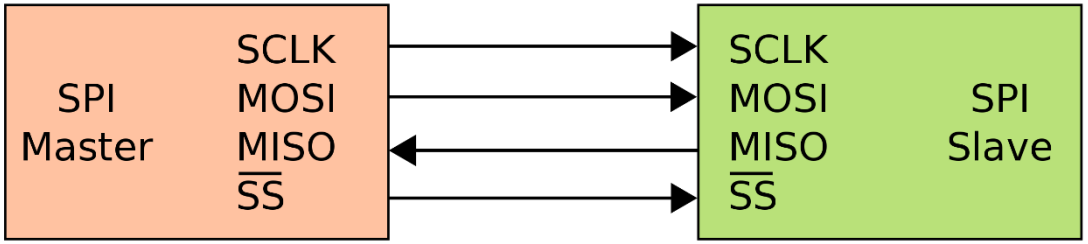
Protocollo SPI

Serial Peripheral Interface

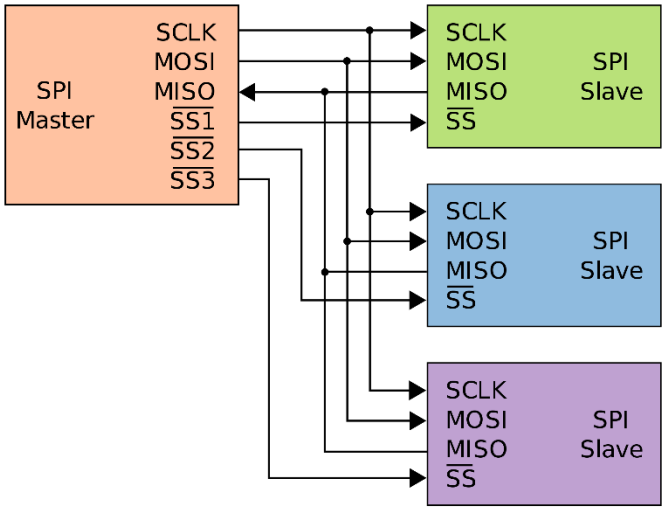
Introduzione

- Comunicazione seriale sincrona
- Sviluppata da Motorola nel 1980
- Divenuta «de facto» standard
  - Impiegata nelle Memorie SD
  - Display a Cristalli liquidi o a LED
  - Audio Codecs (TSC2101)
  - …
- Comunicazione Full-Duplex
- Di tipo Master /Slave
- Vi possono essere eventualmente più Slaves attivati attraverso segnali di tipo $\overline{CS}$ (Chip Select attivo basso)

Interfacce



- Quattro segnali
  - SCLK (Serial clock)
  - MOSI (Master Out Slave In) – DATI
  - MISO (Master In Slave Out) – DATI
  - nSS – Slave Select (attivo basso)
- Master
  - Produce SCLK, MOSI, nSS
  - Riceve MISO
- Slave
  - Riceve SCLK, MOSI, nSS
  - Produce MISO (sincronizzato su SCLK)

