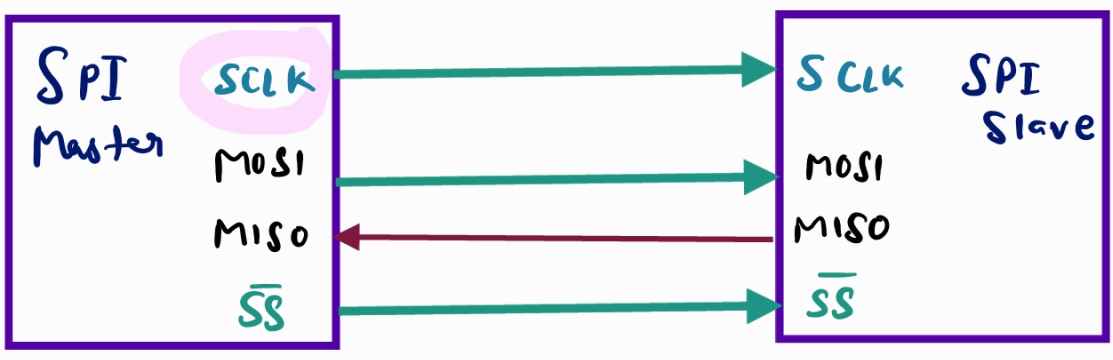
SPI

MOTOROLA

Synchronous type / Full duplex

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



Four wire Interface

Serial clock can go upto 1/2 of System clock, MAX ↗ 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

