Common Serial Communication Protocols

GitHub: samlyu

Table of Contents

SPI - Features

- Serial Peripheral Interface (Motorola mid-1980s)
- 1 master & multiple slaves
- No speed limit
- 4 wires, synchronous, full-duplex
- Applications: Flash memory, LCD display, ADC/DAC, etc.

SPI - Features

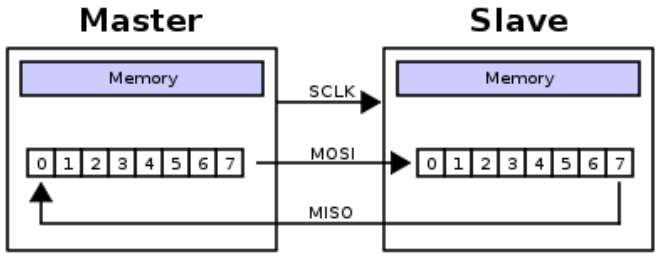
- Advantages:
  - Full-duplex
  - Push-pull drivers
  - Not limited to 8-bit words
  - Slaves do not need unique addresses
- Disadvantages:
  - More wires than I2C
  - No flow control
  - No slave acknowledgement
  - No error-checking protocols
  - No offcial standards

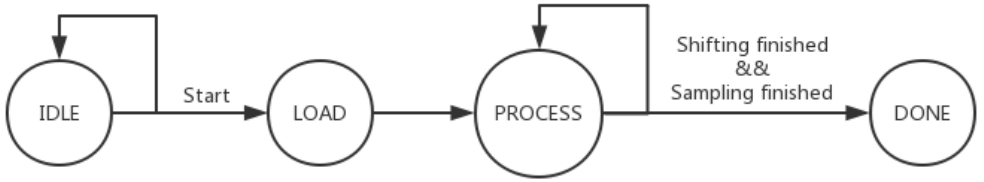
SPI - Interface

- SCLK : Serial clock (M -> S)
- MOSI: M out S in (M -> S)
- MISO: M in S out (S -> M)
- SS (CS): Slave select (active low, M -> S)

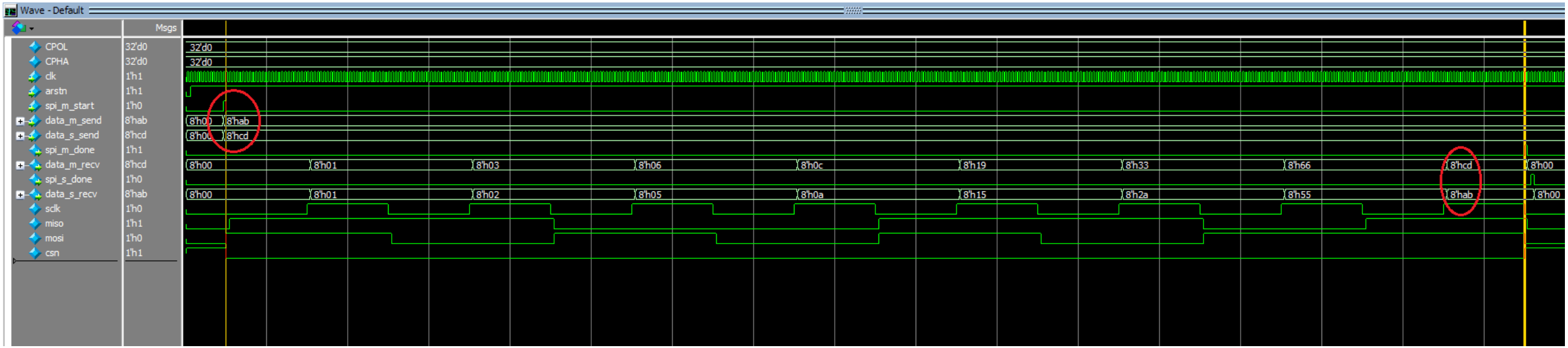
```
module spi_master
#(
    parameter   CLK_FREQ = 50_000_000,
                SPI_FREQ = 5_000_000,
                DATA_WIDTH = 8,
                CPOL = 0,
                CPHA = 0
)
(
    input   clk, arstn,
    input   [DATA_WIDTH-1:0]    data_send,
    input   spi_start,
    output  reg sclk, csn,
    output  mosi,
    input   miso,
    output  reg spi_done,
    output  reg [DATA_WIDTH-1:0]    data_recv
);
```

SPI - Operation





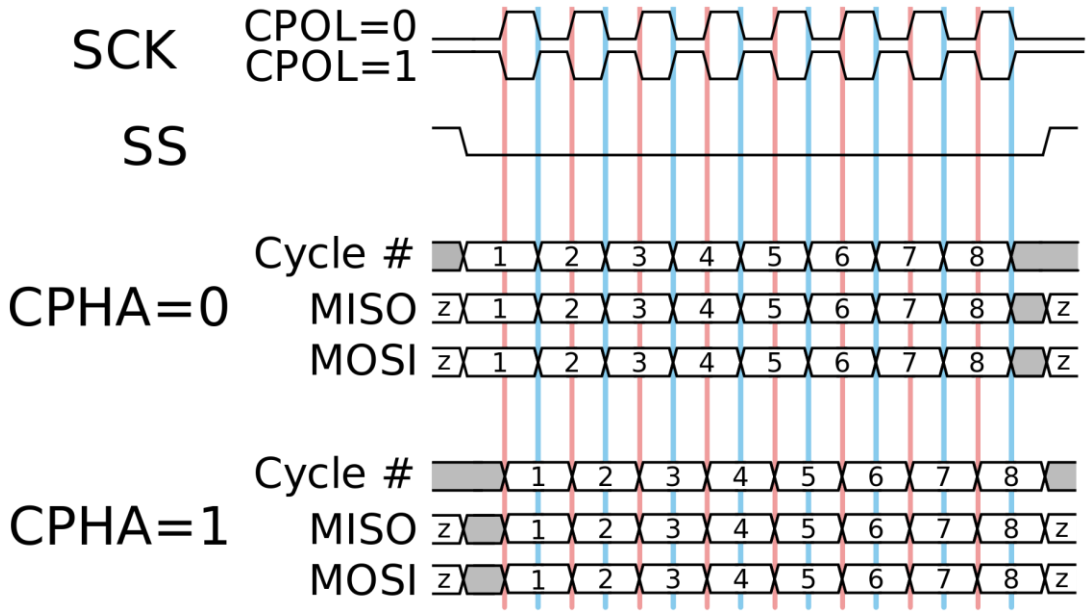
- LOAD: SCLK enable, CS enable, load data into shift register
- PROC: shift data out on MOSI, sample data in on MISO



SPI - Modes

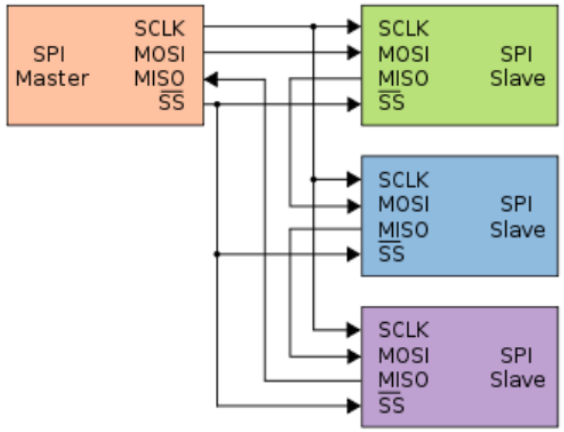
- Clock Polarity (CPOL): 0 -> SCLK idle at 0; 1 -> SCLK idle at 1
- Clock Phase (CPHA):
  - 0 -> Sample data @ leading edge, shift data @ trailing edge
  - 1 -> Shift data @ leading edge, sample data @ trailing edge

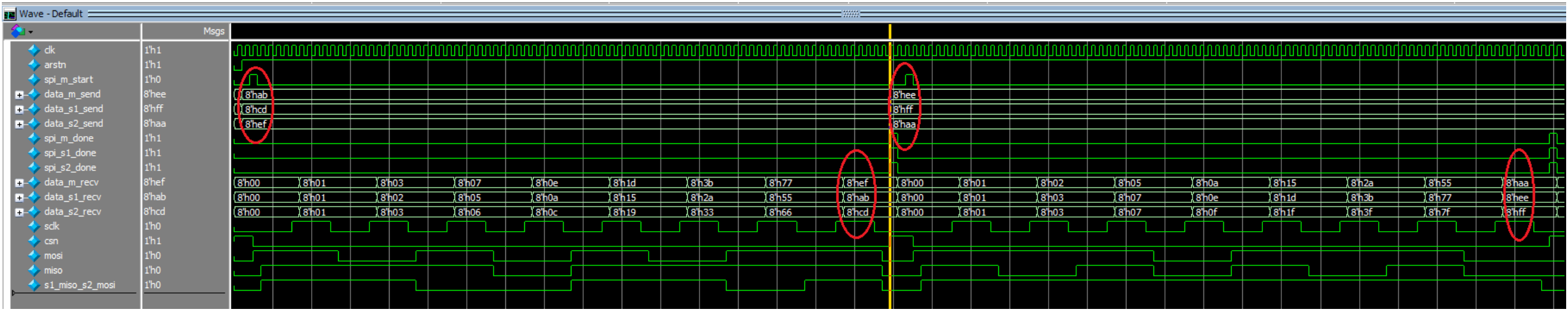
Mode	CPOL	CPHA
0	0	0
1	0	1
2	1	0
3	1	1



SPI - Slave Configurations

- Independent slave configuration: each slave has a CS
- Daisy chain : only 1 CS (Data: M -> S1 -> S2 ... -> M)





SPI - Variants

- Dual SPI: {MOSI, MISO} -> {IO0, IO1}
- Quad SPI: {MOSI, MISO} -> {IO0, IO1, IO2, IO3}
- Example:
  - SPI: 7 6 5 4 3 2 1 0
  - Dual SPI:
    - 7 5 3 1
    - 6 4 2 0
  - Quad SPI:
    - 7 3
    - 6 2
    - 5 1
    - 4 0

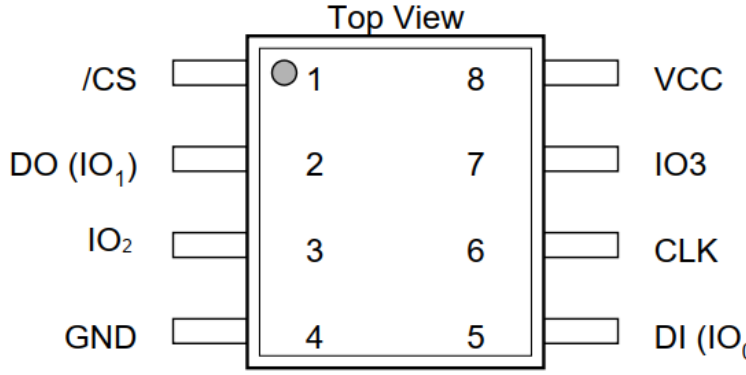


Figure 1a. W25Q32JV Pin Assignments, 8-pin SOIC / VSOP 208-mil (Package Code SS / ST)