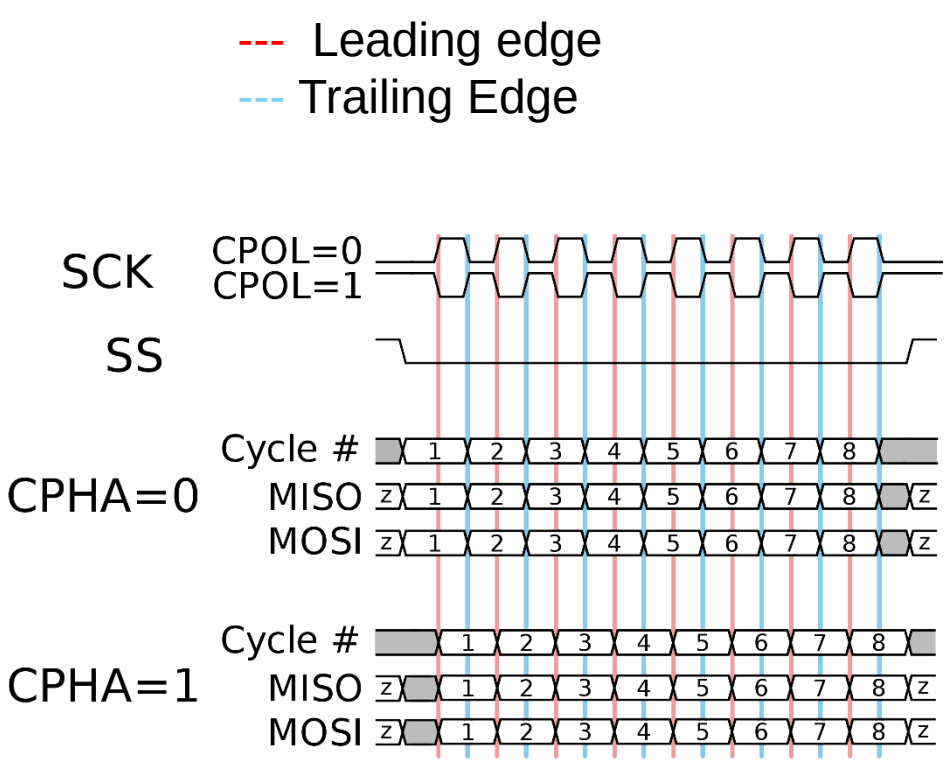


***NOTA**: alcuni dispositivi Slave richiedono espressamente un fronte di discesa su nSS per attivarsi (non basta metterlo fisso a massa)*

Comunicazione

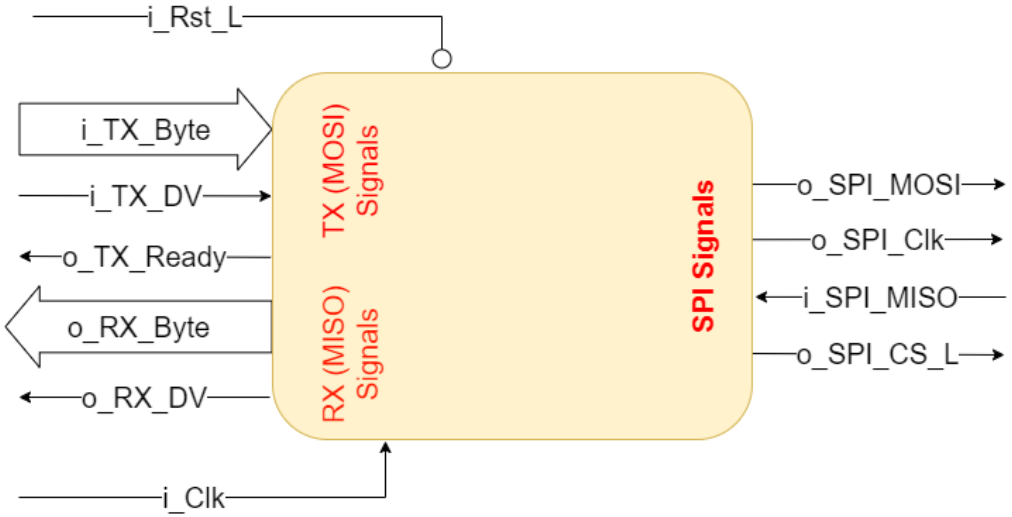
- Ci sono quattro modalità diverse per
  - Polarità del clock
  - Fase dei dati

SPI mode	Clock polarity (CPOL)	Clock phase (CPHA)	Clock edge (CKE/NCPHA)
0	0	0	1
1	0	1	0
2	1	0	1
3	1	1	0



Verilog Code (16 Bits)