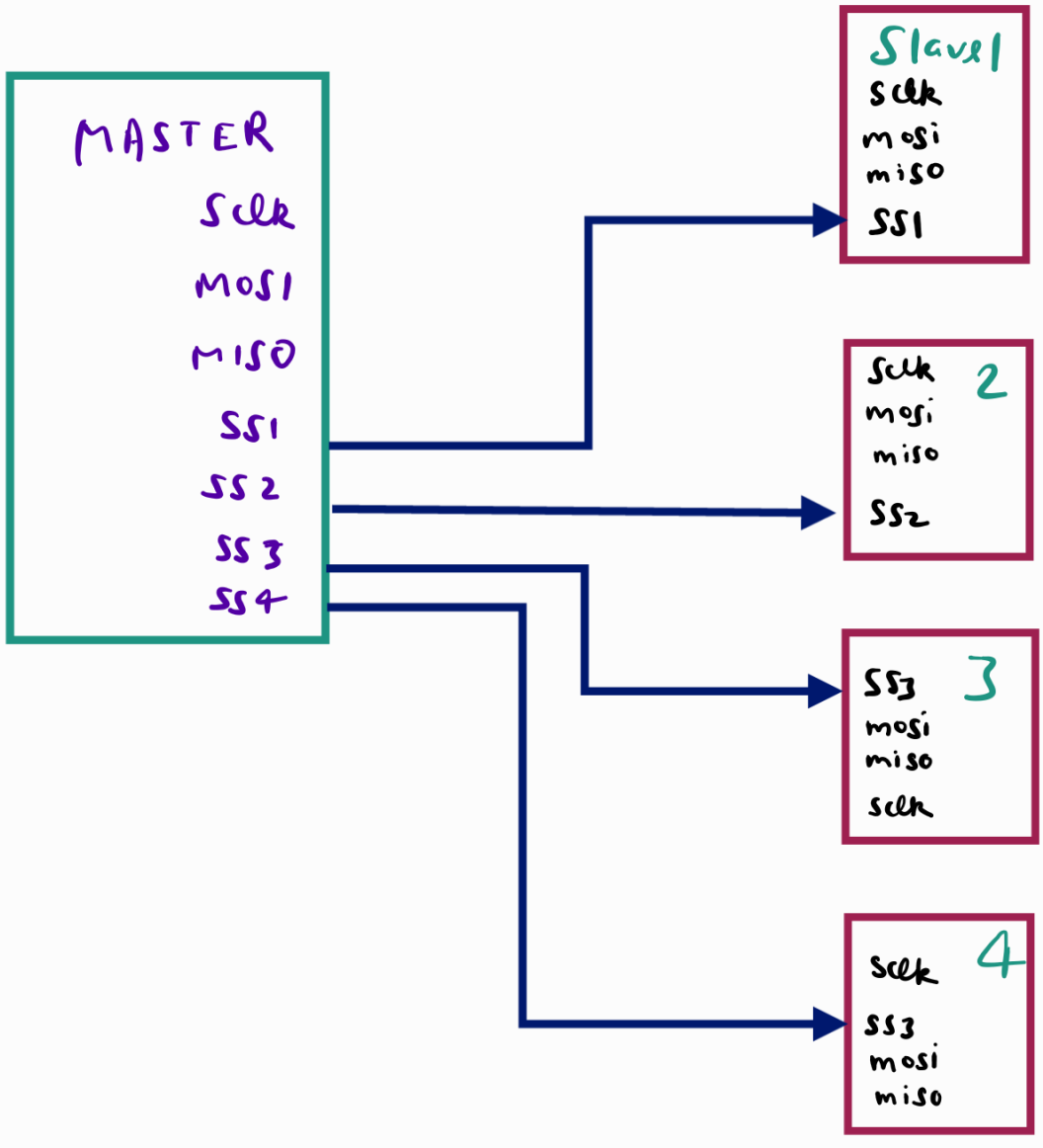
Understanding SPI in Detail

SCLK → Serial Clock

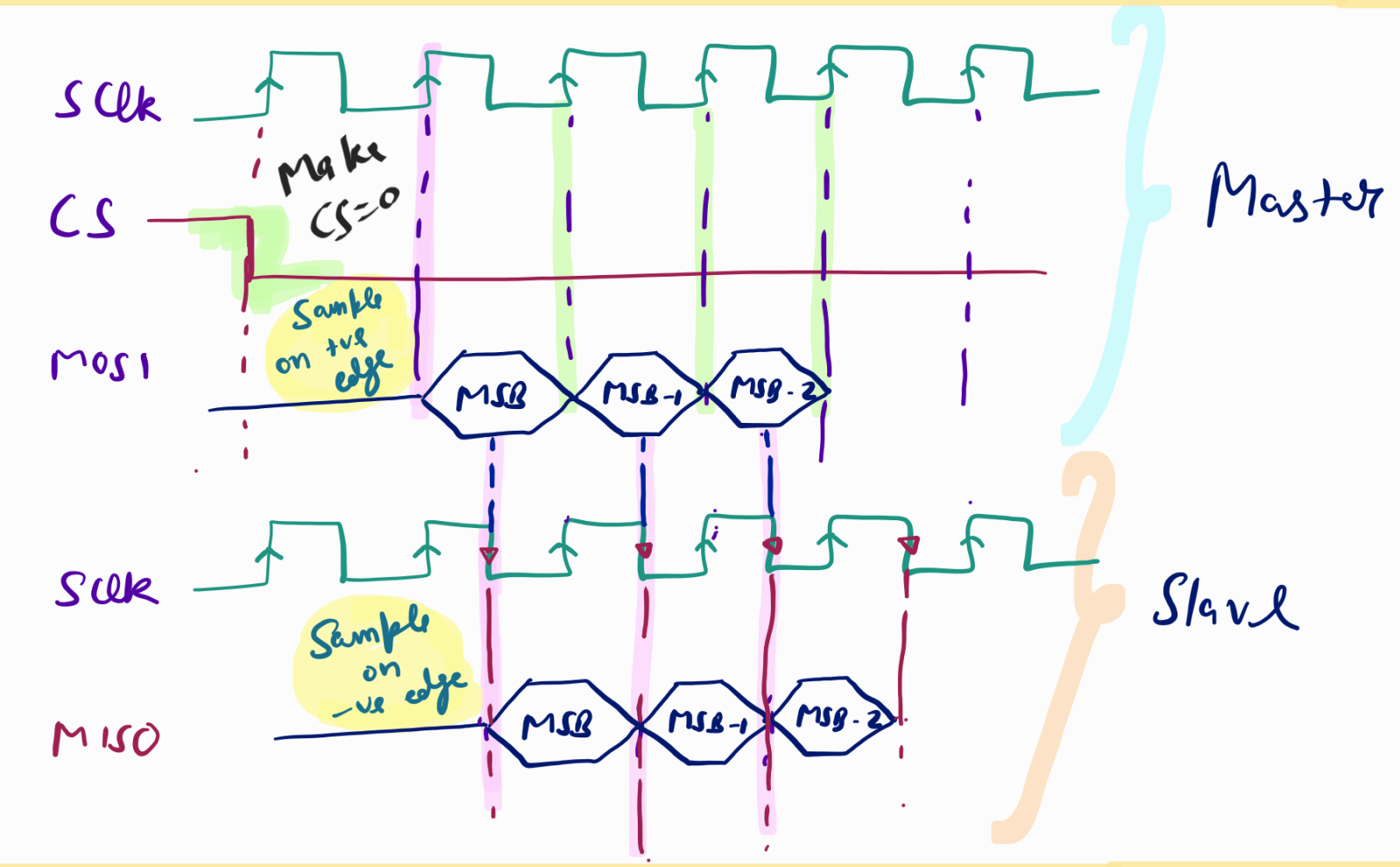
MOSI → Master Out Slave IN

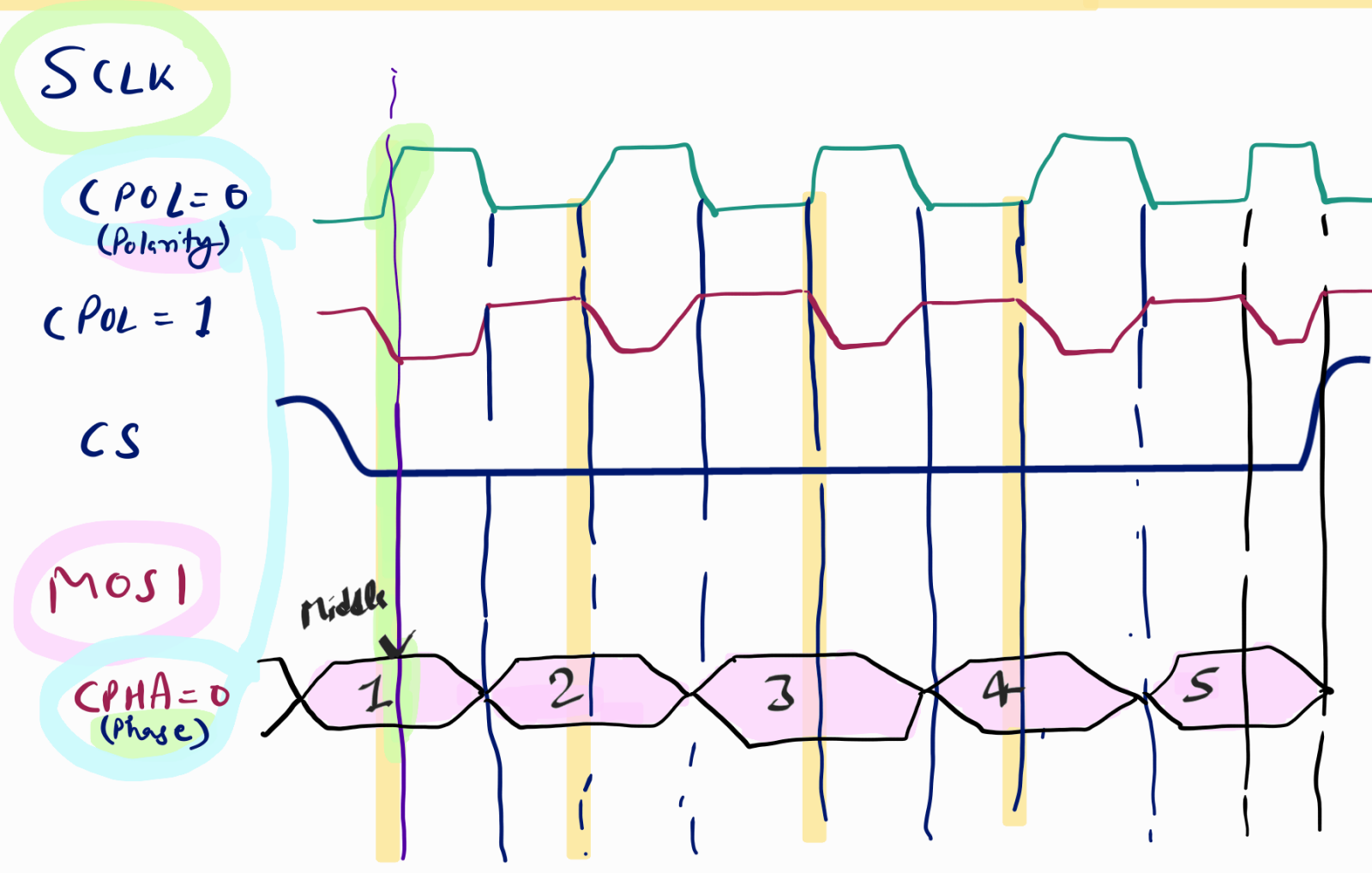
MISO ← " IN " OUT

SS → Chip Select / Slave Select

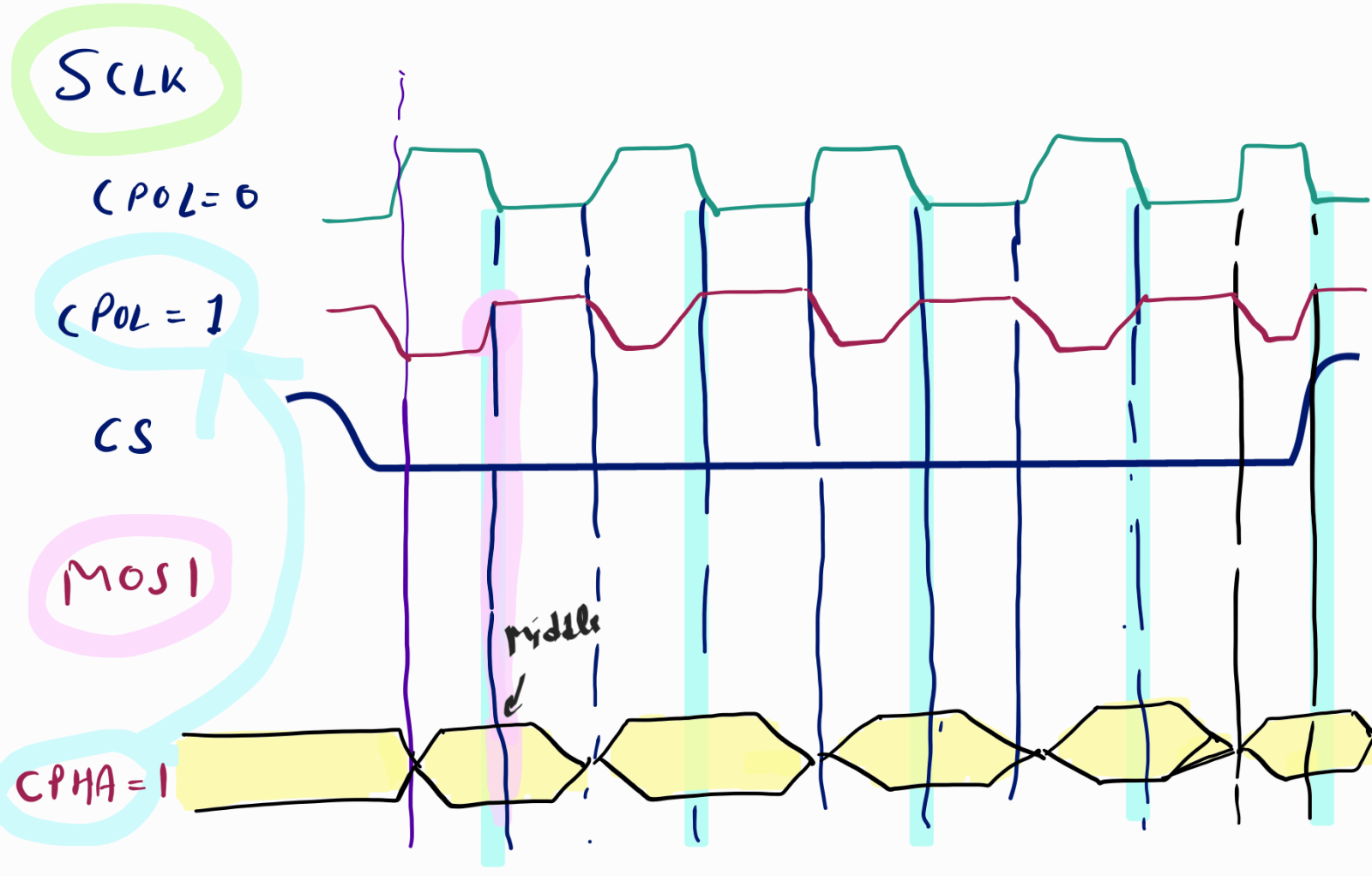


Waveforms





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

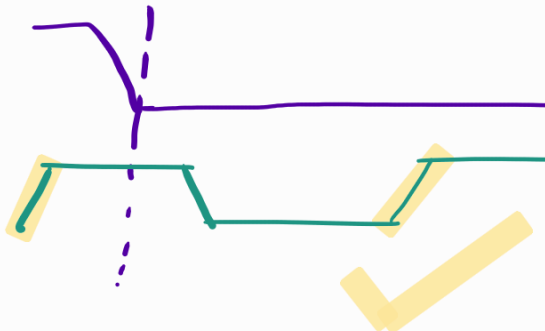