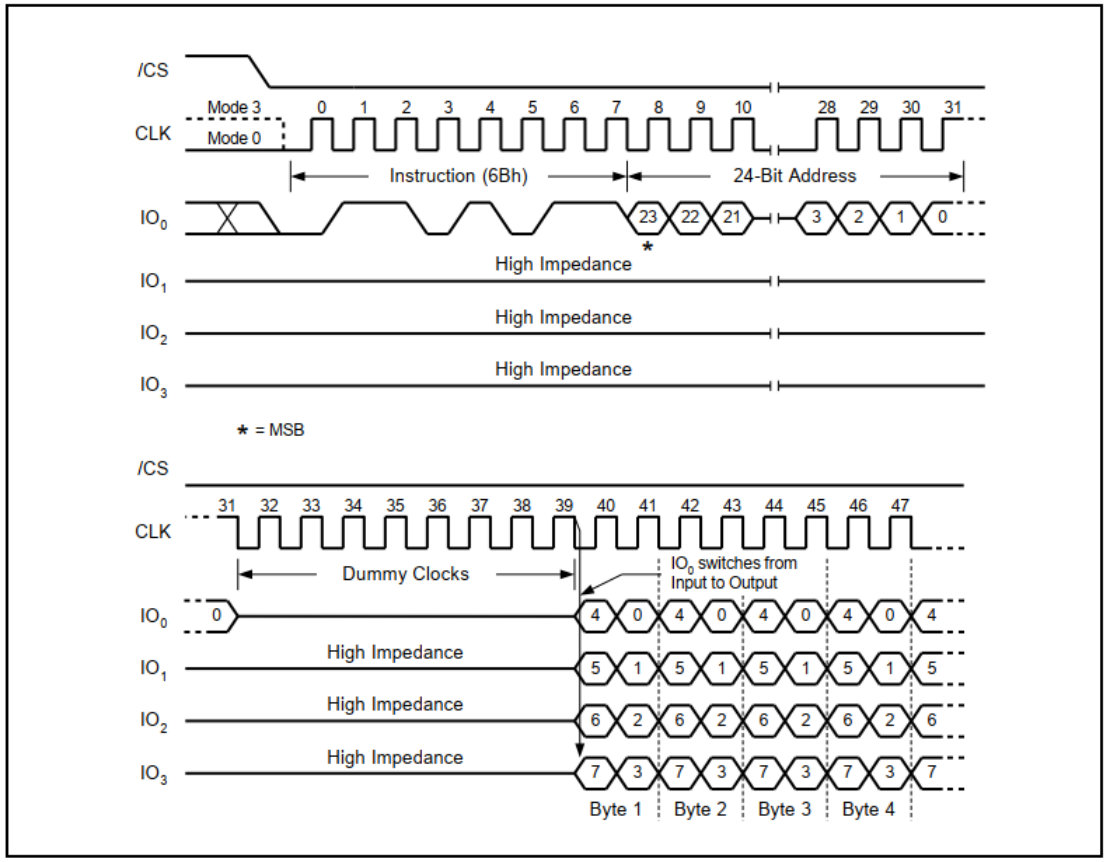


Figure 20. Fast Read Quad Output Instruction

SPI - Examples

winbond

8.2.7 Fast Read (0Bh)

The Fast Read instruction is similar to the Read Data instruction except that it can operate at the highest possible frequency of FR (see AC Electrical Characteristics). This is accomplished by adding eight “dummy” clocks after the 24-bit address as shown in Figure 16. The dummy clocks allow the devices internal circuits additional time for setting up the initial address. During the dummy clocks the data value on the DO pin is a “don’t care”.

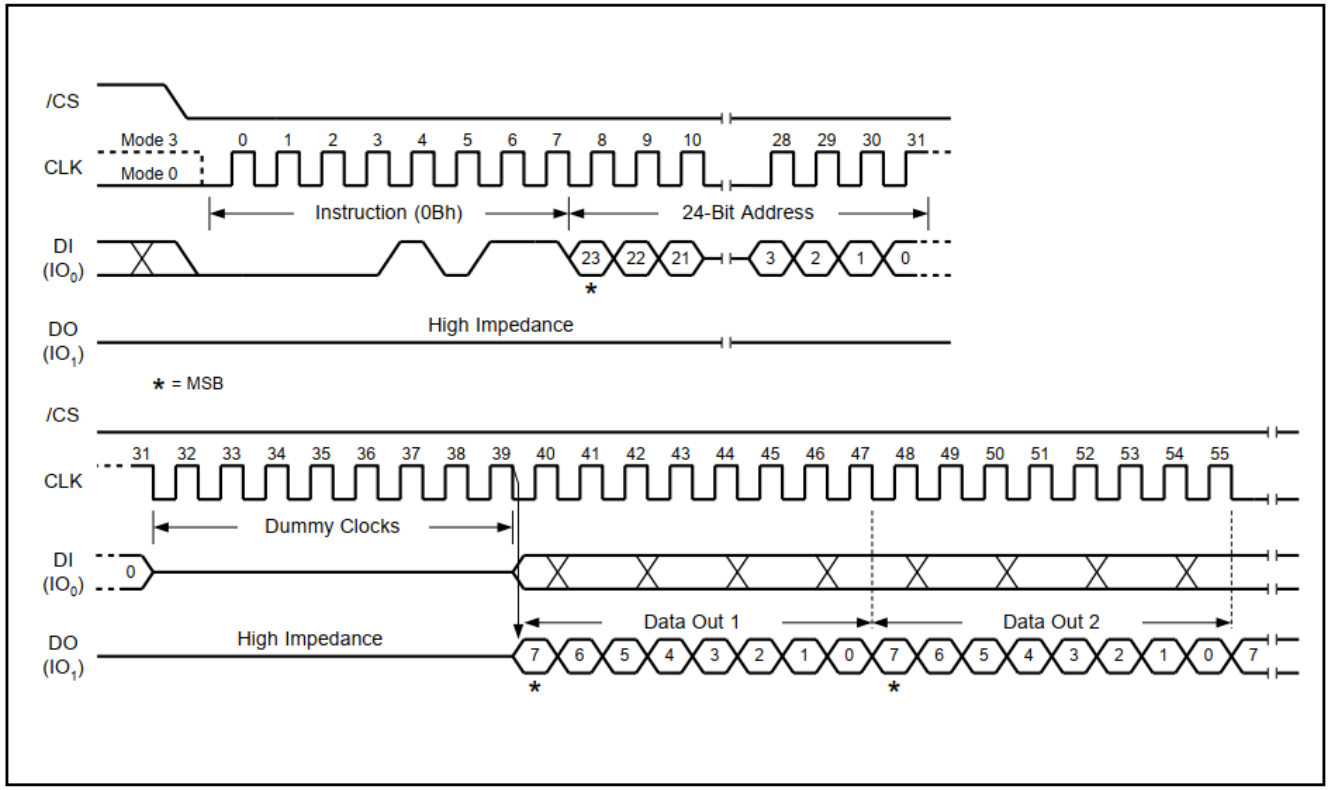
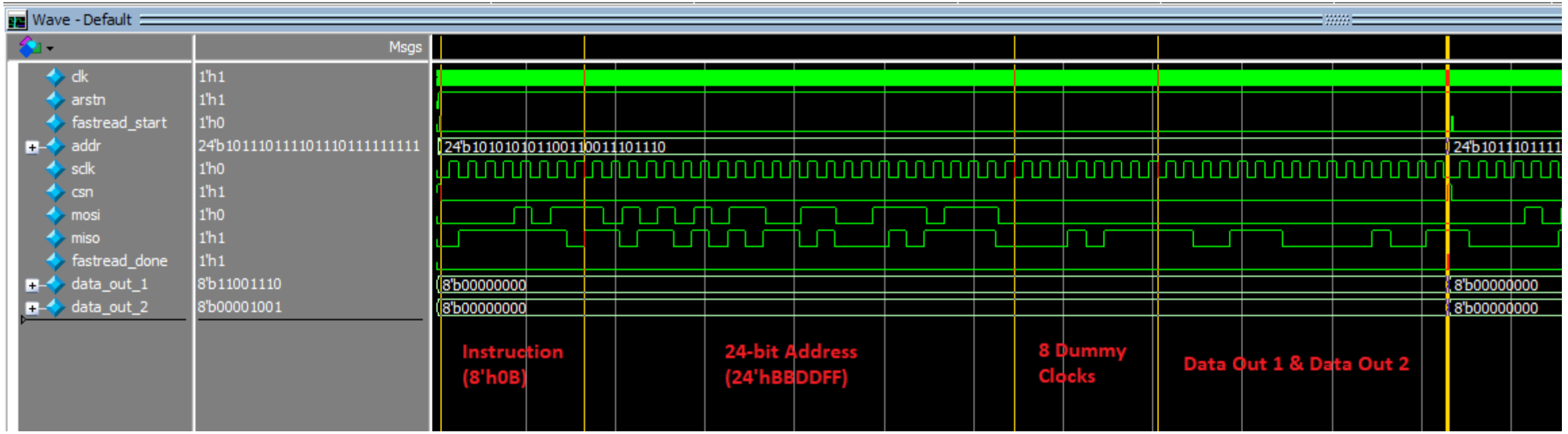


Figure 16. Fast Read Instruction

SPI - Examples



SPI - Examples

8.4.2.6 Manual Channel n Select (MAN_Ch_n)

The devices can be programmed to convert a particular analog input channel by operating in manual channel *n* scan mode (MAN_Ch_n). This programming is done by writing a valid manual channel *n* select command (MAN_Ch_n) in the command register, as shown in Figure 83. As shown in Figure 83, the $\overline{CS}$ signal can be pulled high immediately after the MAN_Ch_n command or after reading the output data of the frame. However, in order to accurately acquire and convert the input signal on the next channel, the command frame must be a complete frame of 32 SCLK cycles. Refer to Table 6 for a list of commands to select individual channels during MAN_Ch_n mode.