- I/O Interface

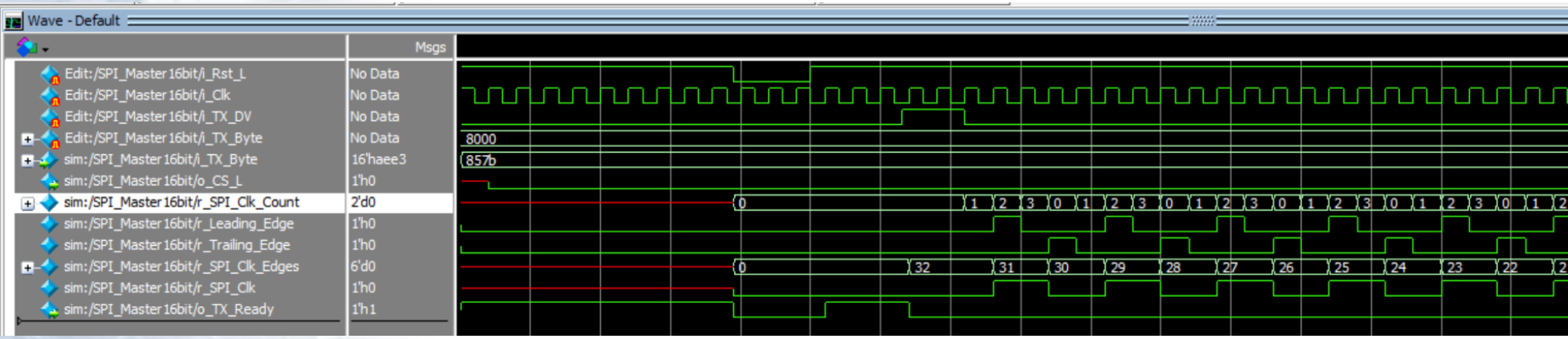


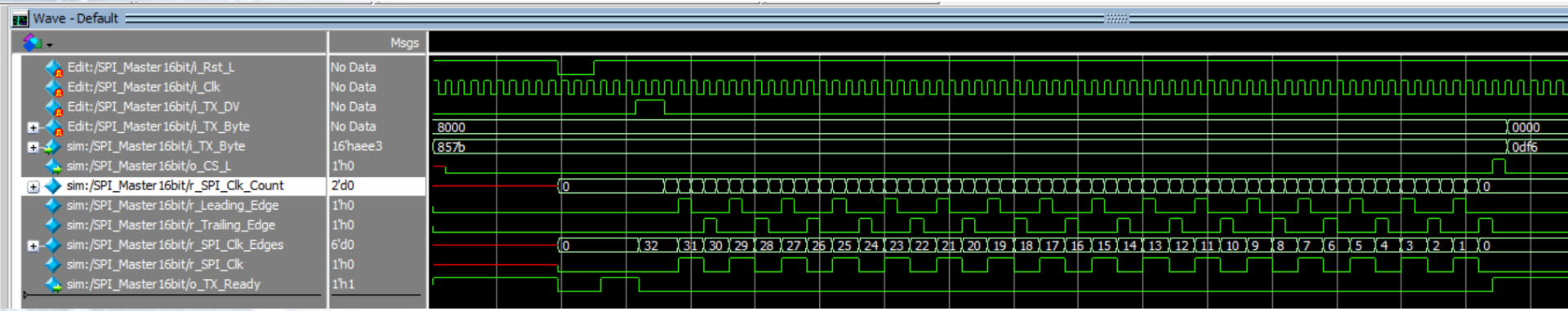

```
33  module SPI_Master16bit
34    #(parameter SPI_MODE = 0,
35      parameter CLKS_PER_HALF_BIT = 2) //250
36    (
37     // Control/Data Signals,
38     input        i_Rst_L,     // FPGA Reset
39     input        i_Clk,       // FPGA Clock
40
41     // TX (MOSI) Signals
42     input [15:0]  i_TX_Byte,        // Byte to transmit on MOSI
43     input        i_TX_DV,          // Data Valid Pulse with i_TX_Byte
44     output reg   o_TX_Ready,       // Transmit Ready for next byte
45
46     // RX (MISO) Signals
47     output reg       o_RX_DV,     // Data Valid pulse (1 clock cycle)
48     output reg [15:0] o_RX_Byte,   // Byte received on MISO
49
50     // SPI Interface
51     output reg o_SPI_Clk,
52     input      i_SPI_MISO,
53     output reg o_SPI_MOSI,
54
55     // Chip Select
56     output reg o_CS_L
57     );
58
```