Just drawing CPHA = 1 with -vl edge of clk data bit middle Part will come

SCLK CPOL	CPHA	MOSI
0	0	Mode 0
0	1	Mode 1
1	0	Mode 2
1	1	Mode 3

Sclk	edge
0	(Posedge) 1st
0	(negedge) 2nd
1	(negedge) 1st
1	(posedge) 2nd

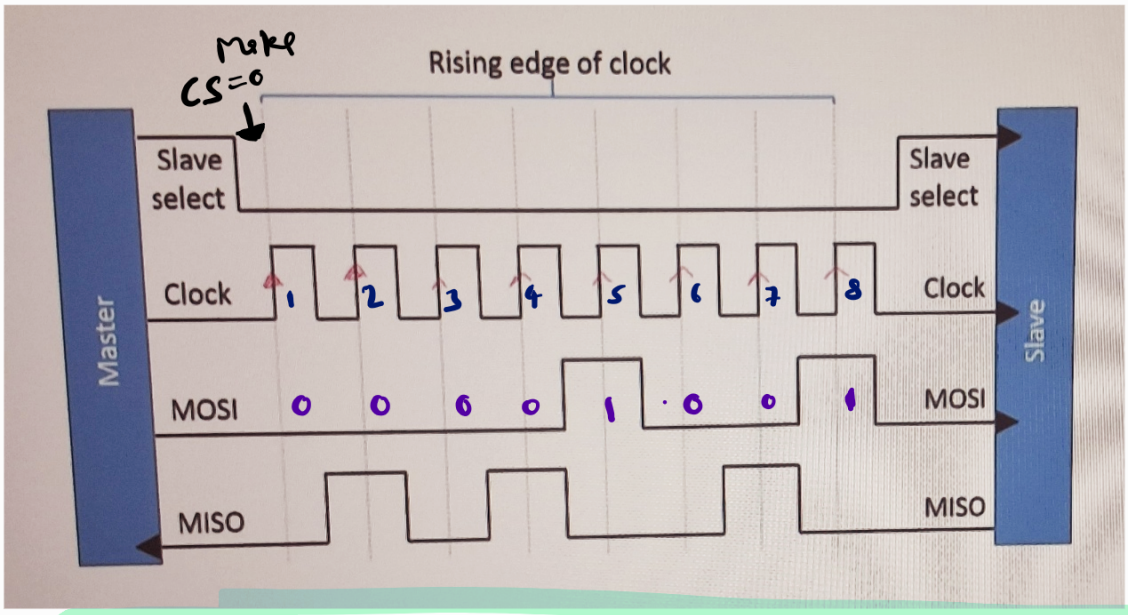
When CS = 0
SCLK = 1
↓
remains stable in its Normal Value



from next start sending

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).

