0;
    o_dq  <= i_addr[addr_num * 8 +: 8];
    addr_num <= addr_num - 1;

    if (addr_num == 3'b0) begin
        state <= CMD2;
    end
end
```

- In the `ADDR` state:
  - `o_ale` is set to `1`, enabling the address latch to send the address to the NAND flash.
  - `o_cle` is set to `0`, disabling the command latch.
  - The address (`i_addr`) is split into 8-bit chunks and sent over the data bus (`o_dq`).
  - `addr_num` is decremented with each cycle until all address cycles are complete.
- Once all addresses are sent (`addr_num == 0`), the state machine transitions to the `CMD2` state.

CMD2 State: Sending Second Command

Verilog Code

```
CMD2: begin
    if (has_cmd2) begin
        o_cle <= 1'b1;
        o_ale <= 1'b0;
        o_dq  <= {16'b0, i_cmd};
        o_dq_tri_en <= 1'b1;
    end

    if (!i_rb_n) begin
        state <= BUSY;
        o_cle <= 1'b0;
    end
end
```

- In the `CMD2` state:
  - If `has_cmd2` is `1`, the second command is sent.
  - `o_cle` is set to `1`, enabling the command latch to send the second command.
  - The command is placed on the data bus (`o_dq`), and the data tri-state is enabled.

- If `i_rb_n` is low, indicating the NAND flash is not ready, the controller moves to the `BUSY` state to wait for the NAND flash to become ready.

BUSY State: Waiting for NAND Flash

Verilog Code

```
BUSY: begin
    if (i_rb_n) begin
        if (data_num == 15'b0) begin
            state <= IDLE;
            o_status <= 2'b11;
        end else begin
            state <= RPRE;
        end
    end
end
```

- In the `BUSY` state:
  - The controller waits for the NAND flash to become ready (`i_rb_n == 1`).
  - If `data_num` is `0`, meaning no data needs to be read, the controller returns to the `IDLE` state and sets `o_status` to `11` (READY).
  - Otherwise, it transitions to the `RPRE` (Read Prepare) state to begin reading data.

RPRE State: Preparing to Read

Verilog Code

```
RPRE: begin
    o_dqs_tri_en <= 1'b0;
    o_dq_tri_en  <= 1'b0;
    state <= DATA;
end
```

- In the `RPRE` state:
  - The data strobe tri-state (`o_dqs_tri_en`) and data bus tri-state (`o_dq_tri_en`) are disabled, setting the signals to input mode to receive data from the NAND flash.
  - The controller then transitions to the `DATA` state to begin reading data.

DATA State: Reading Data

Verilog Code

```
DATA: begin
    if (i_rready) begin
        o_rvalid <= 1'b1;
        o_rdata  <= i_dq;
        o_rid    <= i_cmd_id;
```

```
        o_ruser  <= i_cmd_param[31:16];
        data_num <= data_num - 1;

        if (data_num == 15'b0) begin
            state <= RPST;
        end
    end
end
```

- In the `DATA` state:
    - If `i_rready` is `1`, indicating that there is space to receive data, the controller:
        - Sets `o_rvalid` to `1`, indicating that valid data is available.
        - Loads the data from the NAND flash (`i_dq`) into `o_rdata`.
        - Sets `o_rid` to the command ID and `o_ruser` to user-defined metadata from the command parameter.
        - Decrements `data_num`, counting how many pieces of data have been read.
- If all the data has been read (`data_num == 0`), the state machine transitions to the `RPST` (Read Post) state.

RPST State: Finishing Read

```
Verilog Code

RPST: begin
    o_rvalid <= 1'b0;
    state <= IDLE;
    o_status <= 2'b11;
end
```

- In the `RPST` state:
    - The controller clears the `o_rvalid` signal, indicating that no more valid data is available.
    - The state machine returns to the `IDLE` state, and the `o_status` is set to `11` (READY).

Block 1: `io_busy` Signal Control