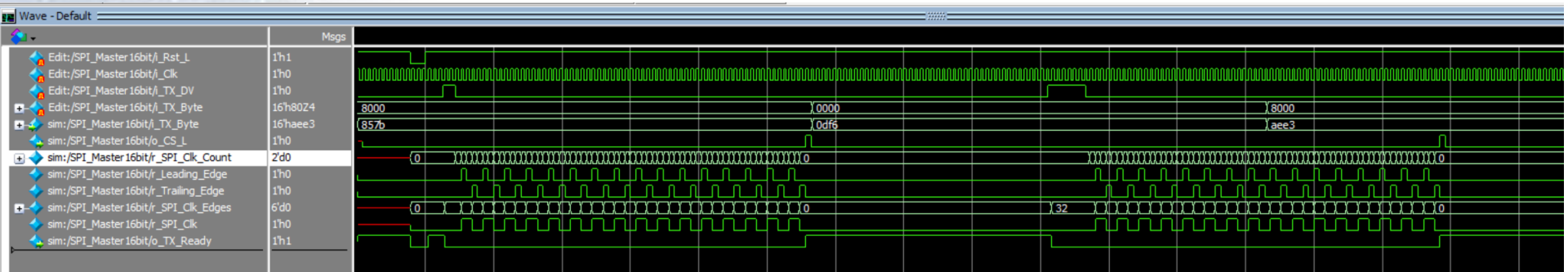
SPI – CLK and timing generation

```
// Purpose: Generate SPI Clock correct number of times when DV pulse comes
always @(posedge i_Clk or negedge i_Rst_L)
begin
  if (~i_Rst_L)
  begin
    o_TX_Ready      <= 1'b0;
    r_SPI_Clk_Edges <= 0;
    r_Leading_Edge  <= 1'b0;
    r_Trailing_Edge <= 1'b0;
    r_SPI_Clk       <= w_CPOL; // assign default state to idle state
    r_SPI_Clk_Count <= 0;
  end
  else
  begin

    // Default assignments
    r_Leading_Edge  <= 1'b0;
    r_Trailing_Edge <= 1'b0;

    if (i_TX_DV)
    begin
      o_TX_Ready      <= 1'b0;
      r_SPI_Clk_Edges <= 32;  // Total # edges in one byte
    end
    else if (r_SPI_Clk_Edges > 0)
    begin
      o_TX_Ready <= 1'b0;

      if (r_SPI_Clk_Count == CLKS_PER_HALF_BIT*2-1)
      begin
        r_SPI_Clk_Edges <= r_SPI_Clk_Edges - 1'b1;
        r_Trailing_Edge <= 1'b1;
        r_SPI_Clk_Count <= 0;
        r_SPI_Clk       <= ~r_SPI_Clk;
      end
      else if (r_SPI_Clk_Count == CLKS_PER_HALF_BIT-1)
      begin
        r_SPI_Clk_Edges <= r_SPI_Clk_Edges - 1'b1;
        r_Leading_Edge  <= 1'b1;
        r_SPI_Clk_Count <= r_SPI_Clk_Count + 1'b1;
        r_SPI_Clk       <= ~r_SPI_Clk;
      end
      else
      begin
        r_SPI_Clk_Count <= r_SPI_Clk_Count + 1'b1;
      end
    end
    else
    begin
      o_TX_Ready <= 1'b1;
    end


  end // else: !if(~i_Rst_L)
end // always @ (posedge i_Clk or negedge i_Rst_L)
```

SPI CLK - Simulation







DATA In - Stored

- Data In Register when Data Valid is HIGH

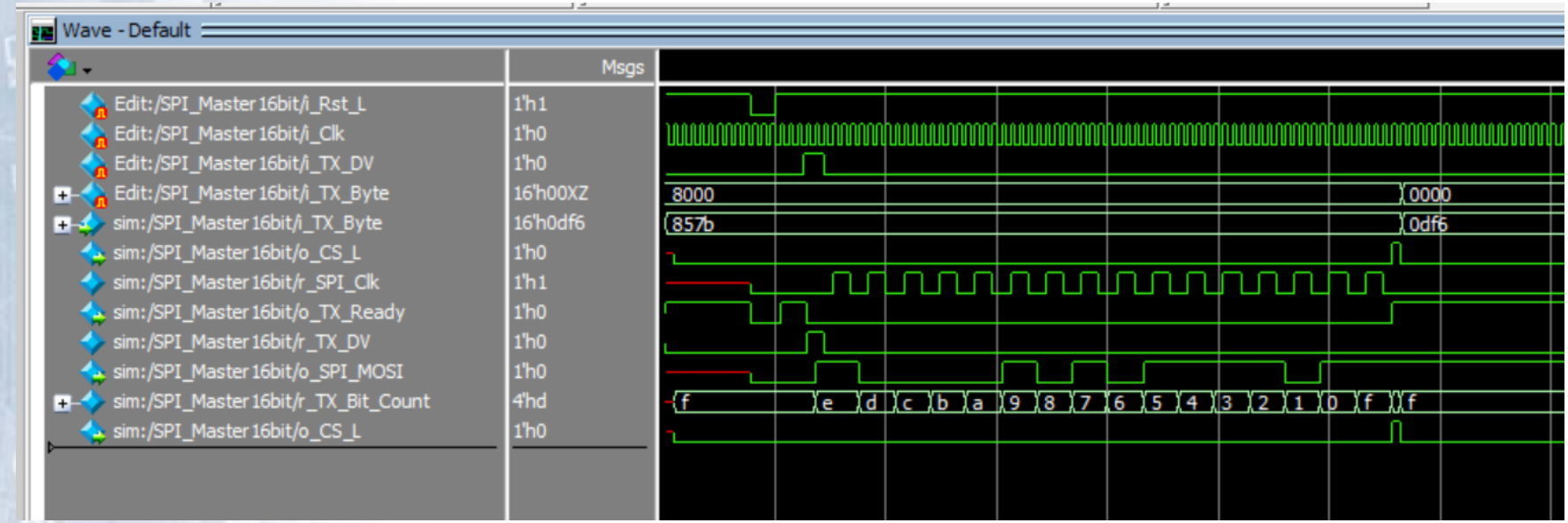
```
145   // Purpose: Register i_TX_Byte when Data Valid is pulsed.
146   // Keeps local storage of byte in case higher level module changes the data
147   always @(posedge i_Clk or negedge i_Rst_L)
148   begin
149     if (~i_Rst_L)
150     begin
151       r_TX_Byte <= 16'h0000;
152       r_TX_DV   <= 1'b0;
153     end
154     else
155       begin
156         r_TX_DV <= i_TX_DV; // 1 clock cycle delay
157         if (i_TX_DV)
158         begin
159           r_TX_Byte <= i_TX_Byte;
160         end
161       end // else: !if(~i_Rst_L)
162   end // always @ (posedge i_Clk or negedge i_Rst_L)
```

```
// CPOL: Clock Polarity
// CPOL=0 means clock idles at 0, leading edge is rising edge.
// CPOL=1 means clock idles at 1, leading edge is falling edge.
assign w_CPOL  = (SPI_MODE == 2) | (SPI_MODE == 3);

// CPHA: Clock Phase
// CPHA=0 means the "out" side changes the data on trailing edge of clock
//              the "in" side captures data on leading edge of clock
// CPHA=1 means the "out" side changes the data on leading edge of clock
//              the "in" side captures data on the trailing edge of clock
assign w_CPHA  = (SPI_MODE == 1) | (SPI_MODE == 3);
```