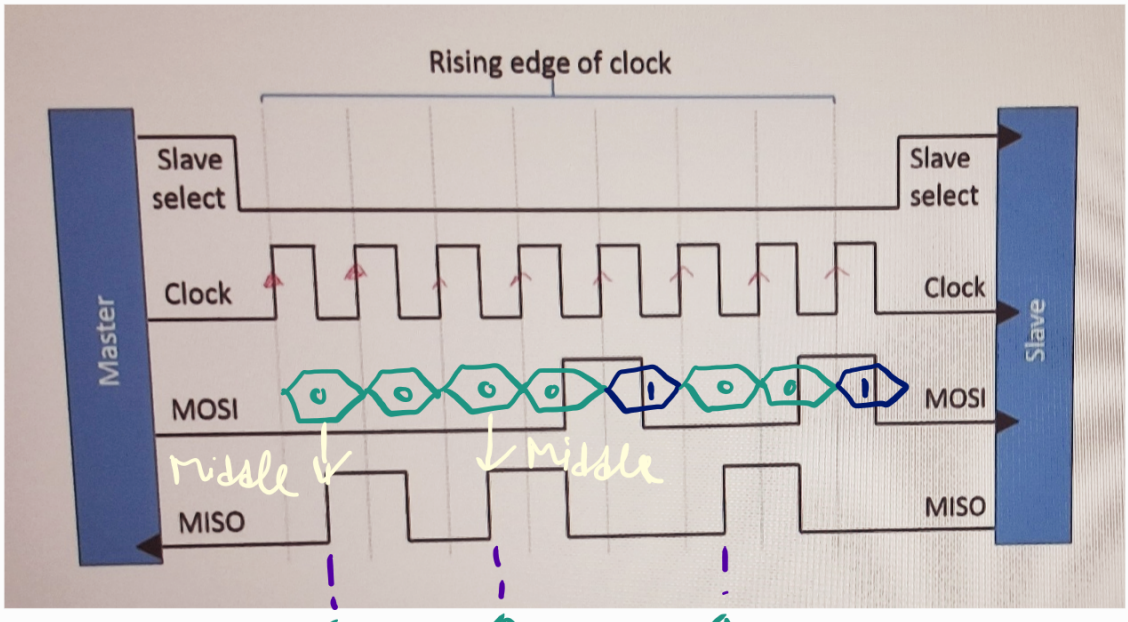
But this thing is n't Mandatory.



Let → > Pos edge ↓ All new data will be transfered from Master to Slave @ posedge.

We will use sending data on 1st clk only.

tx_en=1 → $\overline{CS}$ → Transmit 8-bit data & again make CS=1.



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

tx_en ✓

↓

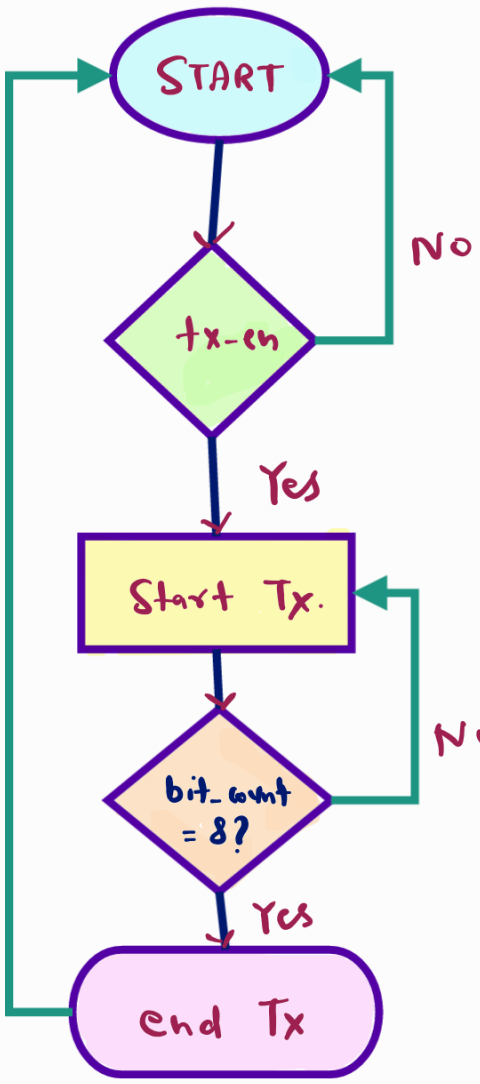
CS=0 (1 Sclk)

↓

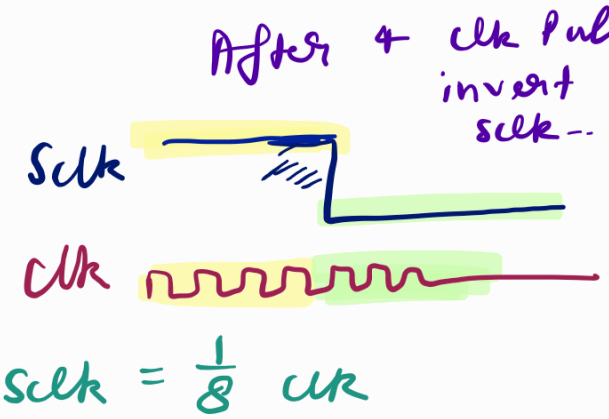
8-bit of data (8 cycles of Sclk)

↓

CS=1 (1 sclk)



} SPI Transaction



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

Breaking the code in Detail

SPI Master Code

Module
spi_master_cyuth

wire clk, rst, tx_ena
reg masi, 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;

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 c=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
```

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;
```

```
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
```

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

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

```
endcase    end ___ ___.
```

```
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
```

```
spi_master_ayush dut( clk, rst, tx_enable, mosi, SS, sclk);
endmodule
```

Slave Module

```
module spi_slave_ayush(
input → sclk
       → cs,
       → mosi
Output → [7:0] dout
       → reg done
```

Intermediate ⟶ integer count = 0;

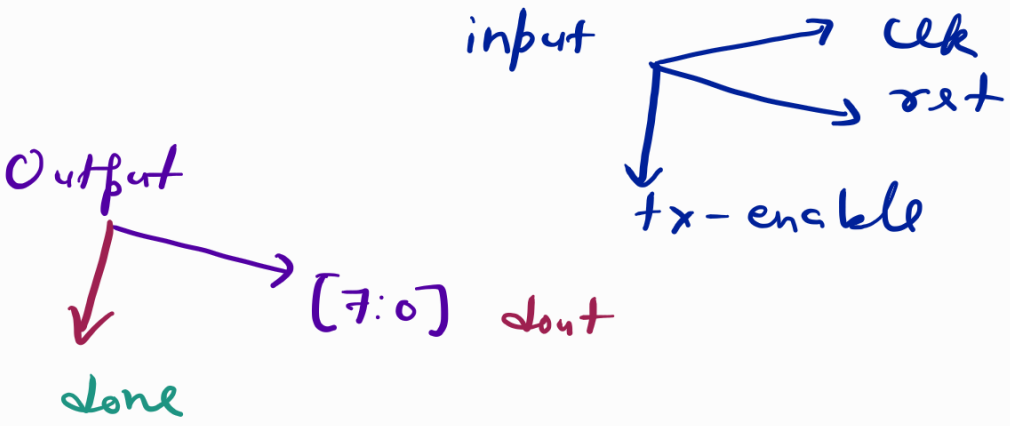
Our Beloved Typedef enum Logic

```
typedef enum [1:0] { idle = 0, sample = 1 } state_type;

State-type state;
reg [7:0] data = 0;   // One more intermediate
```