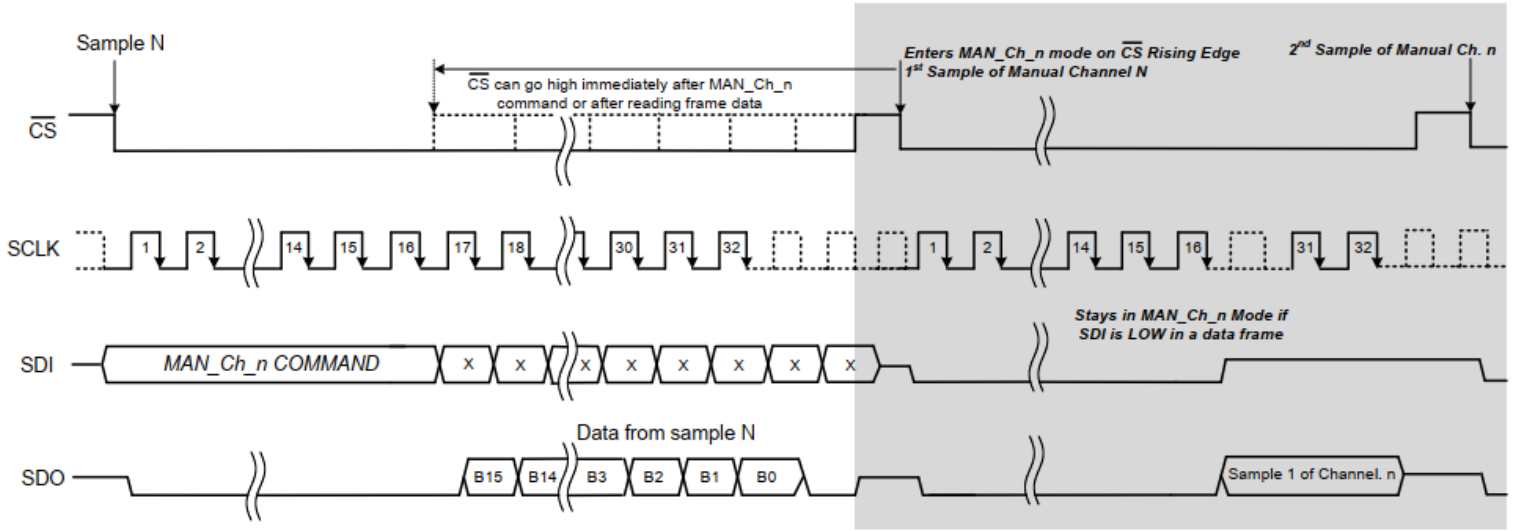
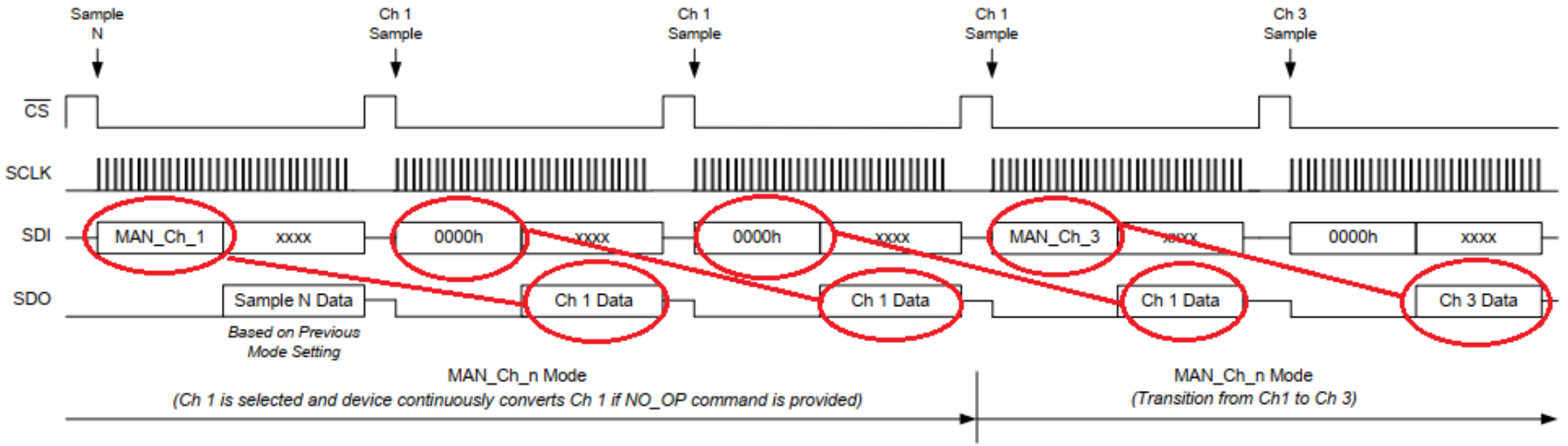
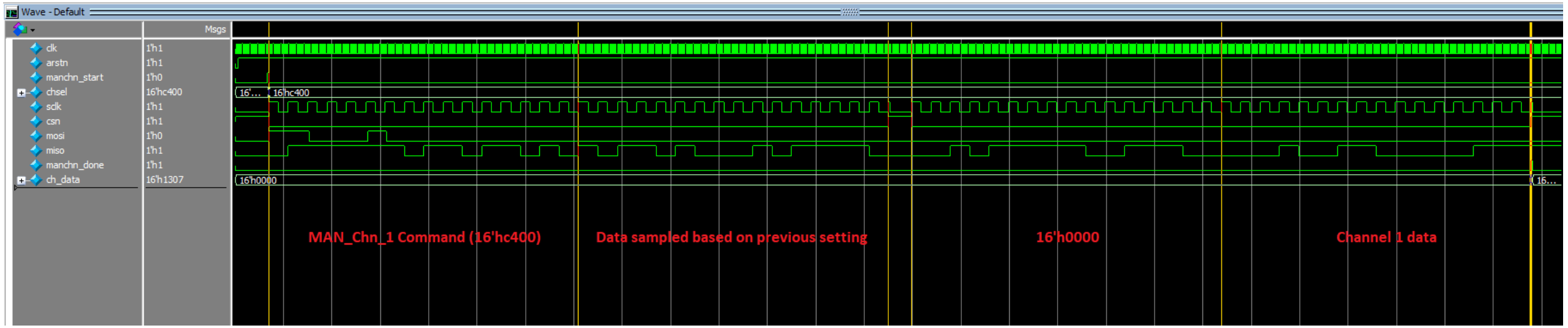


Figure 83. Enter MAN_Ch_*n* Scan Mode Timing Diagram

C000h	Channel 0 input is selected
C400h	Channel 1 input is selected
C800h	Channel 2 input is selected
CC00h	Channel 3 input is selected
D000h	Channel 4 input is selected
D400h	Channel 5 input is selected
D800h	Channel 6 input is selected
DC00h	Channel 7 input is selected

SPI - Examples





SPI - Examples

8.4.1.2 Data Acquisition Example

This section provides an example of how a host processor can use the device interface to configure the device internal registers as well as convert and acquire data for sampling a particular input channel. The timing diagram shown in Figure 72 provides further details.

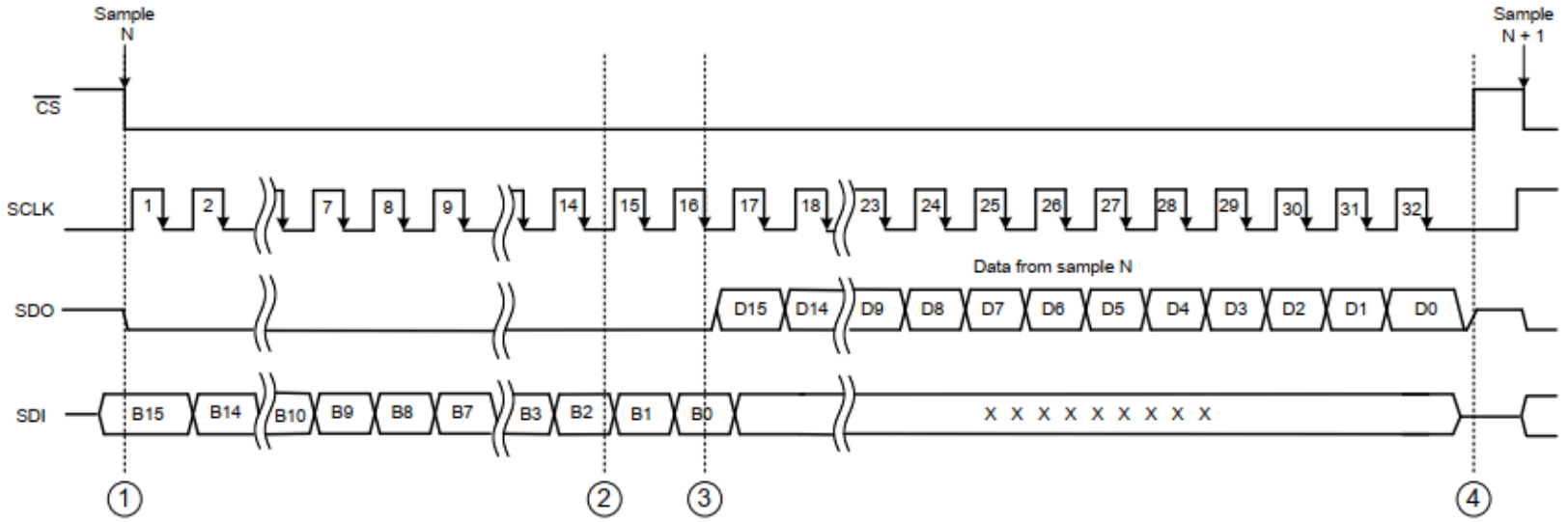


Figure 72. Device Operation Using the Serial Interface Timing Diagram

- ***Event 3:*** At the 16th falling edge of the SCLK signal, the device reads the LSB of the input word on the SDI line. The device does not read anything from the SDI line for the remaining data frame. On the same edge, the MSB of the conversion data is output on the SDO line and can be read by the host processor on the subsequent falling edge of the SCLK signal. For 16 bits of output data, the LSB can be read on the 32nd SCLK falling edge. The SDO outputs 0 on subsequent SCLK falling edges until the next conversion is initiated.

I2C - Features

- Inter-Integrated Circuit (Philips (NXP) 1982)
- Multiple masters & multiple slaves
- Low speed, short distance
- 2 wires, synchronous, half-duplex
- Applications: RAM, E2PROM, Low speed ADC/DAC, etc.

I2C - Features

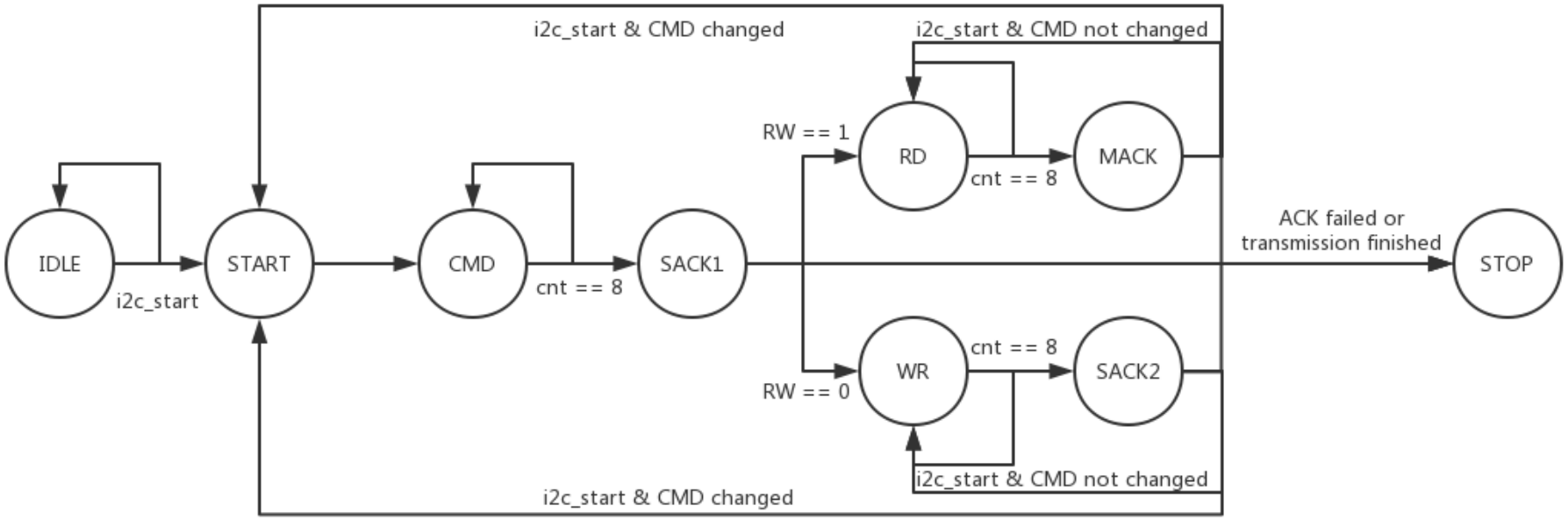
- Advantages:
  - Only 2 wires
  - Flow control (Start, Stop, R/W)
  - Slave acknowledgement
  - Multi-master (need arbitration)
- Disadvantages:
  - Each slave needs a unique address
  - No error-checking protocols
  - Send data byte by byte
  - Low speed (100kbps ~ 5Mbps)
  - Half-duplex
  - Open-drain drivers

