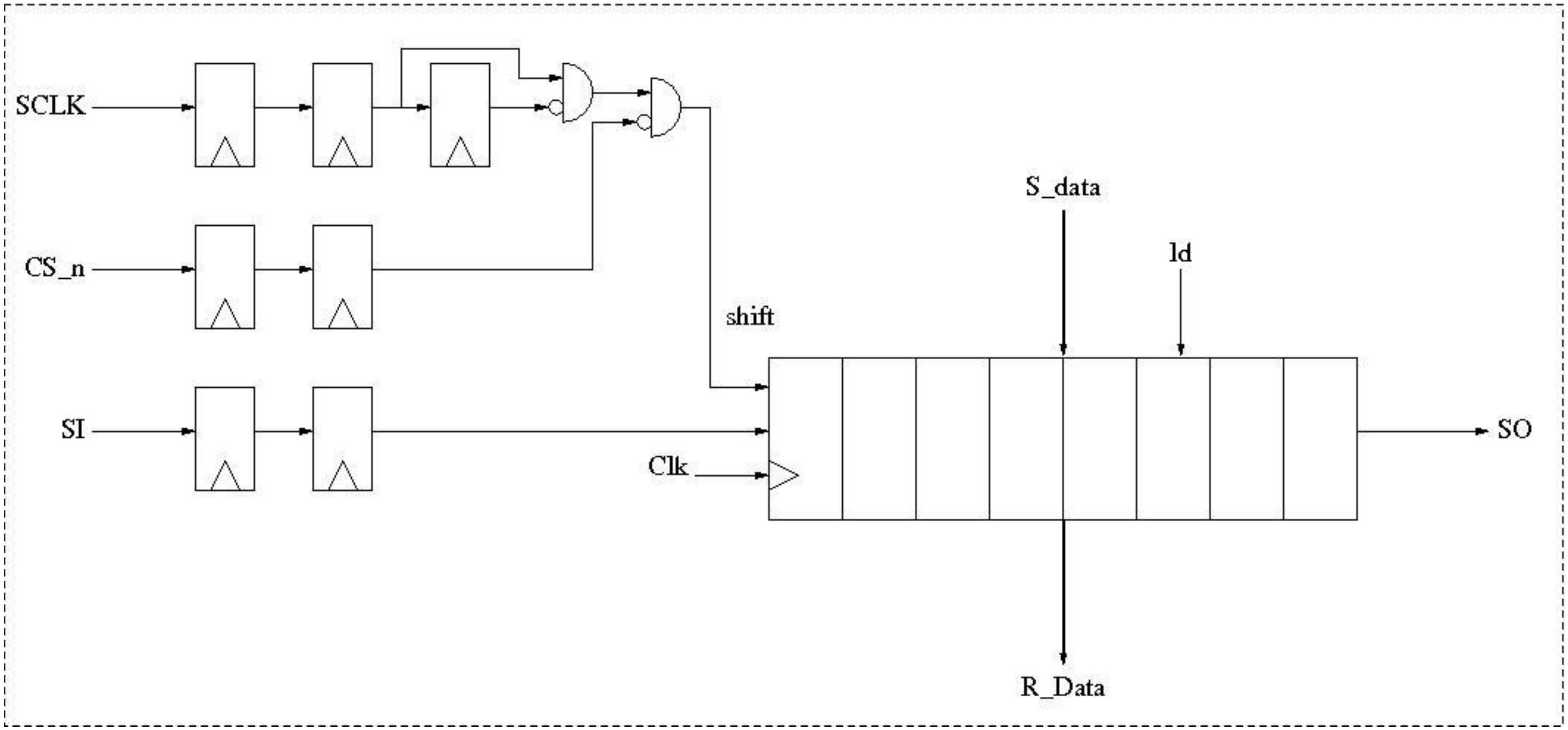
SPI (1/6)



SPI (2/6)

```
//
`timescale 1ns/1ps
//
// spi.v
//
module spi #(parameter REG_DQ = 1) (
   input                Clk,
   input                Reset,
   input                i_sclk,
   input                i_si,
   input                i_ss_n,
   output               o_so,
   input                i_ld,
   input          [7:0] i_data,
   output         [7:0] o_data,
   output reg           o_valid);
```

SPI (3/6)

```
//
// regs and wires
//
 reg [7:0] sreg;
 reg [2:0] sclk_reg, byte_cnt;
 reg [1:0] si_reg, ss_n_reg;
 reg       sclk_edg;
 reg       byte_cnt_end;
 reg       data_valid;
//
 wire       sreg_ld = i_ld;
//
```

SPI (4/6)

```
//
// Input Registers
//
 always @(posedge Clk) begin
    if(Reset) begin
       si_reg   <= #REG_DQ 0;
       sclk_reg <= #REG_DQ 0;
       ss_n_reg <= #REG_DQ 0;
       sclk_edg <= #REG_DQ 0;
    end
   else begin
      si_reg[0]   <= #REG_DQ i_si;
      si_reg[1]   <= #REG_DQ si_reg[0];
      ss_n_reg[0] <= #REG_DQ i_ss_n;
      ss_n_reg[1] <= #REG_DQ ss_n_reg[0];
      sclk_reg[0] <= #REG_DQ i_sclk;
      sclk_reg[1] <= #REG_DQ sclk_reg[0];
      clk_reg[2]  <= #REG_DQ sclk_reg[1];
      sclk_edg    <= #REG_DQ sclk_reg[1] &~sclk_reg[2];
    end end
```

SPI (5/6)

```
//
// Shift Register
//
 always @(posedge Clk) begin
    if(Reset) begin
       byte_cnt     <= #REG_DQ 0;
       sreg         <= #REG_DQ 0;
       byte_cnt_end <= #REG_DQ 0;
    end
    else begin
       if(sreg_ld) sreg <= #REG_DQ i_data;
       else if(sreg_en) begin
          byte_cnt     <= #REG_DQ byte_cnt + 1'b1;
          sreg         <= #REG_DQ {si_reg[1],sreg[7:1]};
          byte_cnt_end <= #REG_DQ (byte_cnt==7);
       end
    end
 end
//
 always @(posedge Clk) data_valid <= #REG_DQ byte_cnt_end;
```

SPI (6/6)

```
//
 always @(posedge Clk) data_valid <= #REG_DQ byte_cnt_end;
//
// Data In Register
//
 always @(posedge Clk) begin
    o_valid <= #REG_DQ byte_cnt_end &~data_valid;
 end
//
 assign o_so    = sreg[0];
 assign o_data  = sreg;
//
endmodule
```