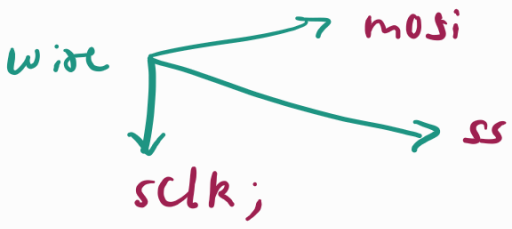
```
always @ (negedge sclk) begin
  case (state)

    idle : begin
      done <= 1'b0;
      if (cs = 1'b0;)
        state <= sample
      else
        state <= idle
    end

    default :
        State <= idle;

    Sample : begin
      if (count < 8) begin
        count <= count + 1;
        data <= {data [6:0], mosi};
        state <= Sample
      end
      else begin
        count <= 0 ;
        State <= idle ;
        done <= 1 ;
      end
    end

  endcase
end
      assign dout = data;
endmodule
```

TOP MODULE

module top-ayush -SPI (



// Intermediate



spi_master_ayush spi_m (clk, rst, tx-enable, mosi, ss, sclk);

spi_slave_ayush spi_s (sclk, mosi, ss, dout, done);

endmodule

Top Testbench

module tb;

reg clk;
" rst;
" tx-enable;

wire [7:0] dout;
" done;

always #5 clk = ~clk;

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

```
initial begin
  tx_enable = 0;
repeat(5)@( ——— );
  tx_enable = 1;
end
```

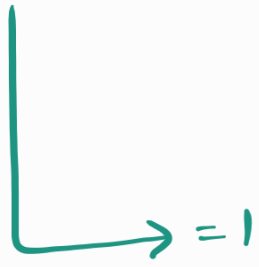
```
top_spi_synth dut ( clk, rst, tx_enable, dout);
endmodule
```

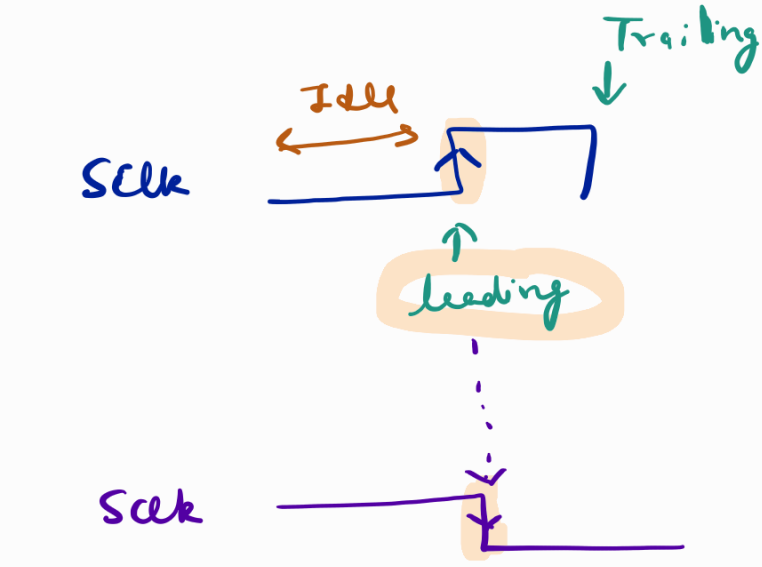
Alternate Method:

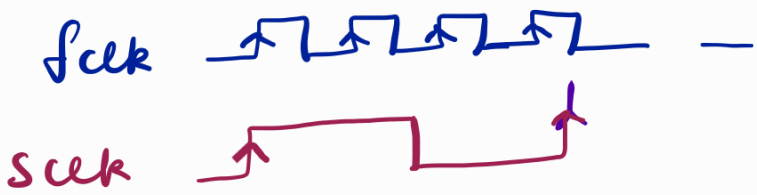
Diff. Mode Implementation

Polarity of clk

CPOL = 0



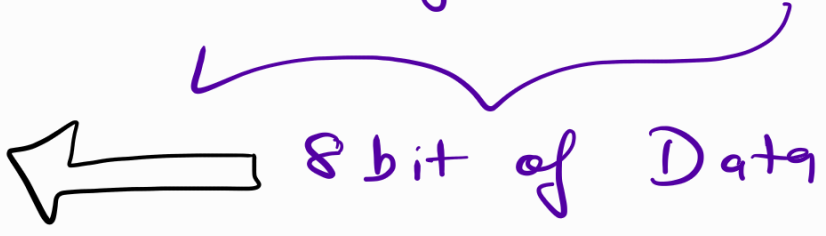




$SClk = \frac{fclk}{4}$

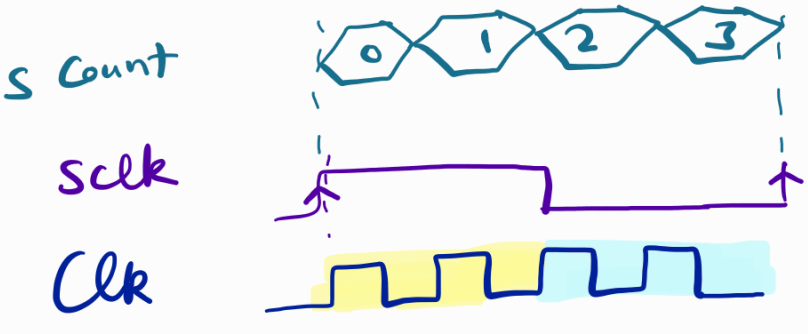
Half-clk-P = 2 edges

counter goes from 0 to 3.

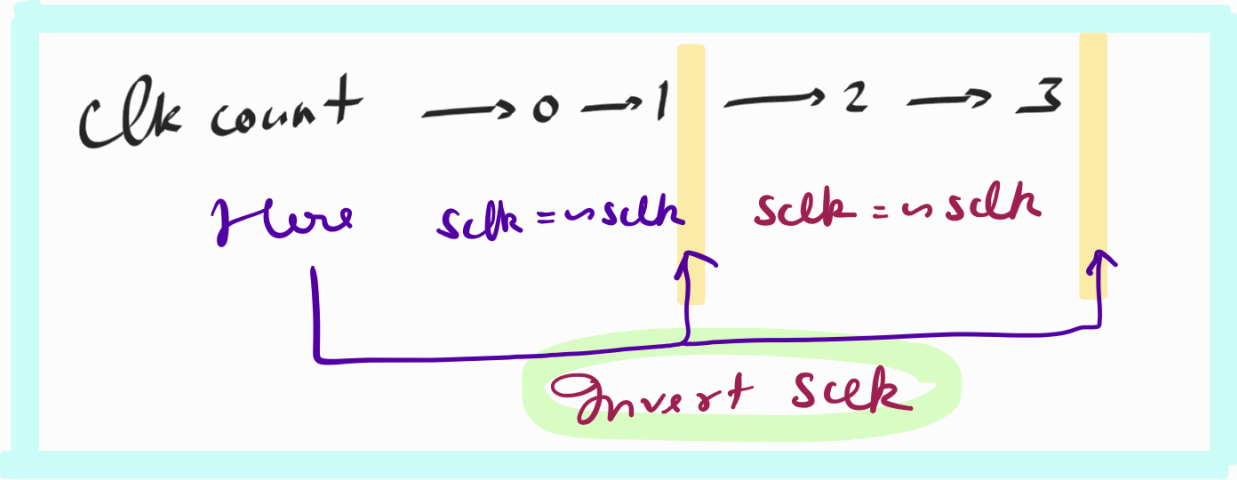


need 16 edges

(both trailing & leading)