Generate SPI MOSI Data

```
164
165   // Purpose: Generate MOSI data
166   // Works with both CPHA=0 and CPHA=1
167   always @(posedge i_Clk or negedge i_Rst_L)
168   begin
169     if (~i_Rst_L)
170     begin
171       o_SPI_MOSI     <= 1'b0;
172       r_TX_Bit_Count <= 4'b1111; // send MSb first
173     end
174     else
175     begin
176       // o_SPI_MOSI <= 1'b0;   /// aggiunto io per pulizia (DEFAULT)
177       // If ready is high, reset bit counts to default
178       if (o_TX_Ready)
179       begin
180         r_TX_Bit_Count <= 4'b1111;
181         o_CS_L <= 1'b0;
182       end
183       // Catch the case where we start transaction and CPHA = 0
184       else if (r_TX_DV & ~w_CPHA)
185       begin
186         o_SPI_MOSI     <= r_TX_Byte[4'b1111];
187         r_TX_Bit_Count <= 4'b1110;
188       end
189       else if ((r_Leading_Edge & w_CPHA) | (r_Trailing_Edge & ~w_CPHA))
190       begin
191         o_SPI_MOSI     <= r_TX_Byte[r_TX_Bit_Count];
192         r_TX_Bit_Count <= r_TX_Bit_Count - 1'b1;
193         if (r_TX_Bit_Count == 4'b1111)
194             o_CS_L <= 1'b1;
195       end
196     end
197   end
198
```