```
Verilog Code

always@(posedge clk or posedge rst)
if(rst) begin
    io_busy <= 1'h0;
end else if((state == IDLE) || (state == LOCK) || (state == WAIT)) begin
    io_busy <= 1'h0;
end else begin
    io_busy <= 1'h1;
end
```

Purpose:

The `io_busy` signal indicates whether the system is busy performing an operation. When `io_busy` is high, the system is actively working, and when low, it's idle or waiting.

Detailed Explanation:

- **Reset condition (`rst`):** On reset, `io_busy` is set to `0` (inactive).
- **Idle, Lock, or Wait states:** In the `IDLE`, `LOCK`, or `WAIT` states, the system is not actively working, so `io_busy` is set to `0`.
- **Other states:** For all other states, `io_busy` is set to `1`, indicating that the system is busy performing a task.

Block 2: Write Enable (`o_we_n`)

```
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:

The `o_we_n` signal is used to control the write enable signal for the memory interface. The write enable signal is typically active low, meaning when `o_we_n` is `0`, writing to memory is enabled.

Detailed Explanation:

- **Reset condition:** On reset, `o_we_n` is set to `1`, disabling the write operation.

- **CMD1, ADDR, and CMD2 states:** During the `CMD1`, `ADDR`, or `CMD2` states, if `is_we_edge` is high (which may indicate a write condition or edge), the value of `o_we_n` is toggled (`~o_we_n`). This controls the timing of the write enable signal.

Block 3: 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)) begin
    o_cle <= 1'h1;
end else begin
    o_cle <= 1'h0;
end
```

Purpose:

The `o_cle` (Command Latch Enable) signal is used to control whether the command latch should be active. It's typically set high during command phases.

Detailed Explanation:

- **Reset condition:** On reset, `o_cle` is set to `0`, meaning the command latch is disabled.
- **CMD1 and CMD2 states:** During the `CMD1` and `CMD2` states, `o_cle` is set to `1`, enabling the command latch to capture the command.
- **Other states:** In all other states, `o_cle` is set to `0`, disabling the command latch.

Block 4: 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` (Address Latch Enable) signal controls the address latch. It should be active when the system is latching an address.

Detailed Explanation:

- **Reset condition:** On reset, `o_ale` is set to `0`, disabling the address latch.
- **ADDR state:** In the `ADDR` state, `o_ale` is set to `1`, allowing the address to be latched.
- **Other states:** In other states, the address latch is disabled (`o_ale = 0`).

Block 5: Read Enable (`o_re`)

```
Verilog Code

always@(posedge clk or posedge rst)
if(rst) begin
    o_re <= 4'hf;
end else if(((state == RPRE) && (rpre_cnt > 8'h1)) || (state == RPST)) begin
    o_re <= 4'h0;
end else if((state == DATA) & (~i_rready)) begin
    o_re <= {4{o_re[3]}};
end else if(state == DATA) begin
    o_re <= 4'h5;
end else begin
    o_re <= 4'hf;
end
```

Purpose:

The `o_re` signal controls the read enable lines. It is dynamically adjusted based on the state to enable or disable reading.

Detailed Explanation:

- **Reset condition:** On reset, `o_re` is set to `4'hf`, indicating that all read enables are inactive.
- **RPRE and RPST states:** In the `RPRE` (Read Pre) state with a counter condition or in the `RPST` (Read Post) state, `o_re` is set to `0`, enabling the read operation.
- **DATA state with `i_rready` low:** If the system is in the `DATA` state but the `i_rready` signal is low (indicating the system is not ready to receive data), the read enable is held constant based on the most significant bit of `o_re[3]`.
- **DATA state with `i_rready` high:** In the `DATA` state, if `i_rready` is high, `o_re` is set to `4'h5`, enabling the read lines in a specific pattern for data transfer.

Block 6: Chip Enable (`o_ce_n`)

```
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 == RPST) &&
(rpst_cnt >= `tRPST)) || (state == LOCK) || (state == WAIT)) begin
    o_ce_n <= 1'h1;
end else begin
```