we need total 8 SClks



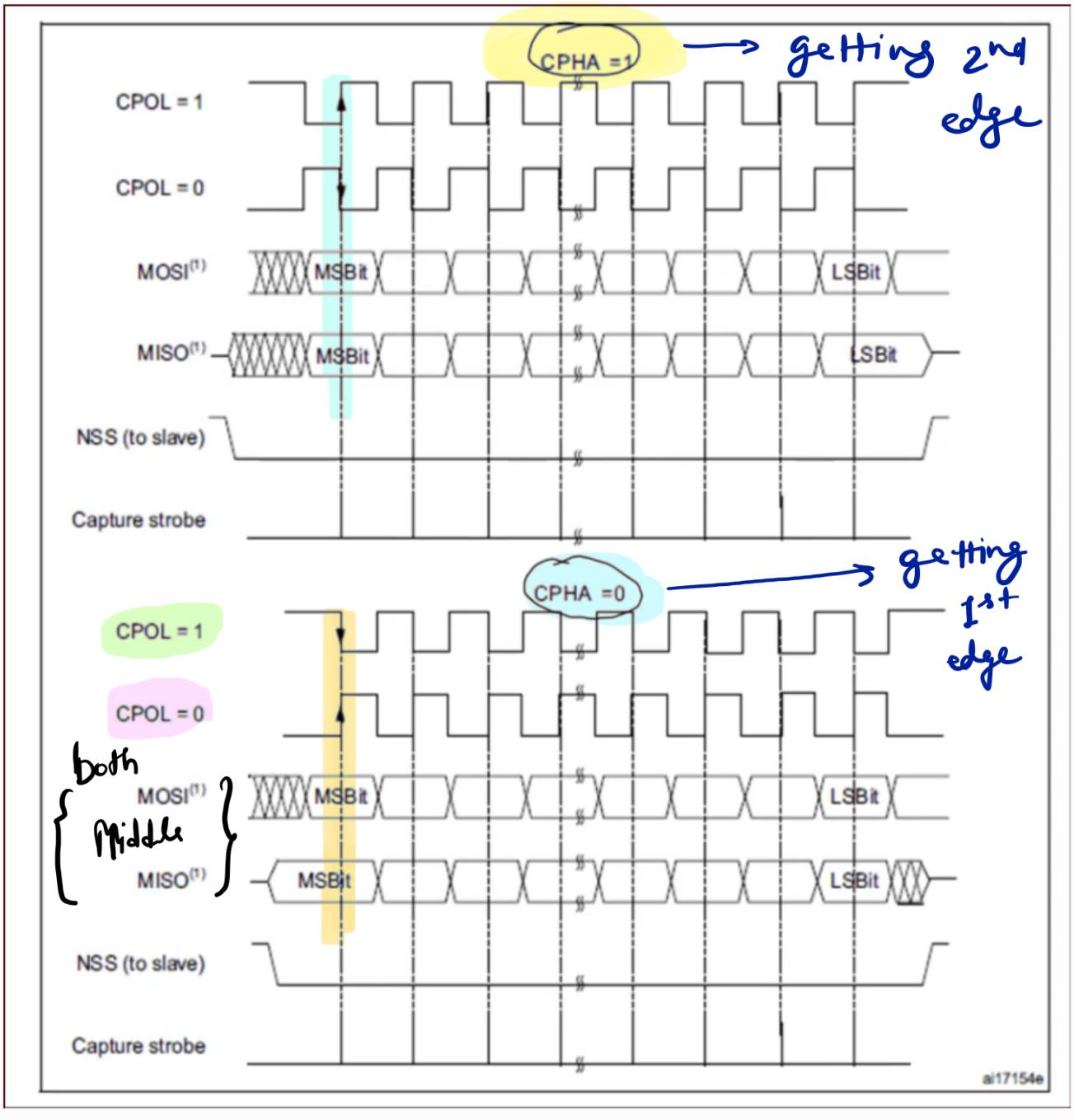
CPHA = 1

CPOL = 0 CPOL = 1

Understanding CPHA

CPHA = 0

CPOL = 0 CPOL = 1

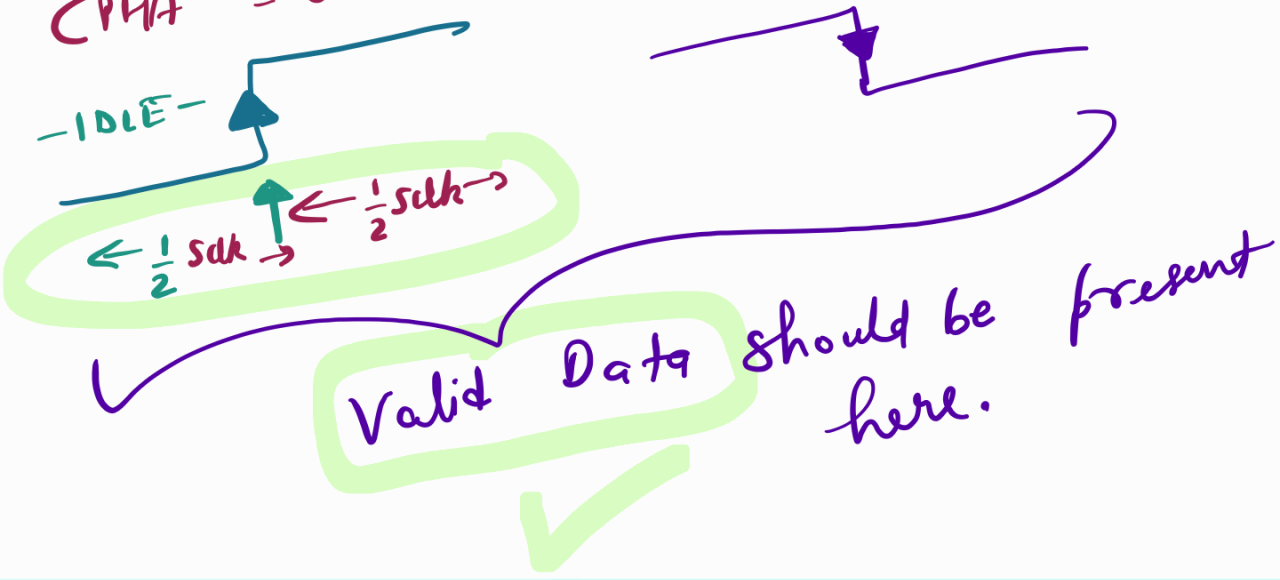


CPOL	CPHA	MODE:	edge
0	0	0	(Posedge) 1st
0	1	1	(negedge) 2nd
1	0	2	(negedge) 1st
1	1	3	(posedge) 2nd