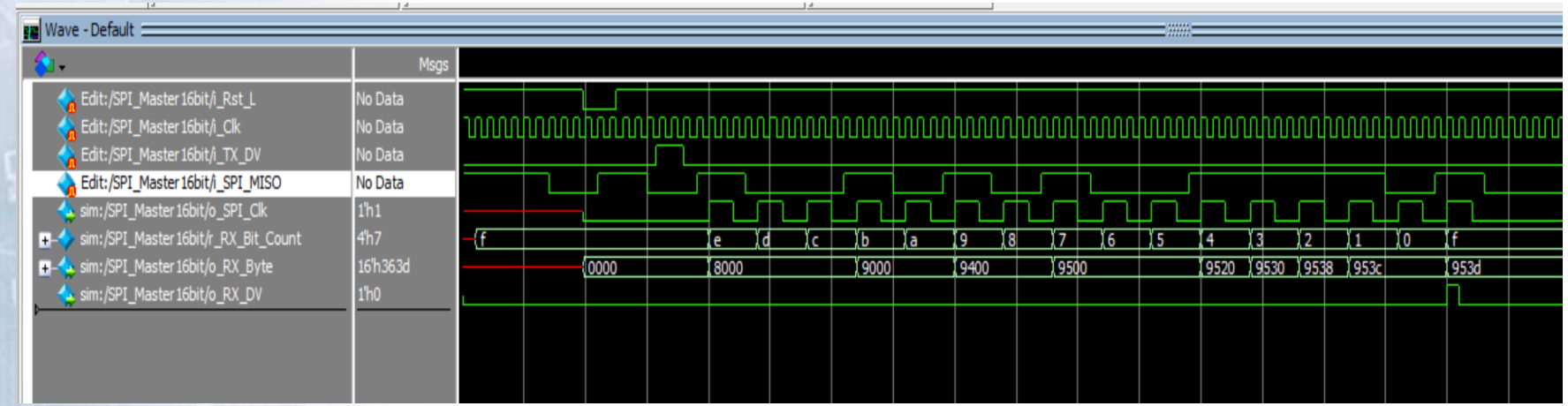
Generate SPI MOSI Simulation



Read MISO Data

```
// Purpose: Read in MISO data.
always @(posedge i_Clk or negedge i_Rst_L)
begin
  if (~i_Rst_L)
  begin
    o_RX_Byte      <= 16'h0000;
    o_RX_DV        <= 1'b0;
    r_RX_Bit_Count <= 4'b1111;
  end
  else
  begin

    // Default Assignments
    o_RX_DV   <= 1'b0;

    if (o_TX_Ready) // Check if ready is high, if so reset bit count to default
    begin
      r_RX_Bit_Count <= 4'b1111;
    end
    else if ((r_Leading_Edge & ~w_CPHA) | (r_Trailing_Edge & w_CPHA))
    begin
      o_RX_Byte[r_RX_Bit_Count] <= i_SPI_MISO;  // Sample data
      r_RX_Bit_Count            <= r_RX_Bit_Count - 1'b1;
      if (r_RX_Bit_Count == 4'b0000)
      begin
        o_RX_DV   <= 1'b1;   // Byte done, pulse Data Valid
      end
    end
  end
end
```

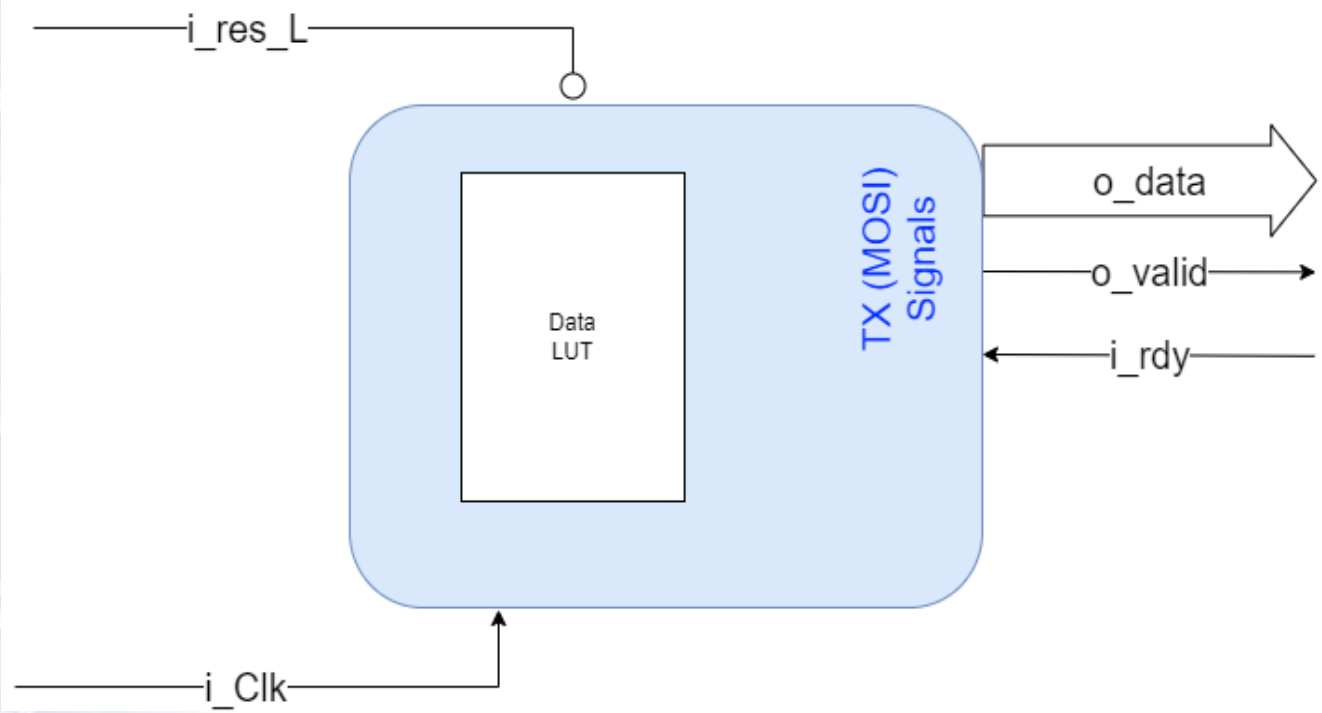
Read MISO - Simulation



Nota: se **non** si attiva i_TX_DV non viene generato SPI_Clk e quindi il dispositivo non è in grado di scrivere su SPI ma al contempo nemmeno di leggere da SPI

