SPI

Synchronous type / Full duplex

Master device generate the Clock for the Sender, which can determines when data can change & when it's ready for Reading.

Four Wire Interface

Serial Clock can go upto 1/2 of System clock, typical it is ——— 1/4 ———

AGENDA : # Implementation of Simple SPI interface (SPI Mode 0)

——— ——— . with different Modes (0, 1, 2, 3).

Real Project : PMOD DA4

Daisy Chain Configuration

Bit Banging

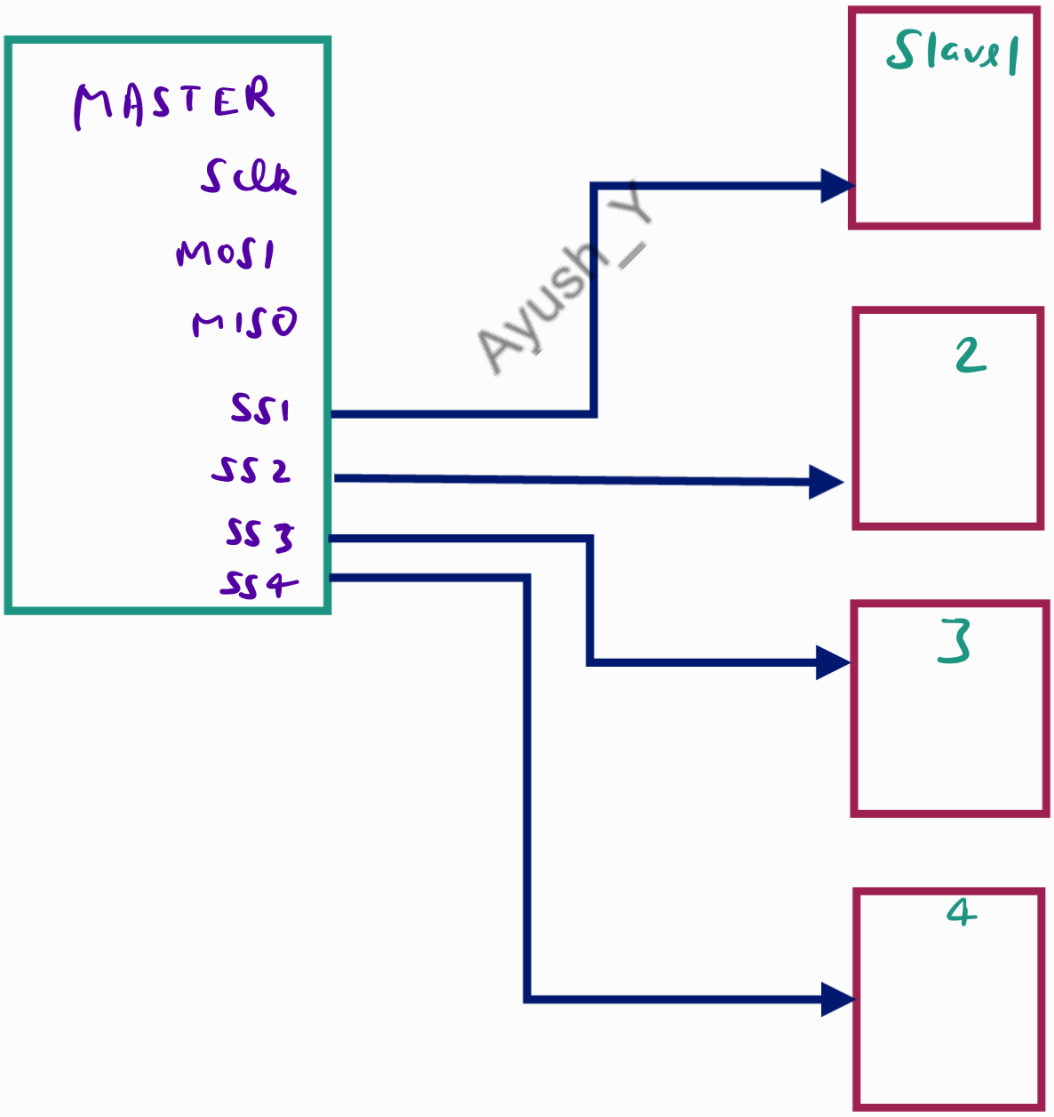
Understanding SPI in Detail

SCLK → Serial Clock

MOSI → Master Out Slave IN