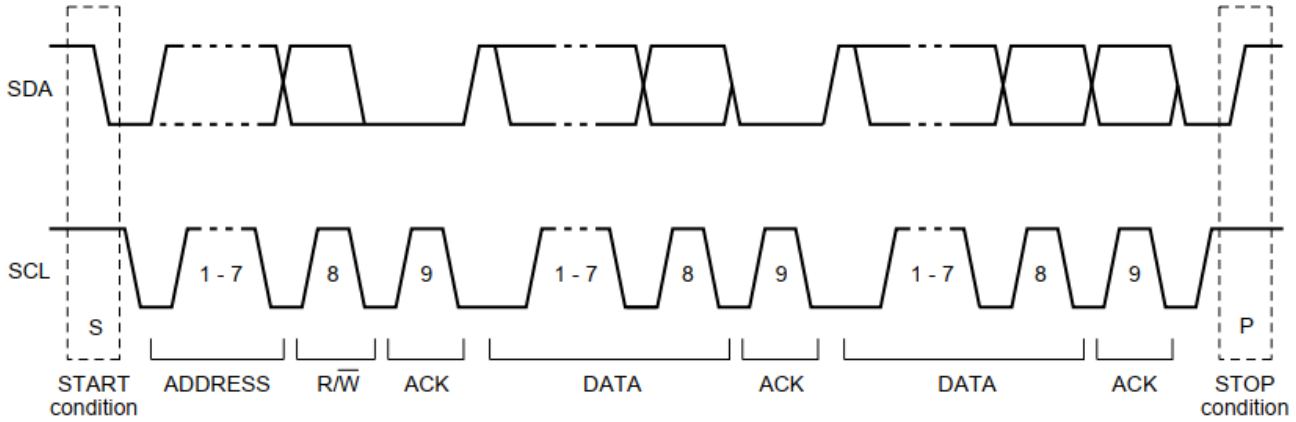
I2C - Interface

- SCL: Serial Clock (inout)
- SDA: Serial Data (inout)

```
module i2c_master
#(
    parameter   CLK_FREQ = 50_000_000,
                I2C_FREQ = 500_000
)
(
    input   clk, arstn,
    input   i2c_start,
    input   [7-1:0] addr,
    input   rw,
    input   [8-1:0] data_send,
    output  reg i2c_done,
    output  reg [8-1:0] data_recv,
    output  data_recv_done,

    inout   sda,
    output  scl
);
```

I2C - Operation





I2C - Operation

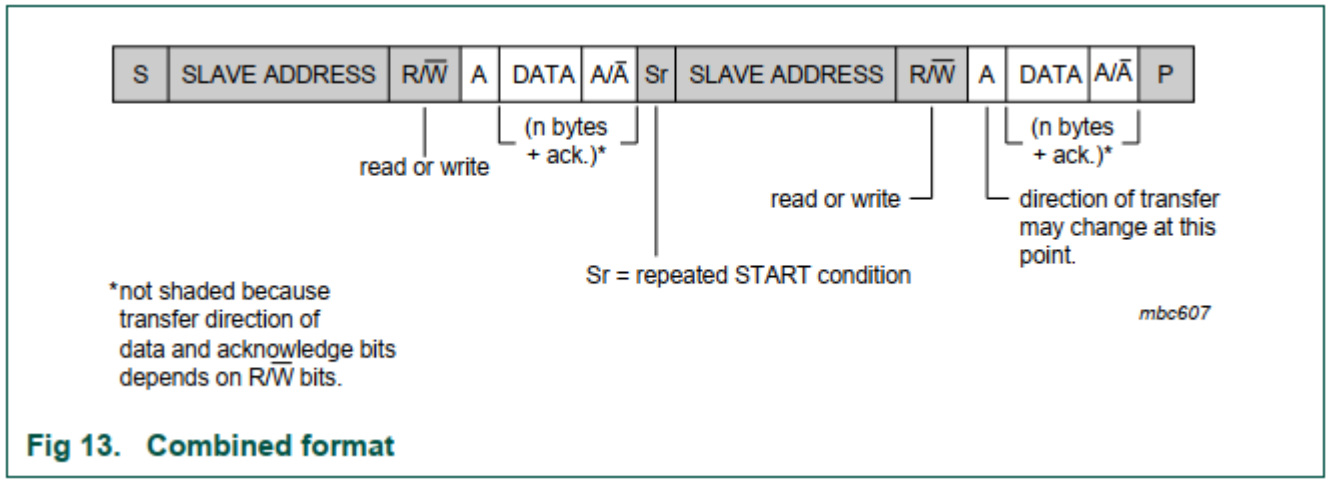
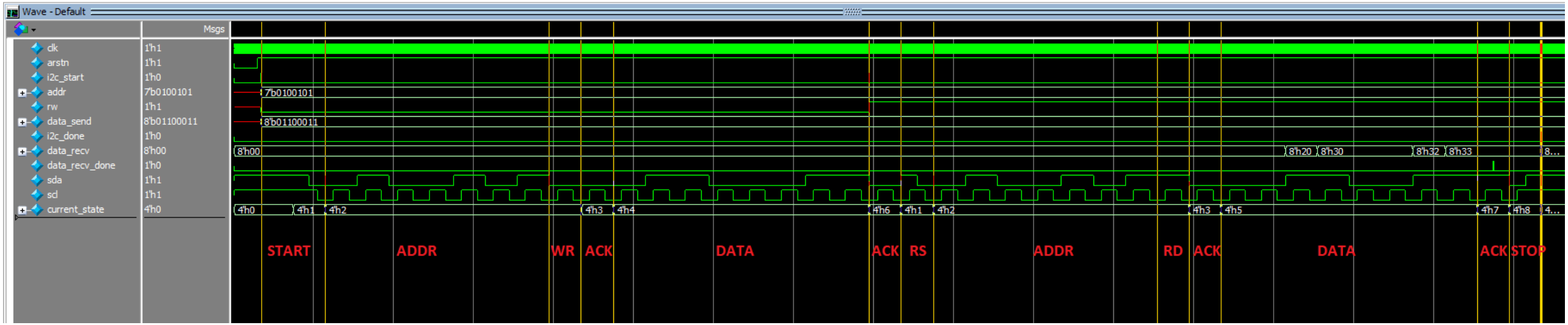


Fig 13. Combined format



I2C - Acknowledgment

- ACK & NACK:
  - ACK: 1'b0; NACK: 1'b1 (passive pullup or drive high)
  - NACK situations:
    - No receiver on the bus
    - Receiver not able to communicate
    - Receiver gets data it does not understand
    - Receiver cannot receive any more data
    - Master receiver signaling the end of transfer to the slave transmitter

I2C - Optional Applicability

- Clock Stretching (SCL):
  - Slave can hold down the SCL line if it is not ready.
- Arbitration (SDA):
  - A master can delay pending I2C transfer if the lines are busy
  - 2 masters start a transfer at the same time: the master who sent the first '0' wins!
  - The master who lost arbitration: stops its I2C transfer
- 10-bit Addressing:

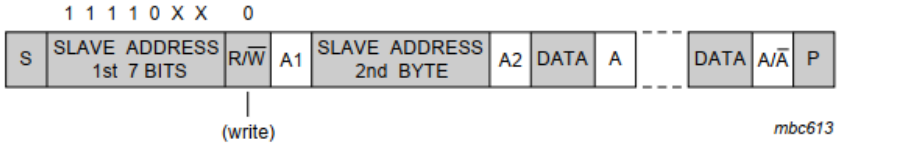


Fig 14. A master-transmitter addresses a slave-receiver with a 10-bit address

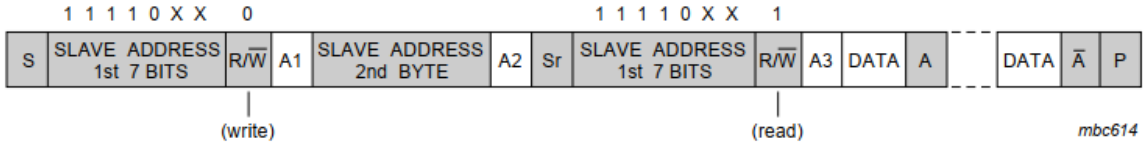
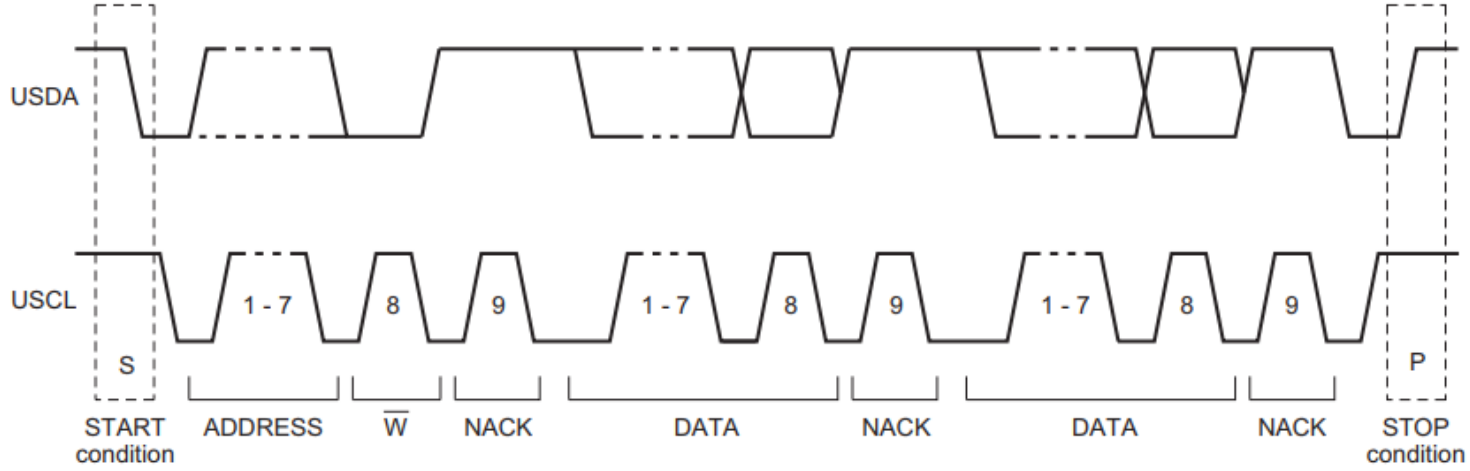


Fig 15. A master-receiver addresses a slave-transmitter with a 10-bit address

I2C - Modes

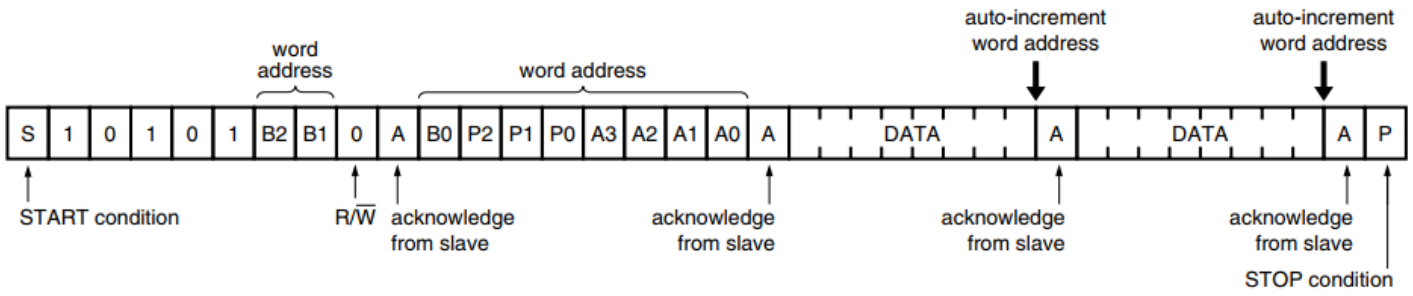
- Standard-mode: 100kbps
- Fast-mode: 400kbps
- Fast-mode Plus: 1Mbps
- High-speed mode: 3.4Mbps
- Ultra Fast-mode: 5Mbps, write-only (No ACK), push-pull, single-master



I2C - Example

6.2.1 Byte/word write

Write command may be used to set the address for a subsequent Read command. For a write operation, the PCA24S08A requires a second address field. The address field associated with the two software selectable bits in the slave address is a word address providing access to the 1024 bytes of memory, as shown in Figure 5. Upon receipt of the word address, the PCA24S08A responds with an acknowledge and awaits the next 8 bits of data, again responding with an acknowledge. Word address is automatically incremented.