Data Generator

Sistema con una memoria interna (LUT) che fornisca sequenzialmente i dati con cui «alimentare» il generatore SPI



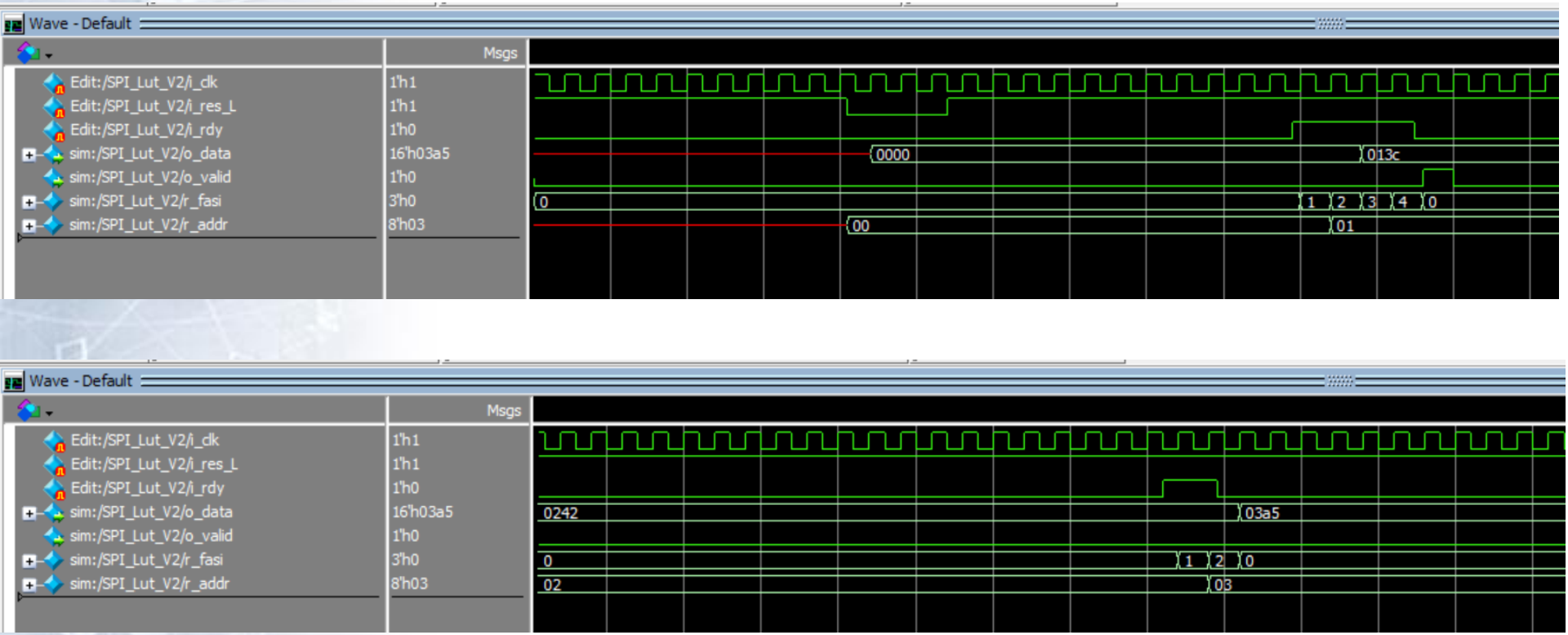
Data Generator - temporizzazioni

```
 1  `define ROM_SIZE 6'd15
 2
 3  module SPI_Lut_V2 (i_clk,i_res_L,i_rdy,o_data,o_valid );
 4    input i_clk,i_res_L,i_rdy;
 5    output reg [15:0] o_data;
 6    output reg o_valid;
 7
 8    reg [2:0] r_fasi;
 9    reg [7:0] r_addr;
10    reg [7:0] ROM[`ROM_SIZE:0];
11
12  // contatore delle fasi
13  always @(posedge i_clk or negedge i_res_L) begin
14    if (!i_res_L)
15      r_fasi<=3'b000;
16    else if (!i_rdy)
17      r_fasi<=3'b000;
18    else if ((i_rdy)&(r_fasi < 3'b111))
19      r_fasi <= r_fasi + 3'b001;
20  end
21
22  // generazione segnale "o_valid"
23  always @(posedge i_clk or negedge i_res_L) begin
24   if (!i_res_L)
25      o_valid<=0;
26   else if ((r_fasi==3'b100))
27      o_valid<=1;
28   else
29      o_valid<=0;
30  end
```