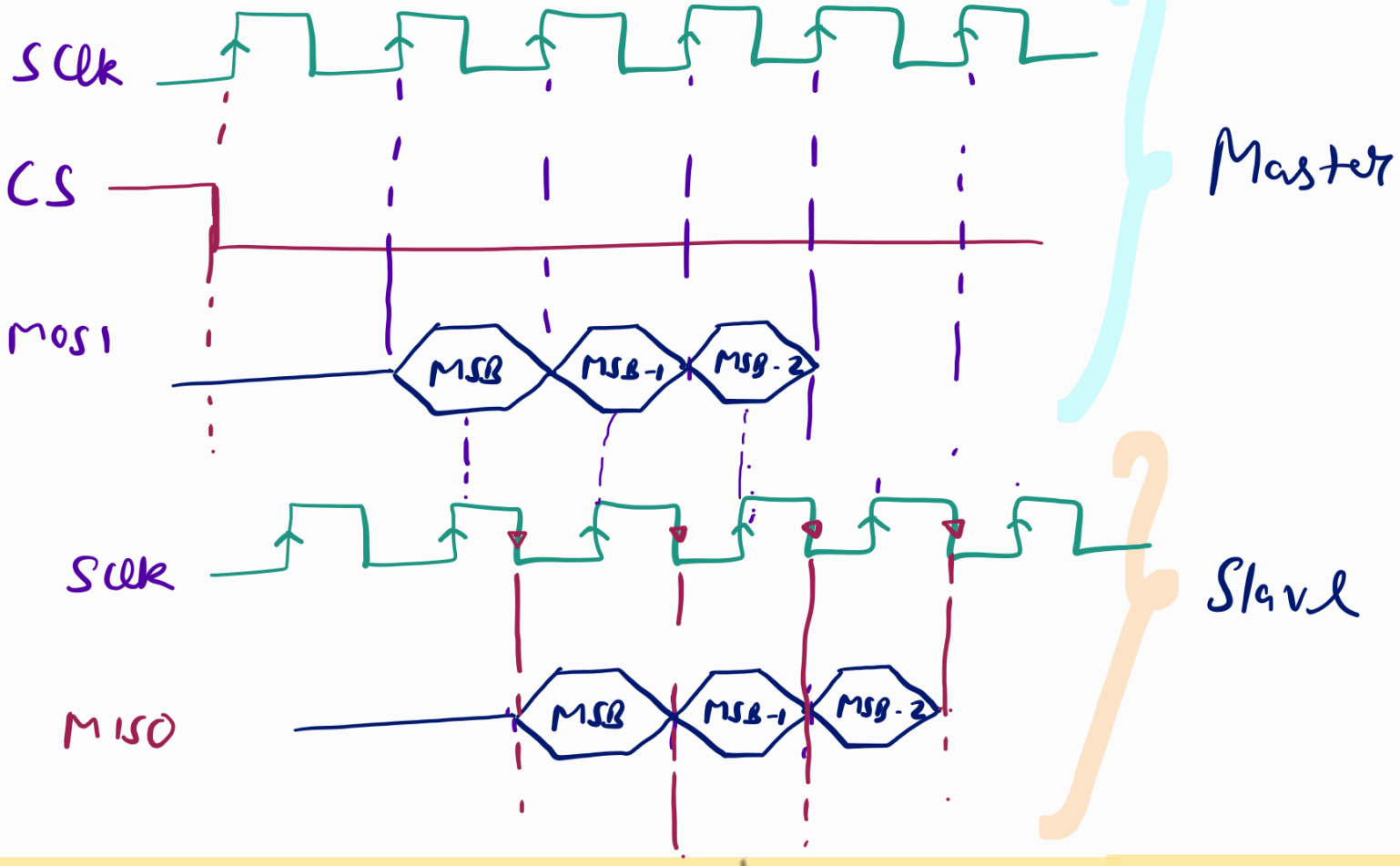
MISO → " IN " OUT

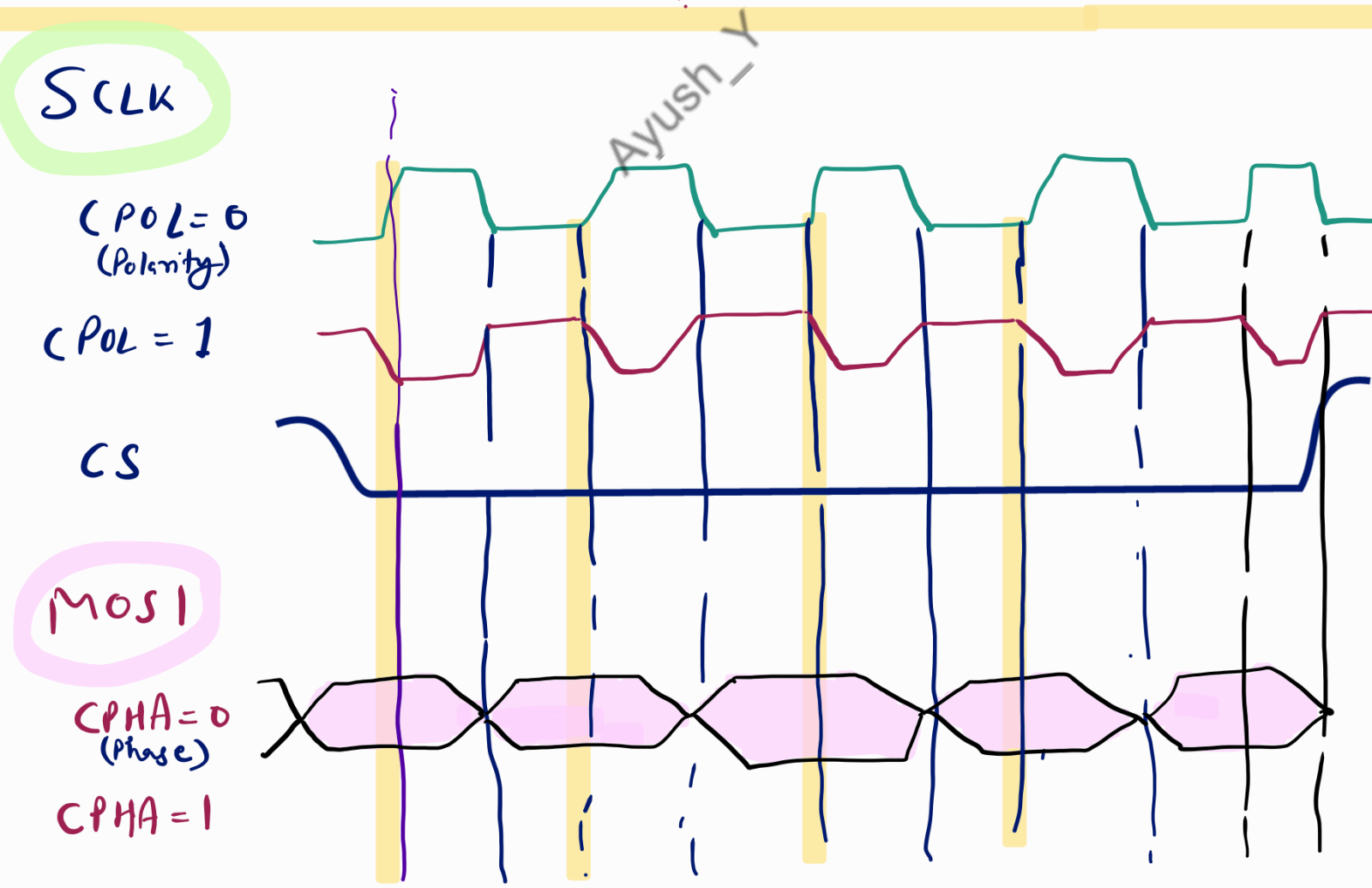
SS → Chip Select / Slave Select



Ayush_Y

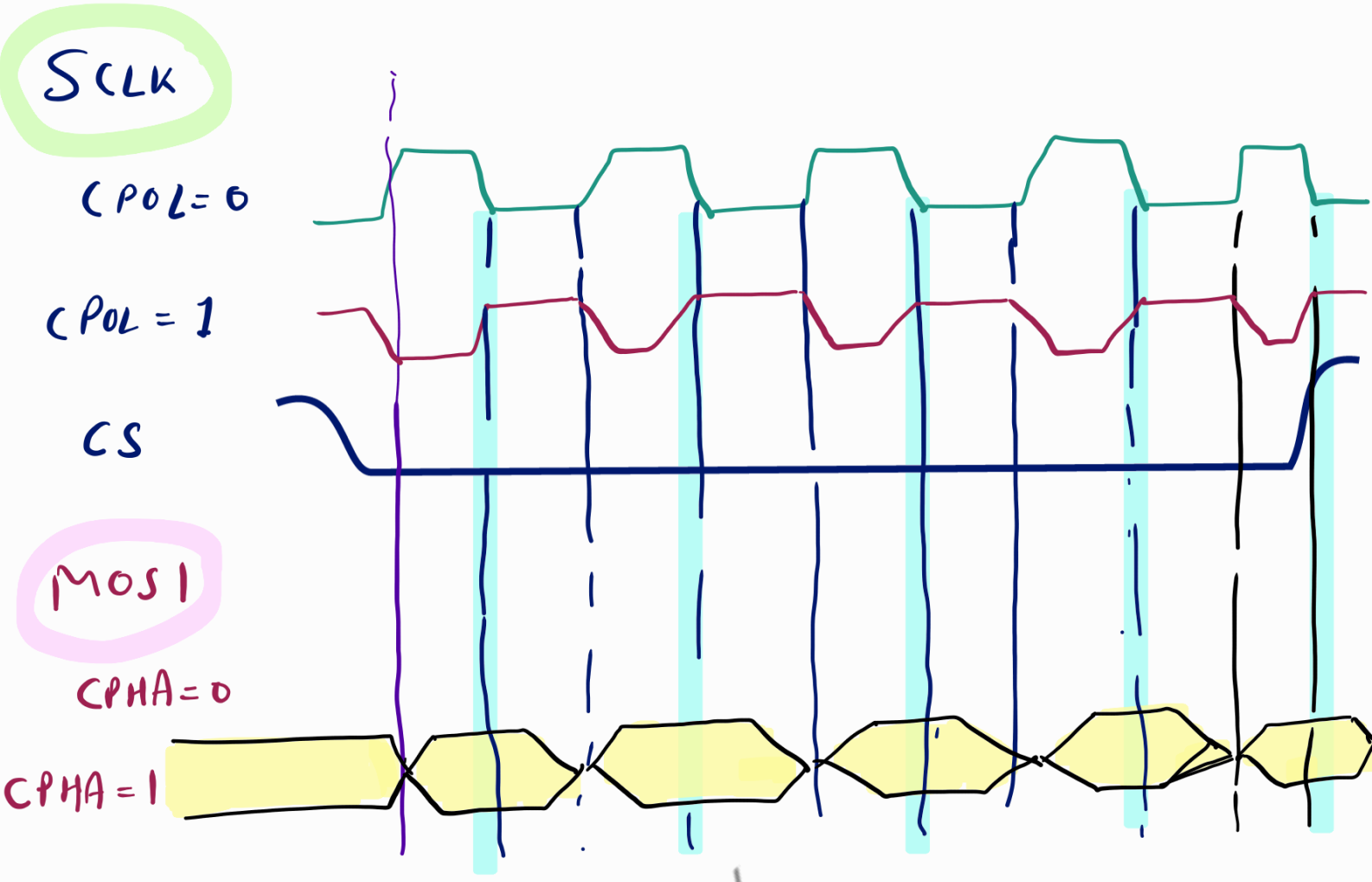
Waveforms





Ayush_Y

Just drawing CPHA = 0 with +ve edge of clk data bit middle Part will come

