```
//`include "source/spi_interface.svh"
//
// SPI slave, designed to share SPI bus with one other slave
//
// components:
//   sck rising edge detector, generates rsh_ena only when slave select asserted
//   serial to parallel shift reg with shift enable (rsh_ena)
//   bit counter: 0 to 7 counter, reset to 0, assert Done on count of 7,
//     enabled by rsh_ena, wrap-around 7 to 0
//   output buffer Ctrl.Rcvd with enable. Loads shift reg contents on Done signal
//   Ctrl.Ready = Done from bit counter

module slave #(parameter ID=0) (SPIbus.Slave Spis, SPIctrl.Slave Ctrl,
                                  input Clk_i, Rst_ni);

  logic sck1,sck2;                    // synchronizer for SPI clock input
  logic [3:0] bitcnt_r,bitcnt_nxt;    // count # bits in current 1 byte transfer
  logic rsh_ena,done;                 // shift enable for rcv, receive/xmit done
  logic [7:0] spi_in_nxt,spi_in_r;    // shift register for incoming data on MOSI
  logic [7:0] rcvd_nxt,rcvd_r;        // buffer data from SPI shift register
  logic mosi_sync1, mosi_sync2;       // synchronizer for MOSI input
  logic strobe2, strobe_sync;         // synchronizer for strobe input
  typedef enum {IS_EMPTY,IS_FULL} preXmitBuf_st_t;
  preXmitBuf_st_t preXmitBuf_st, preXmitBuf_st_nxt;  // xmit buff state, next state
  logic preXmitBuf_clear;             // xmit buffer clear for new value to xmit
  logic xsh_ena;                      // shift enable for transmit shift reg
  typedef enum {STROBE_WAIT, LOAD_WAIT, LOAD, XMIT} xmit_ctrl_st_t;
  xmit_ctrl_st_t xmit_ctrl_st, xmit_ctrl_st_nxt; // state, next state for xmit
  logic xmit_load;                    // enables load of xmit shift reg
  logic [7:0] preXmitBuf,preXmitBuf_nxt;  // buffer for data to be transmitted
  logic [7:0] xmit_r, xmit_nxt;       // xmit shift register
  logic ss1, ss2, ss_rise;            // slave select sync. ss_rise on ss rising edge


  // Receiving

  // registers associated with serial receipt of SPI data
  always_ff @(posedge Clk_i, negedge Rst_ni) begin
    if (Rst_ni == 1'b0) begin
      bitcnt_r <= 4'd0;
      sck1 <= 1'b0;
      sck2 <= 1'b0;
      mosi_sync1 <= 1'b0;
      mosi_sync2 <= 1'b0;
      rcvd_r <= 8'b0;
      spi_in_r <= 8'b0;
      end
    else begin
      bitcnt_r <= bitcnt_nxt;
      sck1 <= Spis.sck;
      sck2 <= sck1;
      mosi_sync1 <= Spis.mosi;
      mosi_sync2 <= mosi_sync1;
      rcvd_r <= rcvd_nxt;
      spi_in_r <= spi_in_nxt;
    end
  end

  // determines when receive shift reg takes in another bit
  assign rsh_ena = sck1 && !sck2 && (Spis.ss[ID]==1'b1);

  // increment bit count for each received bit
```

```
always_comb begin
  bitcnt_nxt = bitcnt_r;
  if (rsh_ena) begin
    if (~done)
      bitcnt_nxt = bitcnt_r + 1;
    else
      bitcnt_nxt = 4'd1;
    end
  else if (bitcnt_r == 4'd8)
    bitcnt_nxt = 4'd0;
end

assign done = (bitcnt_r == 4'd8); // byte transfer complete

// next state for receive shift register
always_comb begin
  spi_in_nxt = spi_in_r;
  if (rsh_ena)
    spi_in_nxt = {spi_in_r[6:0],mosi_sync2};
end

// next state for received data buffer
always_comb begin
  rcvd_nxt = rcvd_r;
  if (done)
    rcvd_nxt = spi_in_r;
end

assign Ctrl.Rcvd = rcvd_r; // return received data on control bus

// tell control bus that data is received
always_ff @(posedge Clk_i, negedge Rst_ni) begin