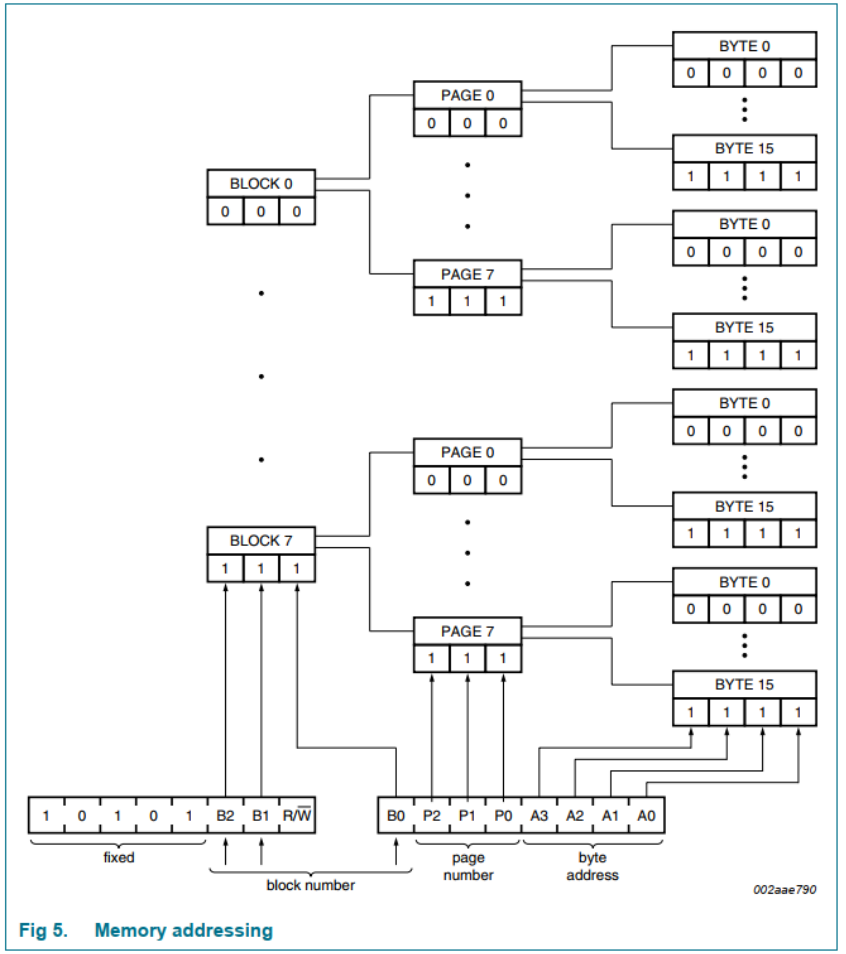
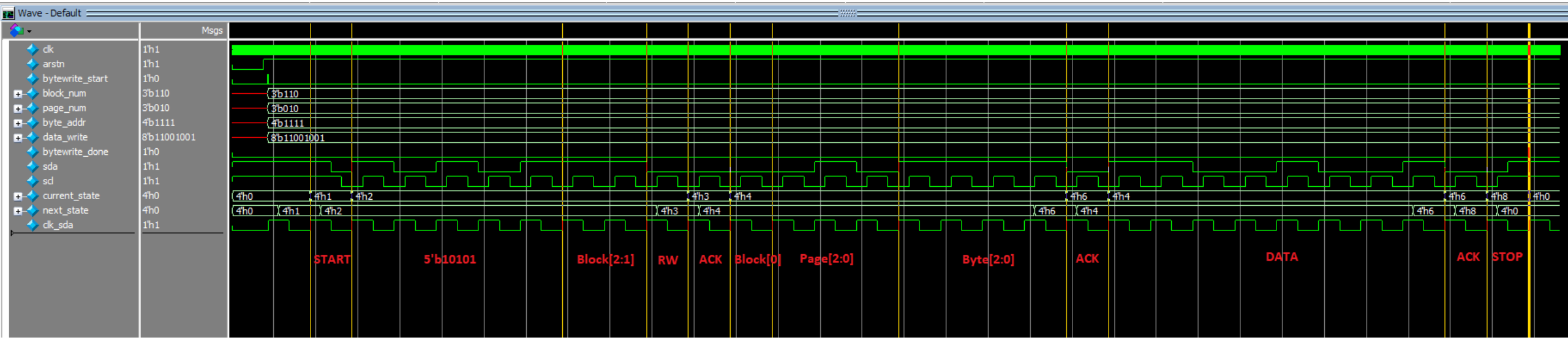


Fig 5. Memory addressing

I2C - Example


Wave - Default
Msgs
clk
arstn
bytewrite_start
block_num
page_num
byte_addr
data_write
bytewrite_done
sda
scl
current_state
next_state
clk_sda
1'h1
1'h1
1'h0
3'b110
3'b010
4'b1111
8'b11001001
1'h0
1'h1
1'h1
4'h0
4'h0
1'h1
3'b110
3'b010
4'b1111
8'b11001001
START
5'b10101
Block[2:1]
RW
ACK
Block[0]
Page[2:0]
Byte[2:0]
ACK
DATA
ACK
STOP

UART - Features

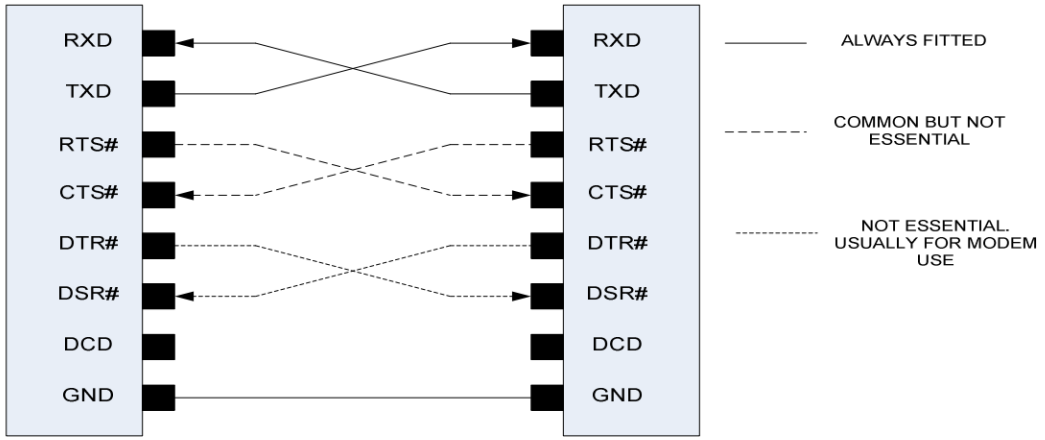
- Universal Asynchronous Receiver-Transmitter (Western Digital, 1971)
- 2 wires
- asynchronous
- full-duplex
- Applications: MODEM, Industrial controller, CCTV, etc.

UART - Features

- Advantages:
  - Only 2 wires
  - No clock signal
  - Parity check
  - Have different connectors (RS-232, RS-422, RS-485, etc.)
- Disadvantages:
  - Max data frame length: 9 bits
  - Baud rate need to be set properly before transmission

UART - Interface

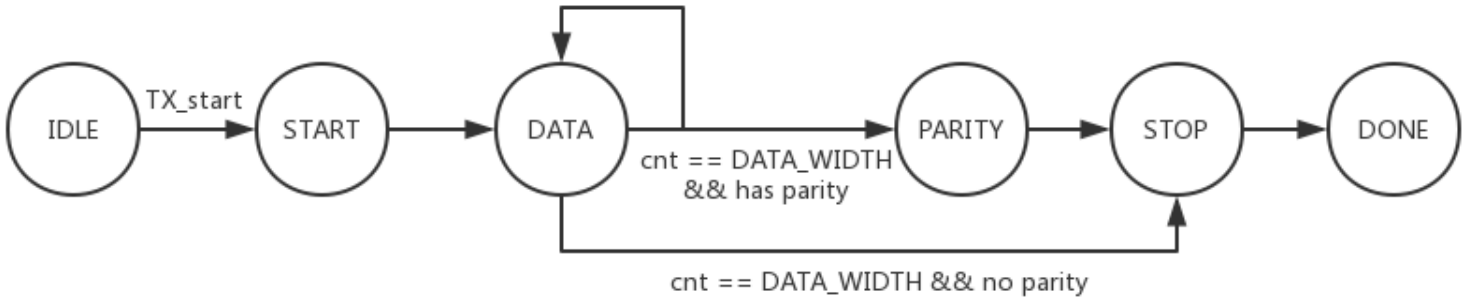
- TXD: Data output (TX -> RX)
- RXD: Data input (TX -> RX)
- Optional flow control signals:
  - RTS: Ready to send
  - CTS: Clear to send
  - DTR: Data terminal ready
  - DSR: Data set ready
  - DCD: Data carrier detect

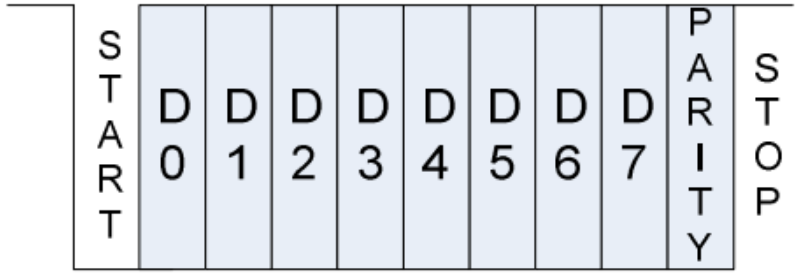



```
module uart_tx
#(
    parameter   CLK_FREQ = 50_000_000,
                BAUD_RATE = 9600,
                PARITY = "NONE",
                DATA_WIDTH = 8
)
(
    input   clk, arstn,
    input   tx_start,
    output  reg tx_done,
    input   [DATA_WIDTH-1:0]    tx_data,
    output  reg TXD
);
```

```
module uart_rx
#(
    parameter   CLK_FREQ = 50_000_000,
                BAUD_RATE = 9600,
                PARITY = "NONE",
                DATA_WIDTH = 8
)
(
    input   clk, arstn,
    output  reg rx_done,
    output  reg [DATA_WIDTH-1:0]    rx_data,
    output  reg rx_error,
    input   RXD
);
```

UART - Operation






```
// FSM combinational
always@(*) begin
    case(current_state)
        IDLE:   next_state = tx_start ? READY : IDLE;
        READY:  next_state = shift_en ? START : READY;
        START:  next_state = shift_en ? SHIFT : START;
        SHIFT:  next_state = (shift_en && bit_count == DATA_WIDTH-1) ? ( (PARITY != "NONE") ? PARI : STOP ) : SHIFT;
        PARI:   next_state = shift_en ? STOP : PARI;
        STOP:   next_state = shift_en ? DONE : STOP;
        DONE:   next_state = IDLE;
        default:    next_state = IDLE;
    endcase
end
```

UART - Operation

- Detect start bit: negedge@RXD
- Number of stop bits: 1, 1.5 or 2 (give the receiver some extra time to process data)
- Sample data bits:
  - Generate 9 sample_en in 1 bit data
  - Sample data at the 3 middle sample_ens
  - Majority of the 3 sampled data is taken as the received data
- Baud rate: symbol transmitted per second
  - Baud_rate = Bit_rate / bits_per_symbol
  - Common baud rates: 110, 150, 300, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200...