CPOL = 0

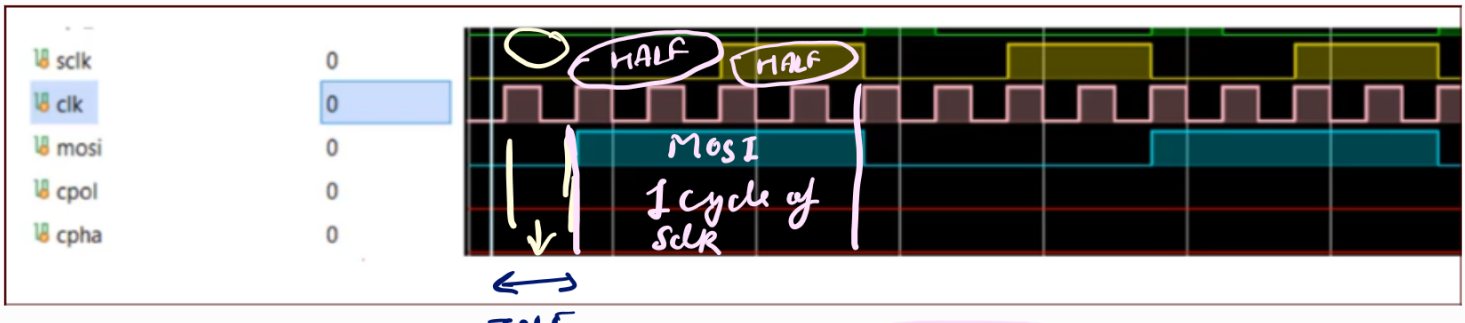
CPHA = 0

CPOL = 1



Mode : 0 CPOL = 0 CPHA = 0

@ → 1st edge



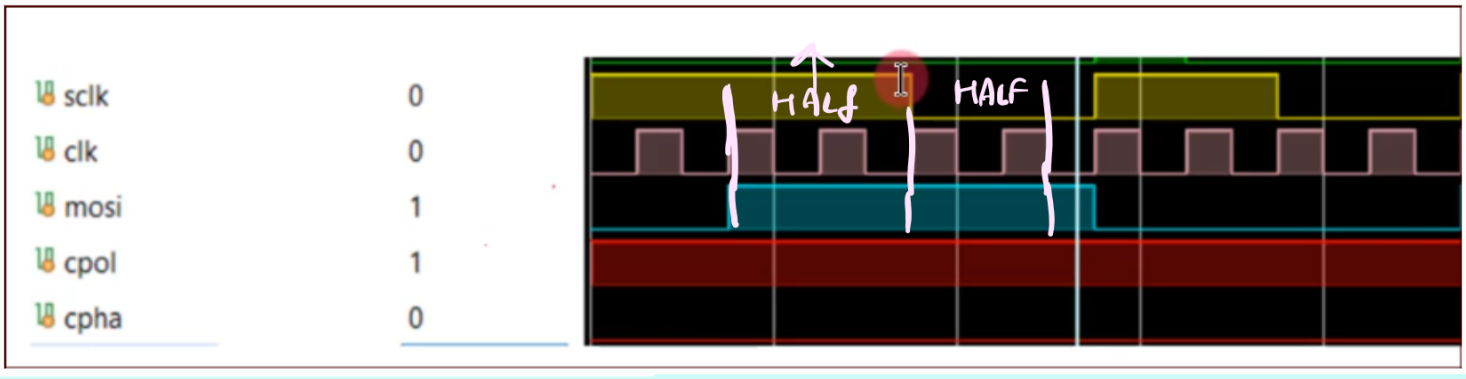
Half before edge of Sclk

& Half after our data should be Stable

Mode ②

CPOL = 1

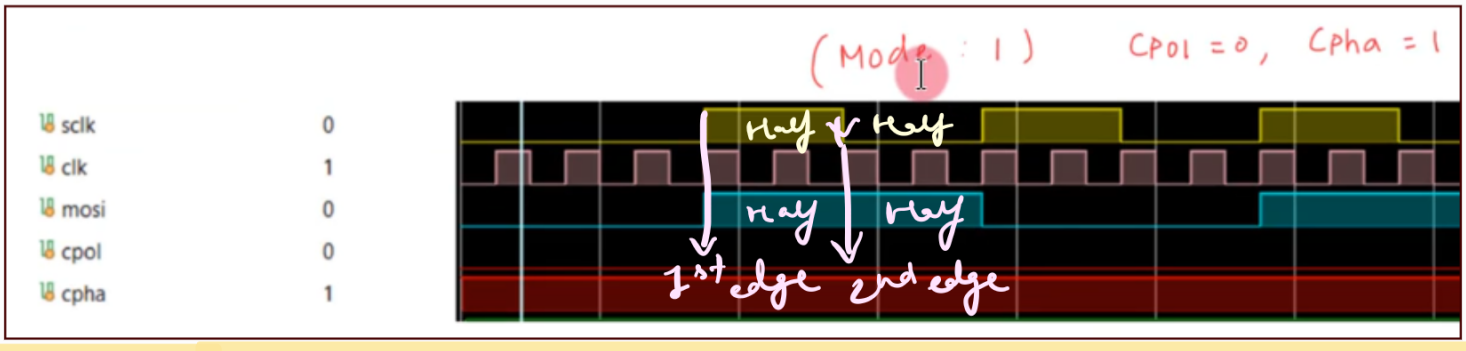
CPHA = 0 → 1st edge



Mode ①

CPHA = 1 → 2nd edge

CPOL = 0

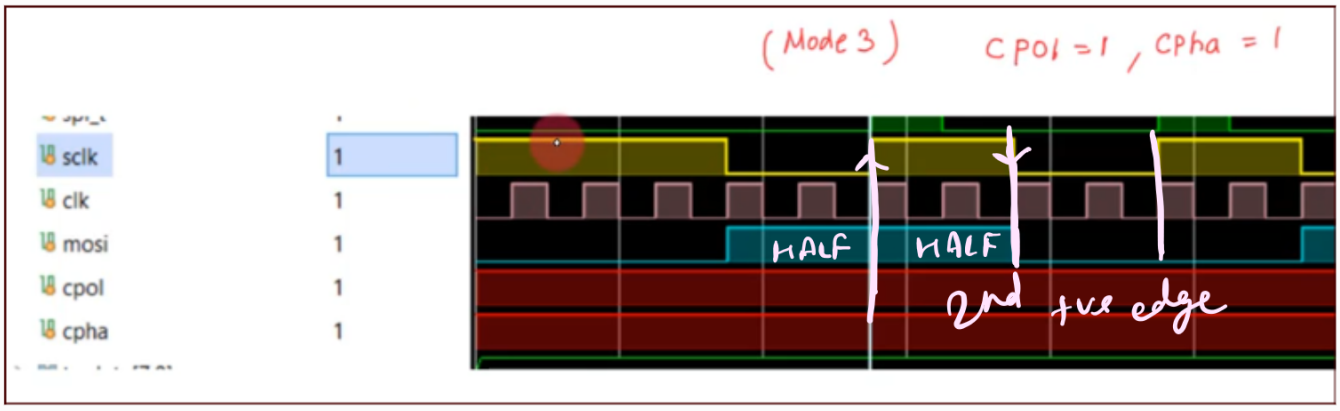


MODE 3

CPHA = 1

CPOL = 1

2nd edge



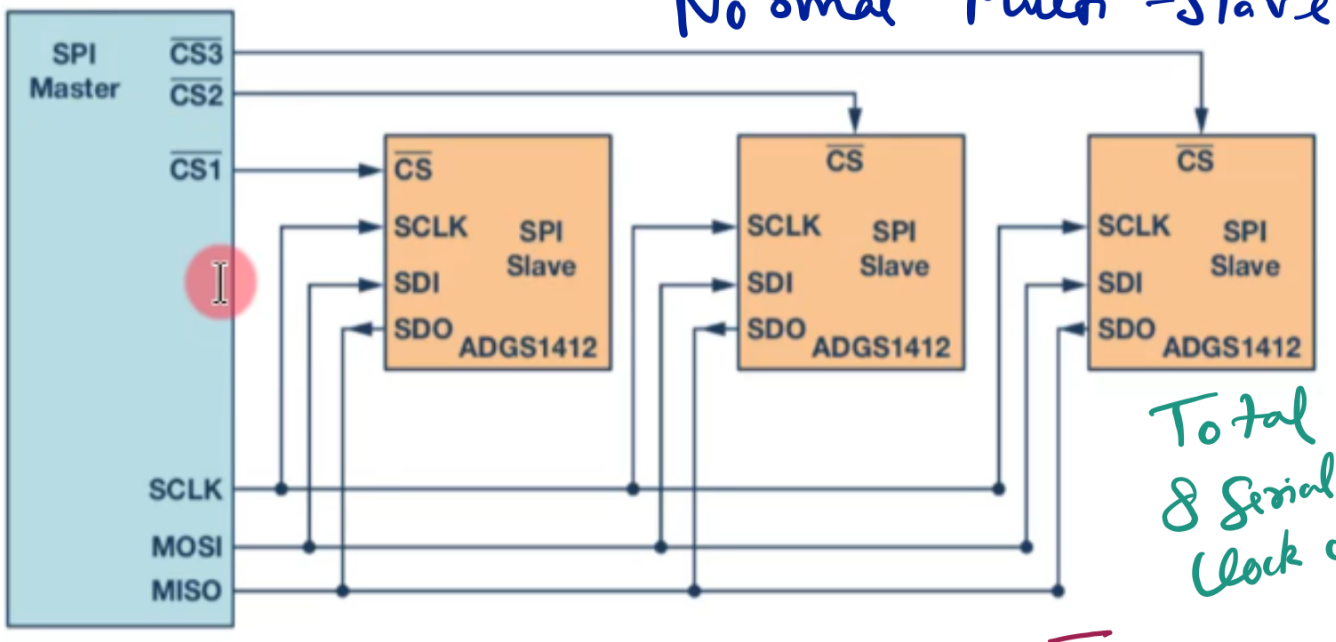
@ 2nd +ve edge

Data Middle point

Digilent PmoD DA4 (ADI AD5628)