Data Generator - dati

```
// incremento dell'indirizzo
always @(posedge i_clk or negedge i_res_L) begin
 if (!i_res_L)
    r_addr = 0;
 else if (r_fasi==3'b001)
    r_addr=r_addr+1;
 else if (r_addr > `ROM_SIZE)
    r_addr = 0;
end

//dato in uscita
always @(posedge i_clk)
   begin
      ROM[0]= 8'h00; // Dummy
      ROM[1]= 8'h3C; // Digit 1
      ROM[2]= 8'h42; // Digit 2
      ROM[3]= 8'hA5; // Digit 3

      ROM[4]= 8'h81; // Digit 4
      ROM[5]= 8'hA5; // Digit 5
      ROM[6]= 8'h99; // Digit 6
      ROM[7]= 8'h42; // Digit 7

      ROM[8]= 8'h3c; // Digit 8
      ROM[9]= 8'h00; // Decode 9
      ROM[10]=8'h03; // Intensity 10
      ROM[11]=8'h07; // Scan 11

      ROM[12]=8'h01; // Shutdown/Normal 12
      ROM[13]=8'h01; // Dummy
      ROM[14]=8'h01; // Dummy
      ROM[15]=8'h00; // Test/Normal
      o_data={r_addr[7:0],ROM[r_addr]};
        end
endmodule
```

Data Generator - simulazioni



Nota: *i_rdy* deve rimanere alto per un certo tempo altrimenti non si genera *o_valid*
Strategia: i_rdy viene generato (dal blocco SPI) come conseguenza del fatto che o_valid si sia attivato.

MAX7219

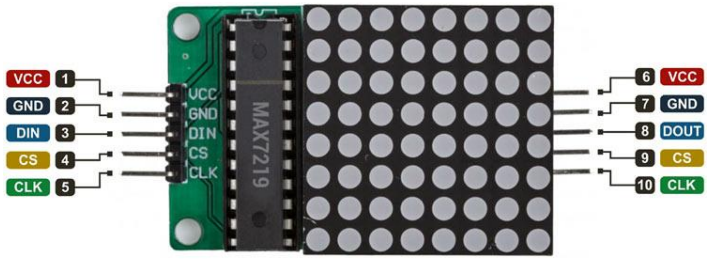


Table 1. Serial-Data Format (16 Bits)

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
X	X	X	X	ADDRESS				MSB	DATA						LSB

Table 2. Register Address Map

REGISTER	ADDRESS					HEX CODE
	D15–D12	D11	D10	D9	D8	
No-Op	X	0	0	0	0	0xX0
Digit 0	X	0	0	0	1	0xX1
Digit 1	X	0	0	1	0	0xX2
Digit 2	X	0	0	1	1	0xX3
Digit 3	X	0	1	0	0	0xX4
Digit 4	X	0	1	0	1	0xX5
Digit 5	X	0	1	1	0	0xX6
Digit 6	X	0	1	1	1	0xX7
Digit 7	X	1	0	0	0	0xX8
Decode Mode	X	1	0	0	1	0xX9
Intensity	X	1	0	1	0	0xXA
Scan Limit	X	1	0	1	1	0xXB
Shutdown	X	1	1	0	0	0xXC
Display Test	X	1	1	1	1	0xXF

Table 3. Shutdown Register Format (Address (Hex) = 0xXC)

MODE	ADDRESS CODE (HEX)	REGISTER DATA							
		D7	D6	D5	D4	D3	D2	D1	D0
Shutdown Mode	0xXC	X	X	X	X	X	X	X	X
Normal Operation	0xXC	X	X	X	X	X	X	X	1

Table 10. Display-Test Register Format (Address (Hex) = 0xXF)

MODE	REGISTER DATA							
	D7	D6	D5	D4	D3	D2	D1	D0
Normal Operation	X	X	X	X	X	X	X	0
Display Test Mode	X	X	X	X	X	X	X	1

Simulazione sistema completo