Figure 242. Start bit detection when oversampling by 16 or 8

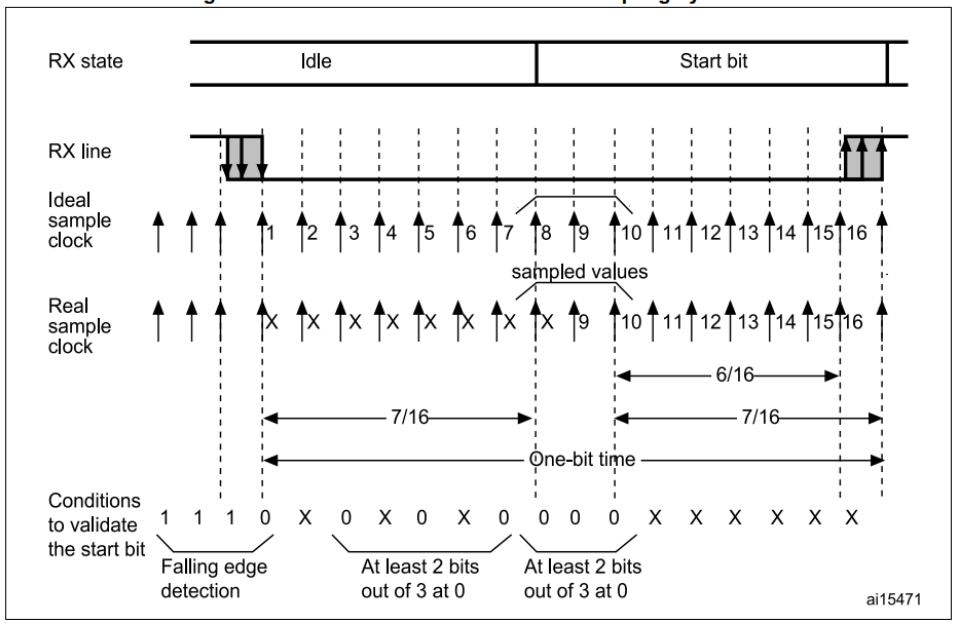
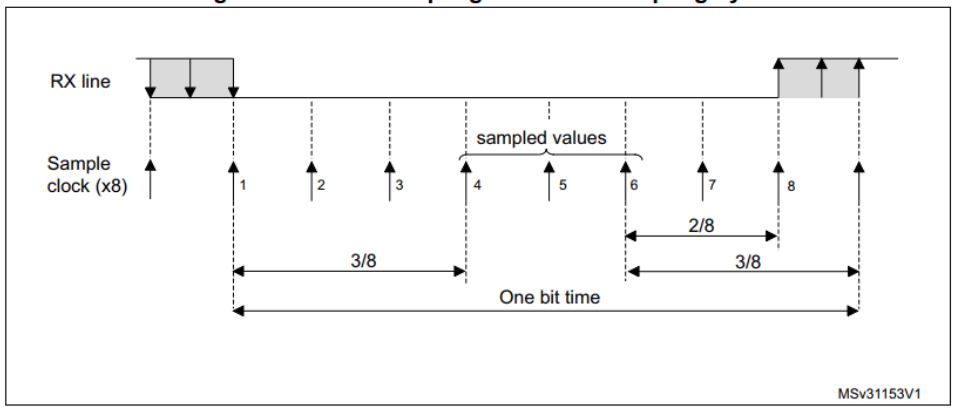


Figure 244. Data sampling when oversampling by 8

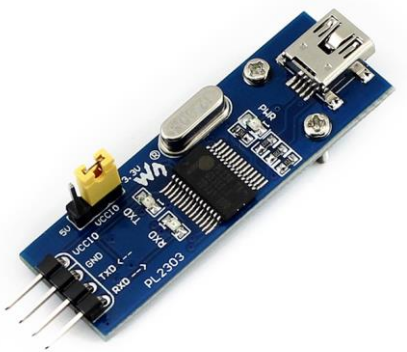


UART - Operation

- Parity check:
  - Odd parity: number of 1'b1 in {data, parity_bit} should be odd
  - Even parity: number of 1'b1 in {data, parity_bit} should be even
- Types of error:
  - Overrun error: receiver buffer already full before transmission finished
  - Underrun error: transmitter buffer already empty before transmission finished
  - Framing error: fail to receive STOP bit at the expected time
  - Parity error: fail the parity check
- USART:
  - UART with clock signal
  - Can reach higher data rate
  - Can support more protocols than UART

UART - Standards

- TTL UART
  - 0 to 3.3V/5V
  - Single ended
  - 1 TX, 1 RX
  - Transmission length < 2m



Get a Serial instance and configure/open it later:

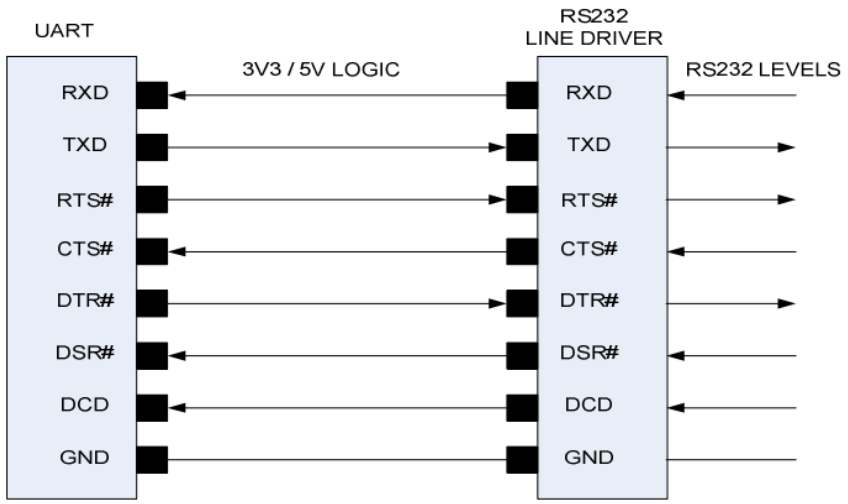
```
>>> ser = serial.Serial()
>>> ser.baudrate = 19200
>>> ser.port = 'COM1'
>>> ser
Serial<id=0xa81c10, open=False>(port='COM1', baudrate=19200, bytesize=8, parity='N', stopbits=1, timeout=None, xonxoff=0, rtscts=0)
>>> ser.open()
>>> ser.is_open
True
>>> ser.close()
>>> ser.is_open
False
```

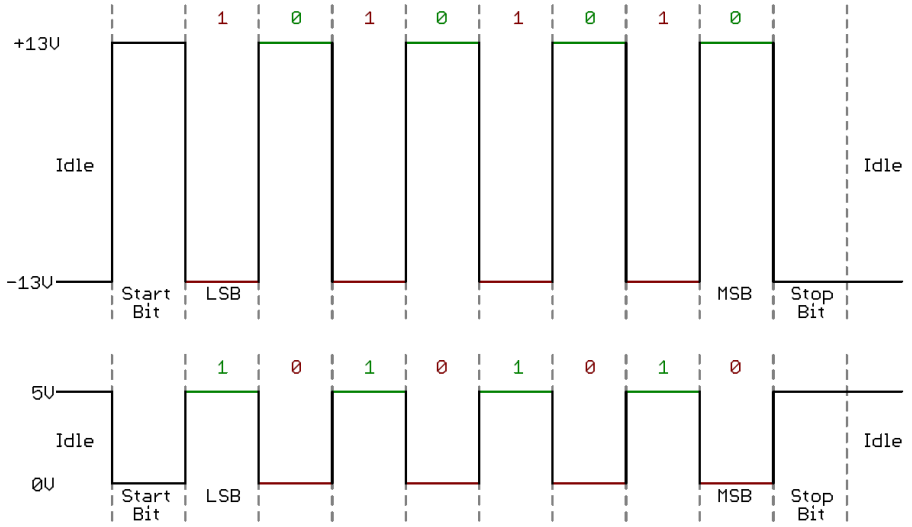
Also supported with context manager:

```
serial.Serial() as ser:
    ser.baudrate = 19200
    ser.port = 'COM1'
    ser.open()
    ser.write(b'hello')
```

UART - Standards

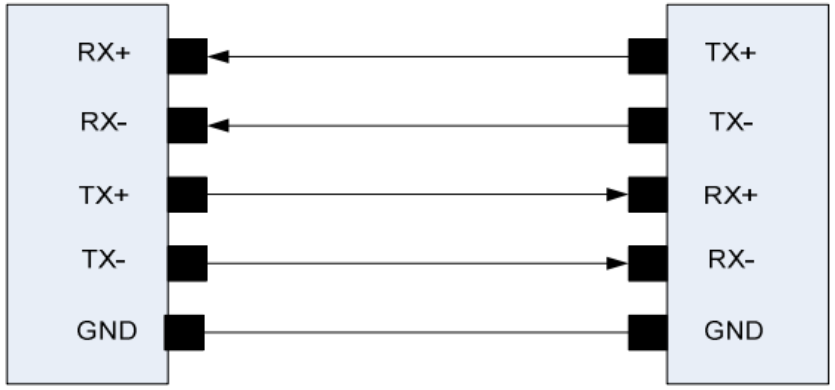
- RS-232
  - +/-3V to +/-15V
  - Single ended
  - 1 TX, 1 RX
  - Transmission length < 15m



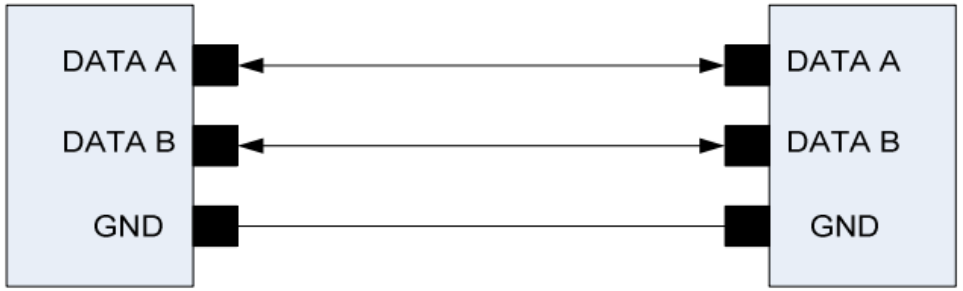


UART - Standards

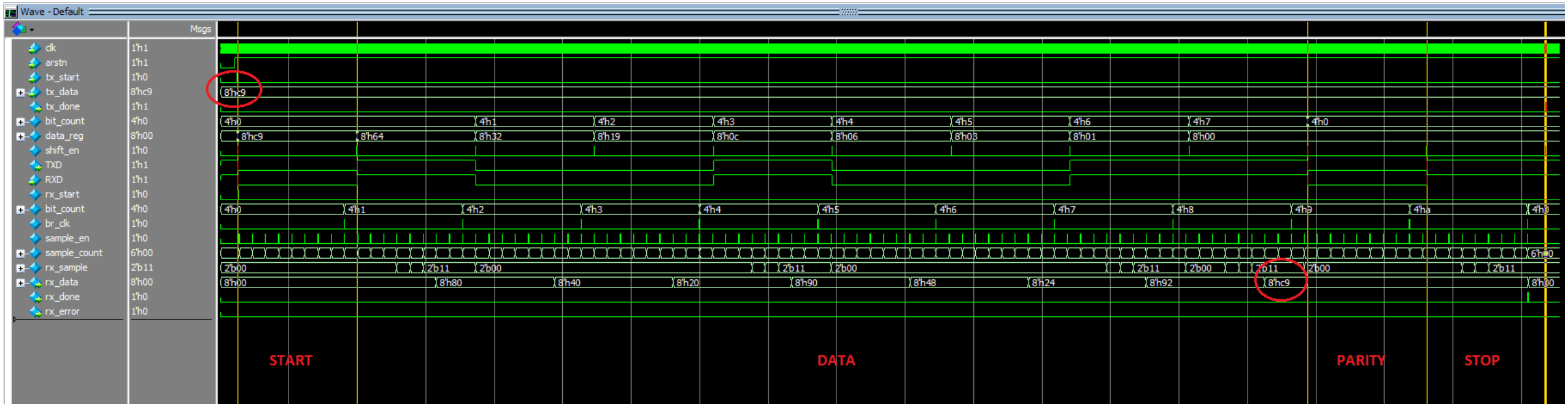
- RS-422 (ANSI/TIA/EIA-422)
  - +/-2V
  - Differential
  - 1 TX, 10 RX
  - Transmission length < 1200m
  - Full-duplex



- RS-485 (ANSI/TIA/EIA-485)
  - +/-1.5V
  - Differential
  - 32 TX, 32 RX
  - Transmission length < 1200m
  - Half-duplex