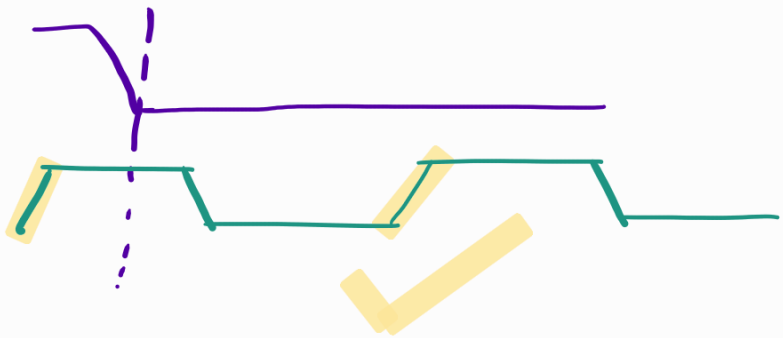


Just drawing CPHA = 1 with -ve edge of clk
data bit middle Part will come

Ayush Y

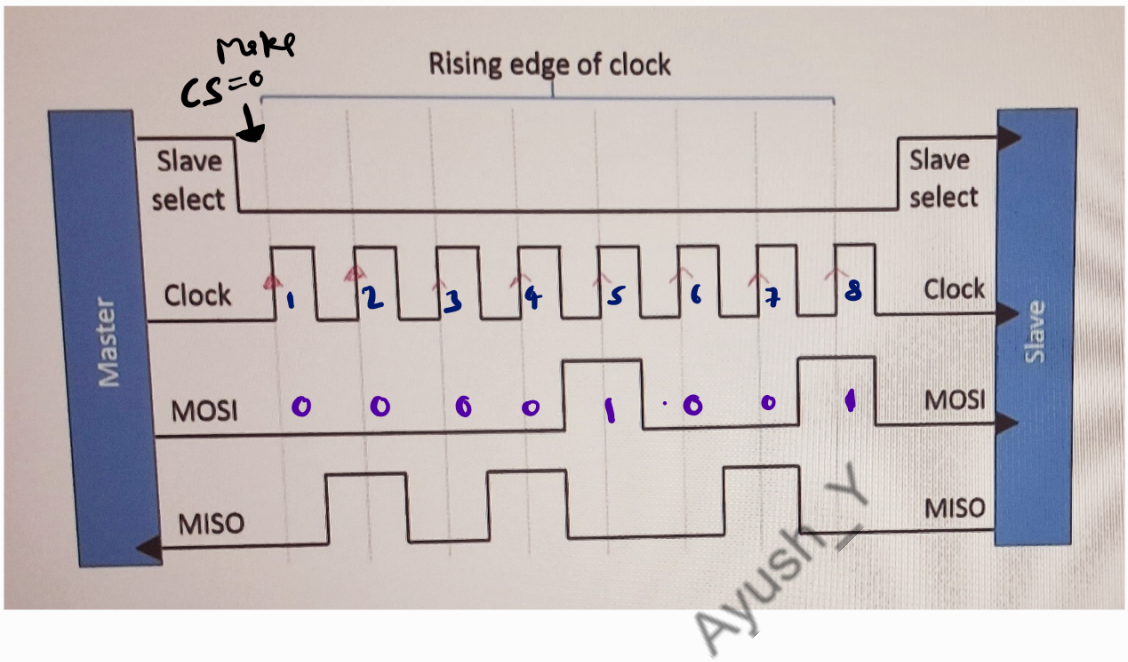
CPOL	CPHA		SCLK	edge
0	0	Mode 0	0	(Posedge) 1st
0	1	Mode 1	0	(negedge) 2nd
1	0	Mode 2	1	(negedge) 1st
1	1	Mode 3	1	(posedge) 2nd

When CS = 0
SCLK = 1
↓
remains stable in its Normal Value



Keep CS = 0 for a small duration & make SCLK = 1 & after next SCLK pulse send the data Now (not for the just first Clock Pulse).

But this thing isn't Mandatory.