```
    o_ce_n <= 1'h0;
end
```

Purpose:

The `o_ce_n` signal controls the chip enable, which activates or deactivates the memory chip.

Detailed Explanation:

- **Reset condition:** On reset, `o_ce_n` is set to `1`, disabling the chip.
- **IDLE, RPST, LOCK, WAIT states:** In the `IDLE` state with `i_cmd_valid` low (no valid command), the `RPST` state with a counter condition, or the `LOCK` or `WAIT` states, the chip is disabled (`o_ce_n = 1`).
- **Other states:** In all other states, the chip is enabled by setting `o_ce_n` to `0`.

Block 7: Data Strobe (`o_dqs`)

Verilog Code

```
assign o_dqs = 4'hf;
```

Purpose:

The `o_dqs` signal is assigned a fixed value of `4'hf`. It may represent an idle or inactive state for the data strobe.

Block 8: Data Strobe Tri-State Enable (`o_dqs_tri_en`)

Verilog Code

```
always@(posedge clk or posedge rst)
if(rst) begin
    o_dqs_tri_en <= 1'h0;
end else if(((state == RPRE) && (rpre_cnt >= `tDQSRH)) || (state == DATA) ||
(state == RPST)) begin
    o_dqs_tri_en <= 1'h1;
end else begin
    o_dqs_tri_en <= 1'h0;
end
```

Purpose:

The `o_dqs_tri_en` signal controls the tri-state behavior of the data strobe lines (`dqs`), determining whether they should be active or floating (disconnected).

Detailed Explanation:

- **Reset condition:** On reset, `o_dqs_tri_en` is set to `0`, disabling the tri-state (data strobe is not floating).
- **RPRE, DATA, RPST states:** In the `RPRE`, `DATA`, or `RPST` states, the `o_dqs_tri_en` signal is set to `1`, enabling the tri-state (data strobe lines are floating).
- **Other states:** In other states, `o_dqs_tri_en` is set to `0`.

Block 9: Data Output Register (`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 begin
    o_dq <= 32'h0;
end
```

Purpose:
`o_dq` is used to output data or address information to the memory or external devices.

Detailed Explanation:

- **Reset condition:** When `rst` is high, `o_dq` is set to `32'h0`, initializing the data output to zero.
- **CMD1 state:** When the system is in the `CMD1` state, `o_dq` is set to `{4{i_cmd[7:0]}}`. This replicates the lower 8 bits of `i_cmd` into a 32-bit wide output.
- **CMD2 state:** When the system is in the `CMD2` state, `o_dq` is set to `{4{i_cmd[15:8]}}`. This replicates the upper 8 bits of `i_cmd` into a 32-bit wide output.
- **ADDR state:** When the system is in the `ADDR` state, `o_dq` is set to `{4{dq_addr}}`. This outputs the address, `dq_addr`, which is an 8-bit address potentially used to interact with memory.
- **Other states:** For all other states, `o_dq` is set to `32'h0`, indicating no valid data.

Block 10: Tri-State Enable for Data Lines (`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)) begin
```

```
    o_dq_tri_en <= 1'h0;  // output
end else begin
    o_dq_tri_en <= 1'h1;
end
```

Purpose:
`o_dq_tri_en` controls whether the data lines are in a high-impedance state (tri-state) or actively driving values.

Detailed Explanation:

- **Reset condition:** When `rst` is high, `o_dq_tri_en` is set to `1'h1`, enabling the tri-state (high-impedance) mode.
- **CMD1, ADDR, CMD2 states:** In these states, `o_dq_tri_en` is set to `1'h0`, which enables the output on the data lines (`o_dq` is actively driving values).
- **Other states:** For all other states, `o_dq_tri_en` is set to `1'h1`, putting the data lines into tri-state (high-impedance mode).

Block 11: Address Calculation (`dq_addr`)

```
Verilog Code

wire [7:0] dq_addr;
assign dq_addr = (i_addr >> {addr_cnt, 3'h0}) & 8'hff;
```

Purpose:
`dq_addr` is used to compute a modified address based on the input address and a counter.

Detailed Explanation:

- **Address Calculation:** `dq_addr` is computed by shifting `i_addr` right by `addr_cnt` bits (where `addr_cnt` is combined with `3'h0` to form the shift amount) and then masking it with `8'hff` to ensure it is 8 bits wide.

Block 12: Data Strobe Output Enable (`dqs_out_en`)