UART - Example



I2S - Features

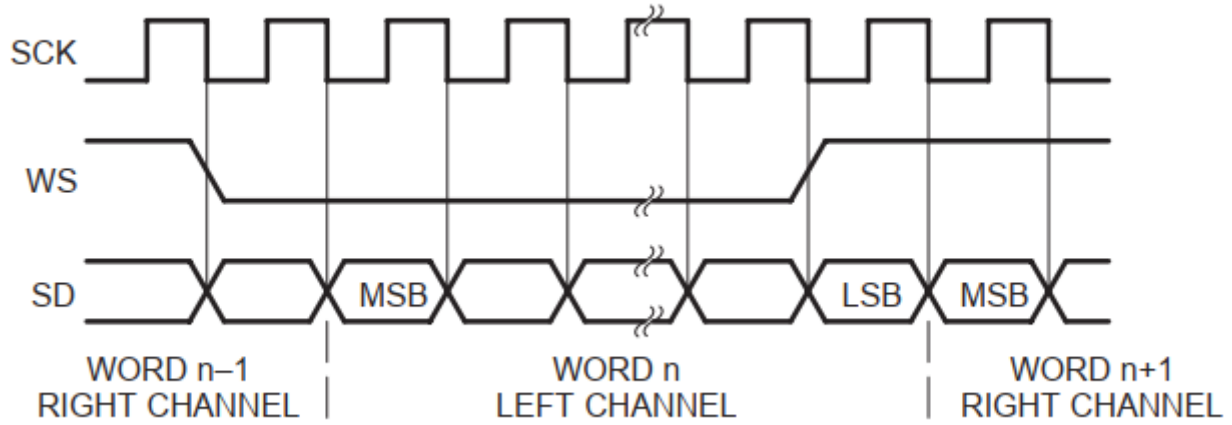
- Inter-IC Sound (Philips (NXP) 1986)
- 1 master & 1 slave
- Only handle audio data
- 3 wires, synchronous, simplex
- Applications: ADC/DAC, CD, digital filters, etc.

I2S - Interface

- SCK (BCLK) : Serial clock (M -> S)
- WS (LRCLK): Channel select (M -> S)
- SD: Serial data (Transmitter -> Receiver)

```
module i2s_master
#(
    parameter   CLK_DIV = 256,
                WS_DIV = 64,
                DATA_WIDTH = 24
)
(
    input   clk, arstn,
    output  reg sck, ws,
    input   sdi,
    output  reg sdo,
    input   [DATA_WIDTH-1:0]    data_send_left, data_send_right,
    output  reg [DATA_WIDTH-1:0]    data_recv_left, data_recv_right
);
```

I2S - Operation



- CLK_DIV: number of system clk periods per sck period
- WS_DIV: number of sck periods per word select period
- DATA_WIDTH: SD data width (MSB to LSB)

I2S - Modes

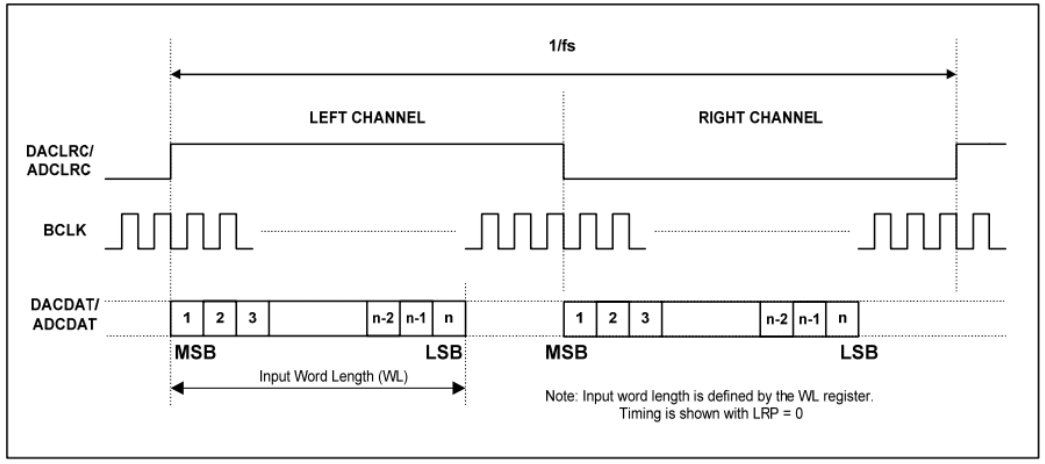


Figure 26 Left Justified Audio Interface (assuming n-bit word length)

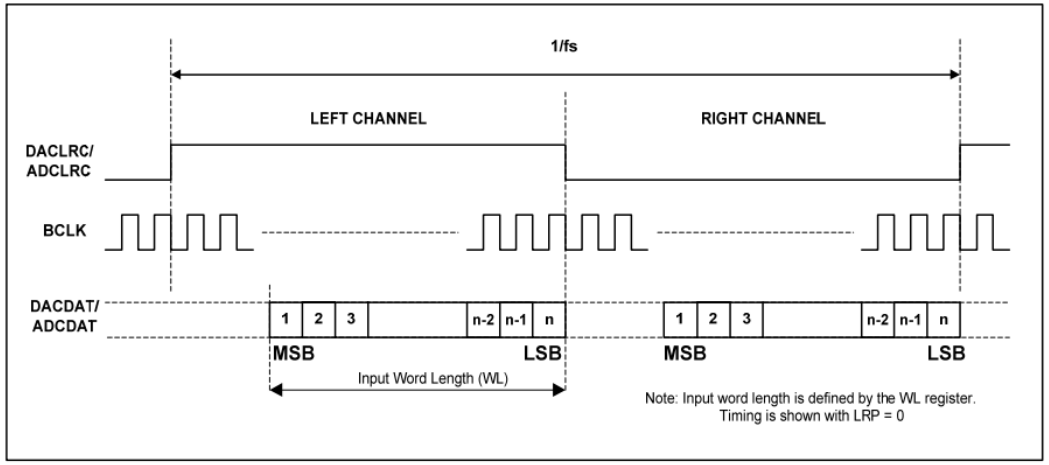


Figure 27 Right Justified Audio Interface (assuming n-bit word length)

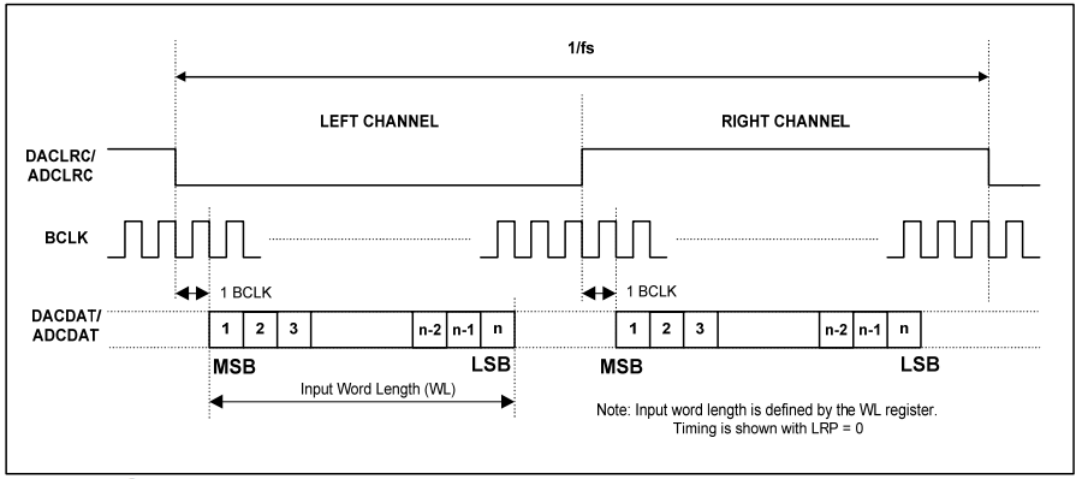


Figure 28 I²S Justified Audio Interface (assuming n-bit word length)

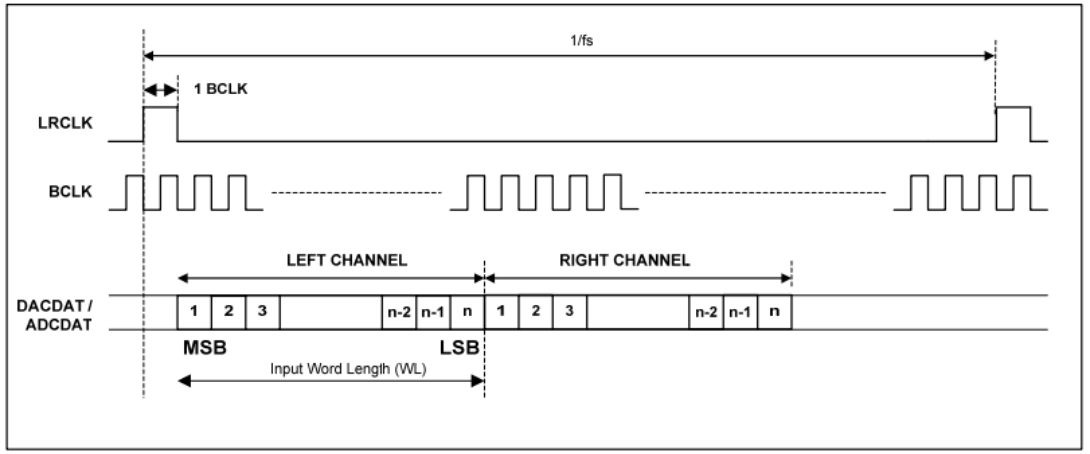
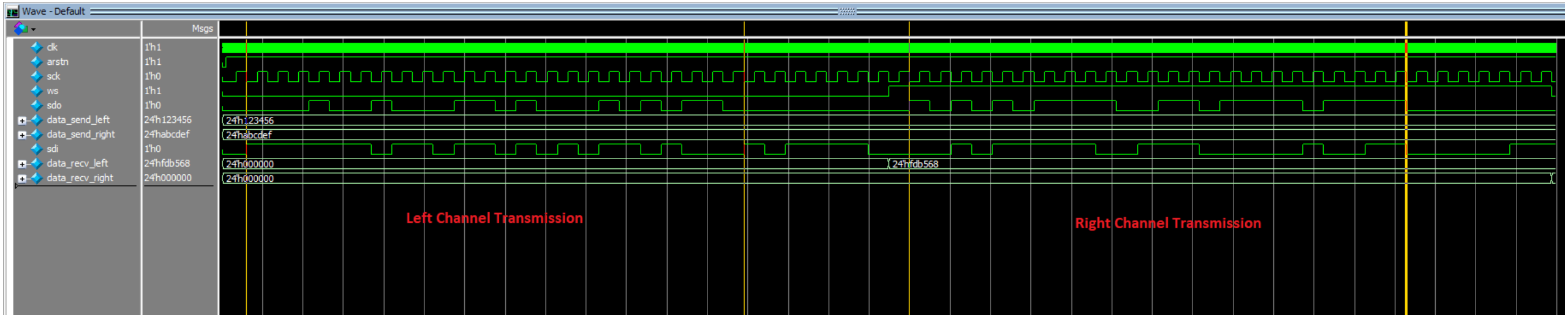


Figure 29 DSP/PCM Mode Audio Interface (mode A, LRP=0, Master)

I2S - Example


Wave - Default
Msgs
clk
arstn
sck
ws
sdo
data_send_left
data_send_right
sdi
data_recv_left
data_recv_right
1'h1
1'h1
1'h0
1'h1
1'h0
24'h123456
24'habcdef
1'h0
24'hfdb568
24'h000000
24'h123456
24'habcdef
24'h000000
24'hfdb568
24'h000000
Left Channel Transmission
Right Channel Transmission