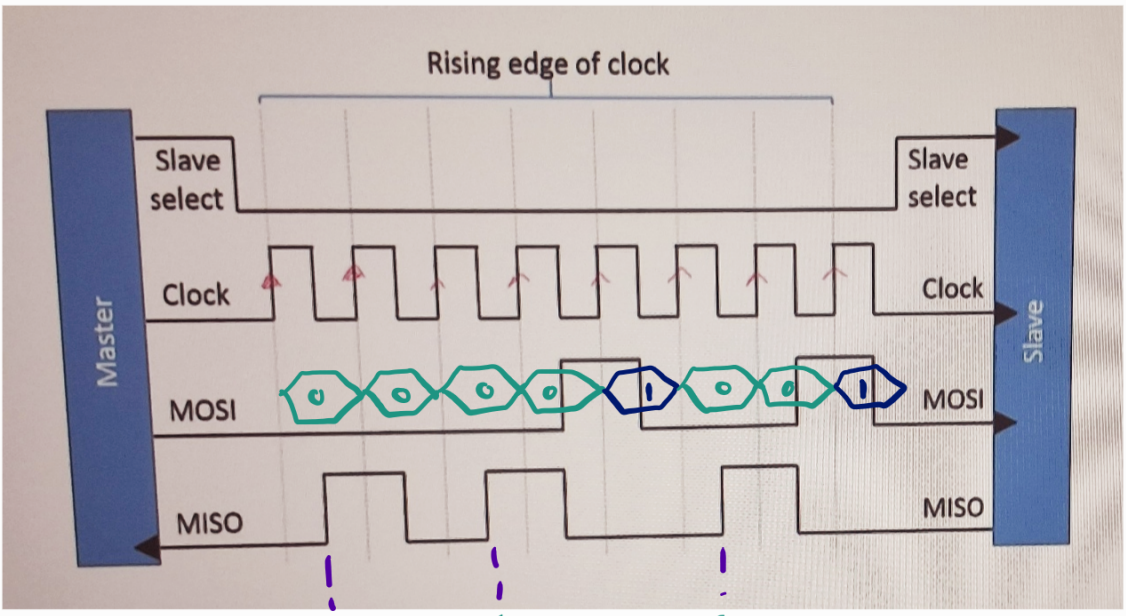
→ > Pos edge

Let

All new data will be transfered from Master to Slave @ posedge.

tx_en = 1 → $\overline{CS}$ → Transmit 8-bit data

& again make CS = 1.



Sampling in the Middle - - - - — —

If Master is sending Data @ +ve edge of Sclk, then Slave can sample @ middle edge of MOSI.

tx_en ✓

↓

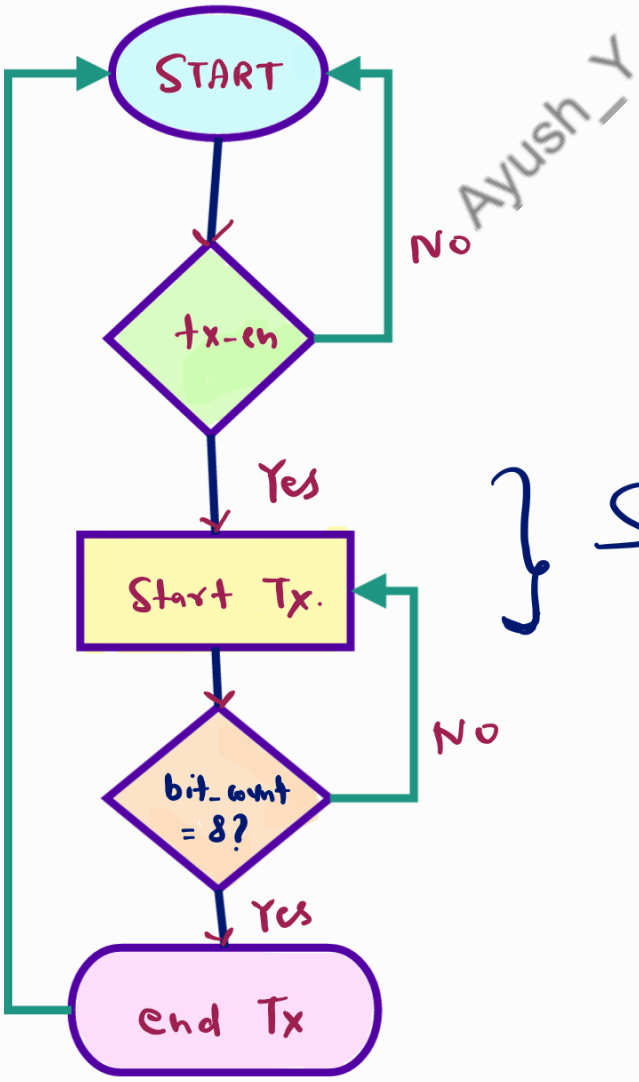
CS=0 (1 Sclk)