```
Verilog Code
wire dqs_out_en;
assign dqs_out_en = ((state == DATA) || (state == RPST));
```

Purpose:
`dqs_out_en` indicates whether the data strobe (DQS) should be enabled based on the current state.

Detailed Explanation:

- **Enable Condition:** `dqs_out_en` is set to `1` if the system is in the `DATA` or `RPST` states. This signal controls the enabling of the data strobe.

Block 13: Data Input Counter (`din_cnt`)

```
Verilog Code
always@(posedge clk or posedge rst)
if(rst) begin
    din_cnt <= 4'h0;
end else if(state == RPRE) begin
    din_cnt <= 4'h0;
end else if(dqs_out_en && (i_dqs == 4'h5) && (din_cnt < WARMUP_DATA_NUM))
begin
    din_cnt <= din_cnt + 4'h4;
end
```

Purpose:
`din_cnt` counts the number of data input cycles, used to manage timing and readiness of the data input.

Detailed Explanation:

- **Reset condition:** When `rst` is high, `din_cnt` is reset to `4'h0`.
- **RPRE state:** When in the `RPRE` state, `din_cnt` is reset to `4'h0`.
- **Data Input Counting:** When `dqs_out_en` is true and `i_dqs` is `4'h5`, and `din_cnt` is less than `WARMUP_DATA_NUM`, `din_cnt` is incremented by `4'h4`. This keeps track of the number of data input cycles.

Block 14: Output Validity and Data (`o_rvalid`, `o_rdata`, `o_rid`, `o_ruser`)

```
Verilog Code

always@(posedge clk or posedge rst)
if(rst) begin
    o_rvalid <= 1'h0;
    o_rdata  <= 'h0;
    o_rid    <= 16'h0;
    o_ruser  <= 16'h0;
end else if(dqs_out_en && (i_dqs == 4'h5) && (din_cnt >= WARMUP_DATA_NUM))
begin
    o_rvalid <= 1'b1;
    o_rdata  <= i_dq;
    o_rid    <= i_cmd_id;
    o_ruser  <= i_cmd;
end else begin
    o_rvalid <= 1'h0;
end
```

Purpose:
This block controls the output validity and data transmission for the read operation.

Detailed Explanation:

- **Reset condition:** When `rst` is high, `o_rvalid`, `o_rdata`, `o_rid`, and `o_ruser` are reset to `0` or their default values.
- **Output Validity and Data:** When `dqs_out_en` is true, `i_dqs` is `4'h5`, and `din_cnt` is greater than or equal to `WARMUP_DATA_NUM`, `o_rvalid` is set to `1`, indicating that the read data is valid. `o_rdata`, `o_rid`, and `o_ruser` are set to `i_dq`, `i_cmd_id`, and `i_cmd`, respectively, providing the data, ID, and user information.

Block 15: DQS Register (`dqs_r`)

```
Verilog Code
reg [3:0] dqs_r;
always@(posedge clk or posedge rst)
if(rst) begin
    dqs_r <= 4'hf;
end else if(state == RPST) begin
    dqs_r <= i_dqs;
end else begin
    dqs_r <= 4'hf;
end
```

Purpose:
`dqs_r` holds the previous value of the data strobe signal (`i_dqs`) for comparison purposes.

Detailed Explanation:

- **Reset condition:** When `rst` is high, `dqs_r` is set to `4'hf`.
- **RPST state:** In the `RPST` state, `dqs_r` is updated with `i_dqs`.
- **Other states:** For all other states, `dqs_r` is set to `4'hf`.

Block 16: Read Last (`o_rlast`)

```
Verilog Code
assign o_rlast = (i_dqs == 4'h0) && (dqs_r == 4'h5);
```

Purpose:
`o_rlast` signals whether the current read operation is the last one based on the data strobe signal (`i_dqs`) and its previous value (`dqs_r`).

Detailed Explanation:

- **Read Last Condition:** `o_rlast` is set to `1` if `i_dqs` is `4'h0` (indicating the end of data strobe) and `dqs_r` was `4'h5` (indicating the previous state of the strobe). This indicates the end of a read operation.