Reference

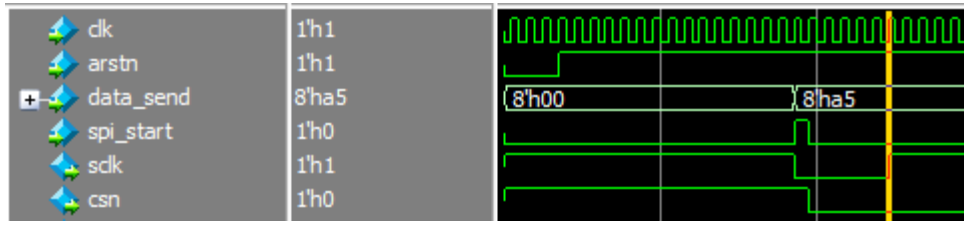
- SPI:
  - Hughes, M. (2017, February 13). Back to Basics: SPI (Serial Peripheral Interface)
  - Winbond Electronics Limited. (2016, August 30). 3V 32M-Bit Serial Flash Memory with Dual, Quad SPI
  - Texas Instruments Incorporated. (2015, April). 16-Bit, 500-kSPS, 4- and 8-Channel, Single-Supply, SAR ADCs with Bipolar Input Range
- I2C:
  - NXP Semiconductors N.V. (2014, April 4). I2C-bus Specification and User Manual
  - Texas Instruments Incorporated. (2015, June). Understanding the I2C Bus
  - NXP Semiconductors N.V. (2010, January 19). 1024 × 8-bit CMOS EEPROM with access protection
- UART:
  - Future Technology Devices International Ltd. (2009, August 7). What is a UART (TN111)
  - Keim, R. (2016, December 20). Back to Basics: The Universal Asynchronous Receiver/Transmitter (UART)
  - Keim, R. (2016, November 23). Convenient, Robust Data Transmission with RS-422 and RS-485
- I2S:
  - Philips Semiconductors. (1996, June 5). I2S Bus Specification
  - Wolfson Microelectronics. (2013, August). Stereo CODEC with 1W Stereo Class D Speaker Drivers and Headphone Drivers for Portable Audio Applications

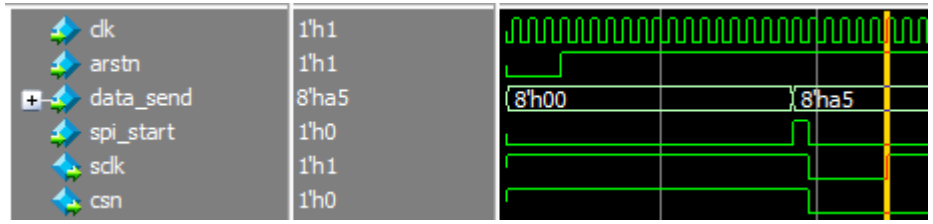
Problem - SPI - Race Condition


```
// sclk
always@(posedge clk or negedge arstn) begin
    if(~arstn)
        sclk <= CPOL;
    else if(CPHA == 1 && spi_start)
        sclk <= ~CPOL;
    else if(clk_count_en && clk_count == FREQ_COUNT)
        sclk <= ~sclk;
    else if(next_state == IDLE)
        sclk <= CPOL;
end
```

```
// clk
initial begin
    clk = 0;
    forever #(CLK_CYCLE/2)  clk = ~clk;
end

// arstn
initial begin
    arstn = 1'b0;
    #(CLK_CYCLE*7/2)    arstn = 1'b1;
end

// spi_start
task gen_spi_start;
begin
    spi_start = 1'b0;
    @(posedge arstn)
        #(CLK_CYCLE*15) spi_start = 1'b1;
        #(CLK_CYCLE)    spi_start = 1'b0;
    @(negedge spi_done)
        #(CLK_CYCLE*15) spi_start = 1'b1;
        #(CLK_CYCLE)    spi_start = 1'b0;
end
endtask
```

```
// clk
initial begin
    clk = 0;
    forever #(CLK_CYCLE/2)  clk = ~clk;
end

// arstn
initial begin
    arstn = 1'b0;
    #(CLK_CYCLE*7/2)    arstn = 1'b1;
end

// spi_start
task gen_spi_start;
begin
    spi_start = 1'b0;
    @(posedge arstn)
        #(CLK_CYCLE*15) spi_start <= 1'b1;
        #(CLK_CYCLE)    spi_start <= 1'b0;
    @(negedge spi_done)
        #(CLK_CYCLE*15) spi_start <= 1'b1;
        #(CLK_CYCLE)    spi_start <= 1'b0;
end
endtask
```

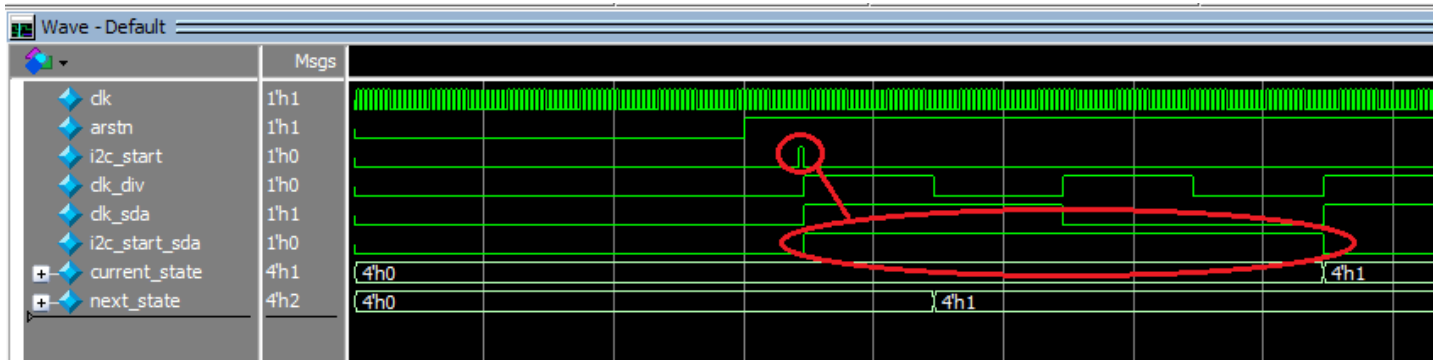




Problem - I2C - CDC

- i2c_start@clk -> i2c_start_sda@clk_sda (CDC):

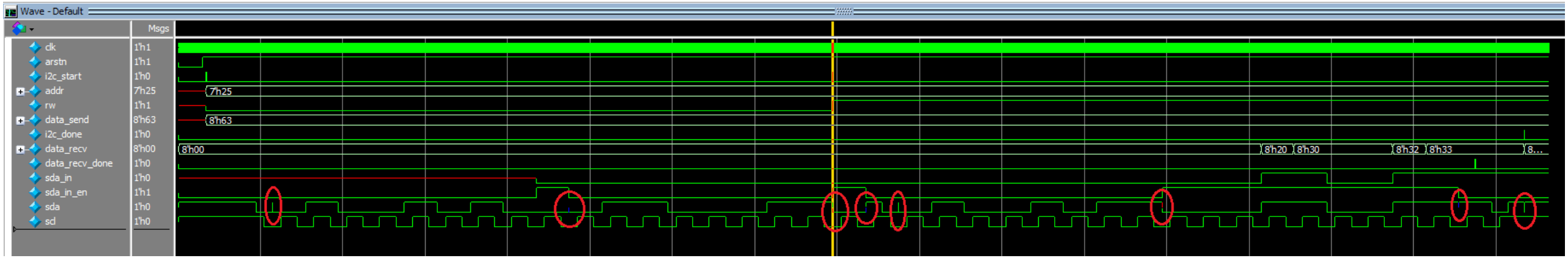
```
// i2c_start_sda (CDC)
always@(posedge clk or negedge arstn) begin
    if(~arstn)
        i2c_start_reg <= 1'b0;
    else if(i2c_start)
        i2c_start_reg <= ~i2c_start_reg;
end

always@(posedge clk_sda or negedge arstn) begin
    if(~arstn) begin
        i2c_start_reg0 <= 1'b0;
        i2c_start_reg1 <= 1'b0;
    end
    else begin
        i2c_start_reg0 <= i2c_start_reg;
        i2c_start_reg1 <= i2c_start_reg0;
    end
end

assign i2c_start_sda = i2c_start_reg0 ^ i2c_start_reg1;
```

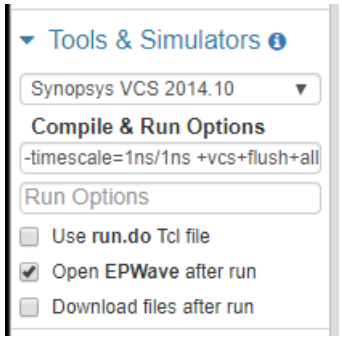


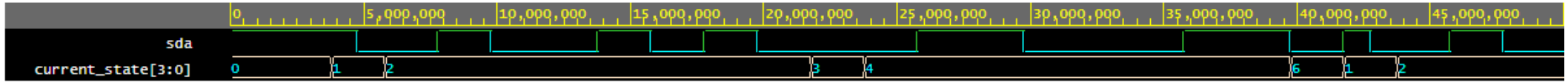
Problem - I2C - Glitch

- SDA glitches:

```
// sda
assign sda = (current_state == SACK1 || current_state == SACK2 || current_state == RD) ? 1'bz : ( (current_state == START) ? clk_sda : ( (current_state == STOP) ? ~clk_sda : sda_reg ) );
```

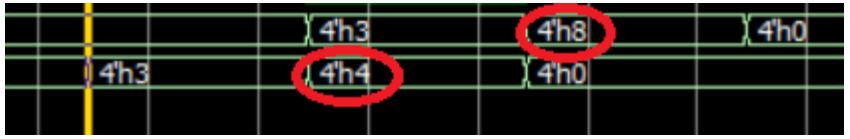




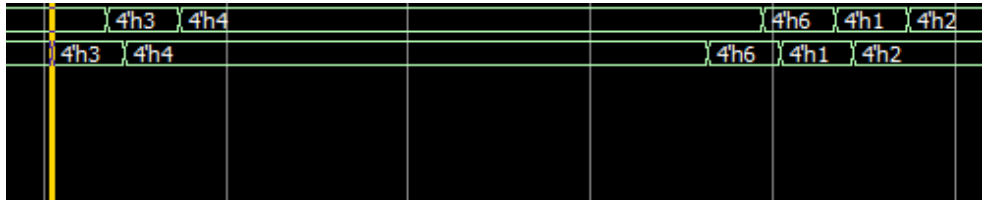


Problem - I2C - FSM

- FSM state transition:



- Solution: FSM next_state logic sampled @negedge clk_div




```
// FSM "combinational"
always@(negedge clk_div) begin
    case(current_state)
        IDLE:   next_state <= i2c_start_sda ? START : IDLE;
        START:  next_state <= CMD;
        CMD:    next_state <= (bit_count == 8) ? SACK1 : CMD;
        SACK1:  begin
            if(sda == 1'b0)
                next_state <= addr_rw[0] ? RD : WR;
            else
                next_state <= STOP;
        end
        WR: next_state <= (bit_count == 8) ? SACK2 : WR;
        RD: next_state <= (bit_count == 8) ? MACK : RD;
        SACK2:  begin
            if(sda == 1'b0) begin
                if(i2c_start_sda)
                    next_state <= addr_rw == {addr,rw} ? WR : START;
                else
                    next_state <= STOP;
            end
            else begin
                next_state <= STOP;
            end
        end
        MACK:   begin
            if(i2c_start_sda)
                next_state <= addr_rw == {addr,rw} ? RD : START;
            else
                next_state <= STOP;
        end
        STOP:   next_state <= IDLE;
        default:    next_state <= IDLE;
    endcase
end
```

Reference

- Verilog Assignments:
  - Cummings, C. (SNUG 2000). Nonblocking Assignments in Verilog Synthesis, Coding Styles That Kill!
  - Cummings, C. (HDLCON 1999). Correct Methods For Adding Delays To Verilog Behavioral Models
- Clock Domain Crossing:
  - Cummings, C. (SNUG 2008). Clock Domain Crossing (CDC) Design & Verification Techniques Using SystemVerilog