  if (Rst_ni == 1'b0) Ctrl.Ready <= 1'b0;
  else Ctrl.Ready <= done;
  end

// Transmitting

// busy flag register and next state, notify control bux
always_ff @(posedge Clk_i, negedge Rst_ni) begin
  if (Rst_ni == 1'b0) begin
    Ctrl.busy <= 1'b0;
    end
  else begin
    if (rsh_ena) begin
      Ctrl.busy <= 1'b1;
      end
    else if (done) begin
      Ctrl.busy <= 1'b0;
      end
    end
end

// most registers associated with transmission from slave
always_ff @(posedge Clk_i, negedge Rst_ni) begin
  if (Rst_ni == 1'b0) begin
    strobe2 <= 1'b0;
    strobe_sync <= 1'b0;
    preXmitBuf <= 8'b0;
    preXmitBuf_st <= IS_EMPTY;
    xmit_ctrl_st <= STROBE_WAIT;
    xmit_r <= 8'b0;
```

```
      ss1 <= 1'b0;
      ss2 <= 1'b0;
    end else begin
      strobe2 <= Ctrl.strobe;
      strobe_sync <= strobe2;
      preXmitBuf <= preXmitBuf_nxt;
      preXmitBuf_st <= preXmitBuf_st_nxt;
      xmit_ctrl_st <= xmit_ctrl_st_nxt;
      xmit_r <= xmit_nxt;
      ss1 <= Spis.ss[ID];
      ss2 <= ss1;
    end
  end

  assign ss_rise = !ss2 && ss1;

  // buffer new data to be transmitted only when buffer is available
  always_comb begin
    preXmitBuf_nxt = preXmitBuf;
    if (strobe_sync & (preXmitBuf_st == IS_EMPTY)) preXmitBuf_nxt = Ctrl.toXmit;
  end

  // monitor state of buffer for data to be transferred
  always_comb begin
    preXmitBuf_st_nxt = preXmitBuf_st;
    case (preXmitBuf_st)
      IS_EMPTY:
        if (strobe_sync) preXmitBuf_st_nxt = IS_FULL;
      IS_FULL:
        if (preXmitBuf_clear) preXmitBuf_st_nxt = IS_EMPTY;
    endcase
  end

  // register and next state for transmit buffer full flag on control bus
  always_ff @(posedge Clk_i, negedge Rst_ni) begin
    if (!Rst_ni) begin
      Ctrl.XmitFull <= 1'b0;
      end
    else begin
      if (xmit_load) begin
        Ctrl.XmitFull <= 1'b1;
        end
      else if (done) begin
        Ctrl.XmitFull <= 1'b0;
        end
      end
  end

  // only shift xmit shift register when data is ready to transmit
  assign xsh_ena = rsh_ena && (xmit_ctrl_st == XMIT);

  // xmit control FSM next state
  always_comb begin
    xmit_ctrl_st_nxt = xmit_ctrl_st;
    case (xmit_ctrl_st)
      STROBE_WAIT:  // wait for data to transmit
        if (preXmitBuf_st == IS_FULL) xmit_ctrl_st_nxt = LOAD_WAIT;
      LOAD_WAIT:    // wait for slave select and completion of previous transfer
        if ((bitcnt_r == 4'd8) || (Spis.ss[ID]==1'b0)) xmit_ctrl_st_nxt = LOAD;
      LOAD:         // load from preXmitBuff into xmit shift reg
        xmit_ctrl_st_nxt = XMIT;
      XMIT:         // enable transmission until done
```

```
        if (done) xmit_ctrl_st_nxt = STROBE_WAIT;
    endcase
  end

  // xmit FSM output logic
  always_comb begin
    xmit_load = 1'b0;
    preXmitBuf_clear = 1'b0;

    case (xmit_ctrl_st)
      LOAD: begin
        xmit_load = 1'b1; // enable load of XMIT SR
        end
      XMIT: begin         // allow xmit to proceed, clear xmit buff flag when done
        if ((xmit_ctrl_st == XMIT) & done) preXmitBuf_clear = 1'b1;
        end
    endcase
  end

  // next state for xmit shift register
  always_comb begin
    xmit_nxt = xmit_r;
    if (xmit_load) begin
      xmit_nxt = preXmitBuf;
      end
    else if (xsh_ena) begin
      xmit_nxt = xmit_r << 1;
      end
  end

  // slave MISO output
  assign Spis.miso = (Spis.ss[ID]==1'b1) ? xmit_r[7] : 1'bz;

endmodule
```