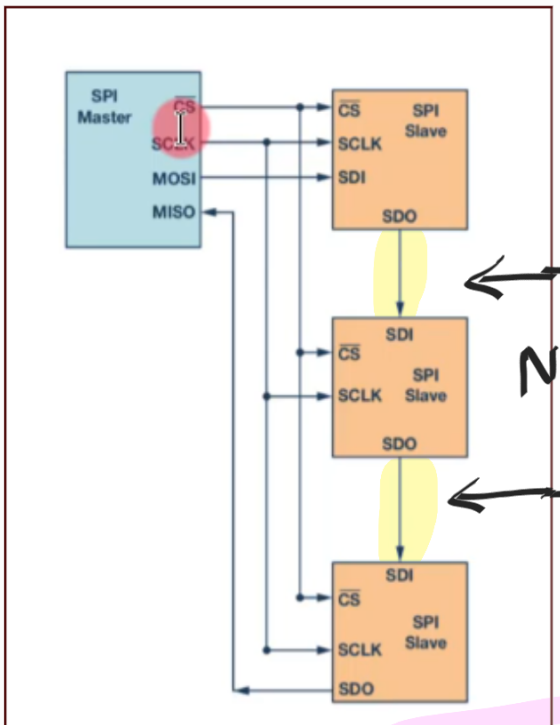
DAISY-CHAIN CONFIGURTION

Normal Multi-Slave



Total 8 Serial Clock cycle

More $\overline{CS}$ pins



Notice

Daisy-Chain

Less $\overline{CS}$ pin

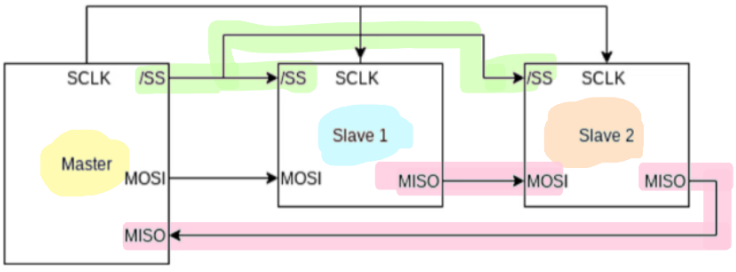
Only 1 pin

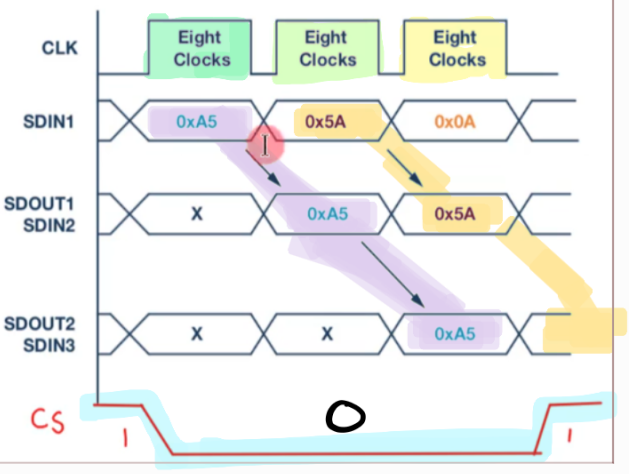
Advntg

Disadvntg

Total 16 Clock cycles — till 2nd Slave

& 24 till 3rd Slave


SCLK
/SS
/SS
SCLK
/SS
SCLK
Master
Slave 1
Slave 2
MOSI
MOSI
MISO
MOSI
MISO
MISO


CLK
Eight Clocks
Eight Clocks
Eight Clocks
SDIN1
0xA5
0x5A
0x0A
SDOUT1
SDIN2
X
0xA5
0x5A
SDOUT2
SDIN3
X
X
0xA5
CS
1
0
1