↓

8-bit of data (8 cycles of Sclk)

↓

CS=1 (1 sclk)



Ayush_Y

} SPI Transaction

Sclk = $\frac{1}{8}$ clk

Breaking the code in Detail

SPI Master Code

Module fsm-spi → wire clk, rst, tx-ena

→ reg mosi, cs

wire sclk;

```
using typedef enum logic [1:0] {
                                } state-type;

State type  State, next-state;
```

Intermediate : reg

```
[7:0] din, spi_clk,
[2:0] count, ccount;
integer bit-count = 0;
```

Ayush Y

```
always @ (posedge clk) // Generating SPI clk
begin

Case (next-state)
    idle :
    begin
     spi-sclk <= 0;
    end

Start-tx:  begin
if(count < 3'b011 ||
   count == 3'b111)
  spi-sclk <= 1'b1;
      else <=0;
end
```

```
tx_data: begin
if (count < 3'b011 ||
    count == 3'b111)
spi_sclk <= 1;
else __ 0
end
```

```
end-tx: begin
if ( count < 3'b011 ||
     count == 3'b111 )
spi-sclk <= 1'b1;
else__0;
end
```

```
default: spi_sclk <= 1'b0;
  endcase
end
```

```
always @ (posedge clk)        // General Logic
  begin
    if(rst)
        state <= idle;
    else
        state <= next- state;
  end
```

Ayush_Y

NEXT STATE DECODER

```
always @ (*)        begin        // Combinational Always Block
case ( State )
```

```
idle:     begin
  mosi = 1'b0;
  cs = 1'b1;
  if (tx_enable)
     next-state = start-tx;
  else
     next _____ = idle
end
```

```
start_tx:     begin
    cs = 1'b0;
  if (count == 3'b111)
  next-state = tx-data;
  else
  next-state = start-tx;
end
```

```
tx-data:    begin
mosi = din [7 - bit_count];
if (bit-count =! 8) begin
        state = tx-data;
end
else begin
        state = end-tx;
end     mosi = 1'b0;
end
```

```
end-tx :    begin
    CS = 1'b1;
    mosi = 1'b0;
  if ( count == 3'b111 )
        next-state = idle;
  else
        next-state = end-tx;
end
```

COUNTER ALWAYS BLOCK

```
always @ (posedge clk)
  begin
        Case (state)
```

```
idle:
    begin
      Count <=0;
      bit_count <=0;
end
```

```
start-tx:
      count <= count +1;
```

```
end-tx:
      begin
        Count <= count+1;
      bit-count <=0;
end
```

```
tx-data:  begin
if (bit-count =! 8) begin
        if(count < 3'b111)
            Count <= count +1;
else begin
        count <=0;
        bit-count <= bit-count+1;
end
end
```

```
default:  begin
    Count <=0;
    bit-count <=0;
end
```

```
endcase   end
```

Ayush_Y

```
assign    sclk = spi_sclk;
endmodule
```

Testbench Overview:

```
module tb;
reg  clk = 0;
     rst = 0;
     tx_enable = 0;

wire mosi;
     ss;
     sclk;

always @ #5 clk = ~clk;

initial begin
  rst = 1;
  repeat (5) @ (posedge clk);
  rst = 0;
end

initial begin
  tx_enable = 0;
  repeat (5) @ (posedge clk);
  tx_enable = 1;
end

fsm_spi dut ( clk, rst, tx_enable, mosi, ss, sclk );
endmodule
```

Ayush_Y