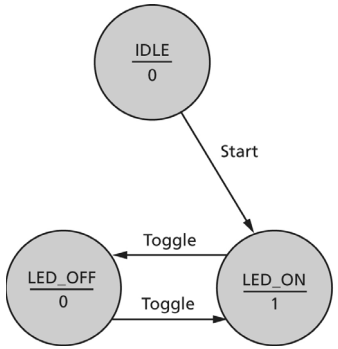
Lecture 1: FSMs & Digital interfaces (UART, SPI, I2C, PWM)

By: Emad Samuel Malki Ebeid

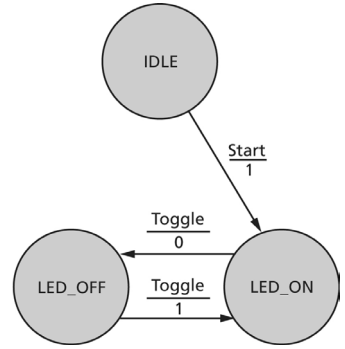
FSM

Moore State Machine



The outputs of a Moore state machine depend **only on the present state**. The outputs are written **only** when the state changes (on the clock edge).

Mealy State Machine

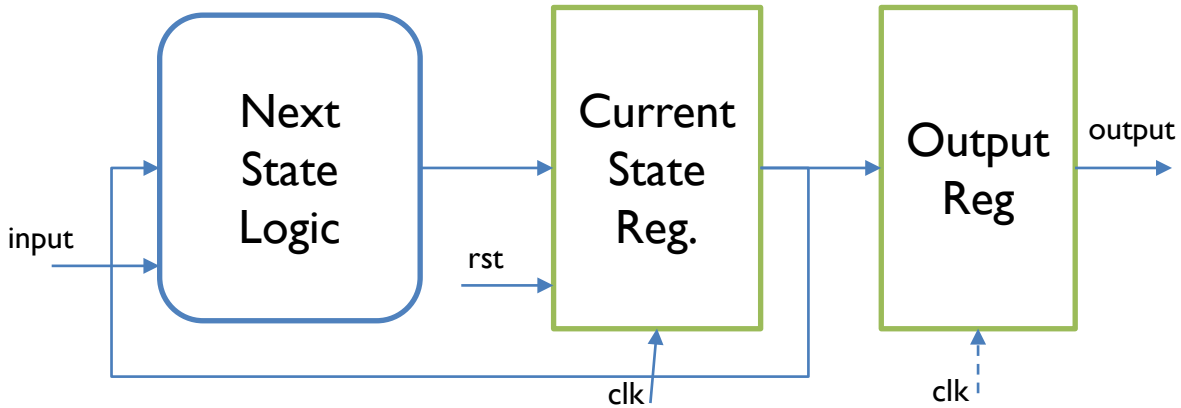


The outputs of a Mealy state machine depend on **both the inputs and the current state**. When the inputs change, the outputs are updated **without** waiting for a clock edge.

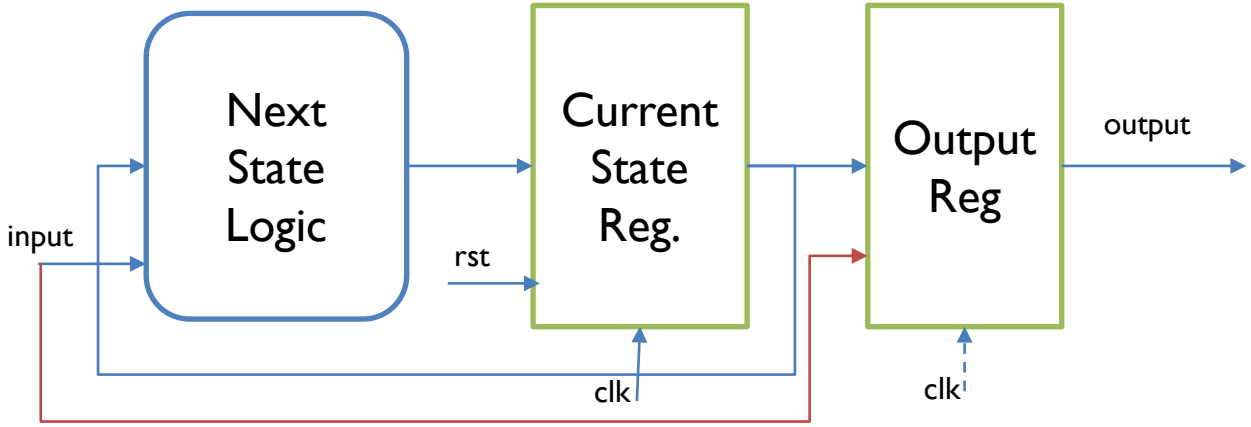
Because modern designs are generally synchronous, the **Moore** option tends to be preferred – *Finite State Machine in Hardware* book by Volnei A. Pedroni

FSM in hardware

Moore

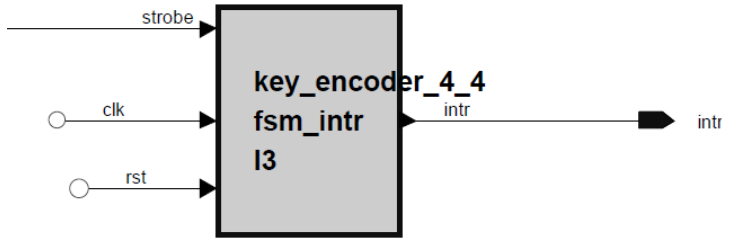


Mealy

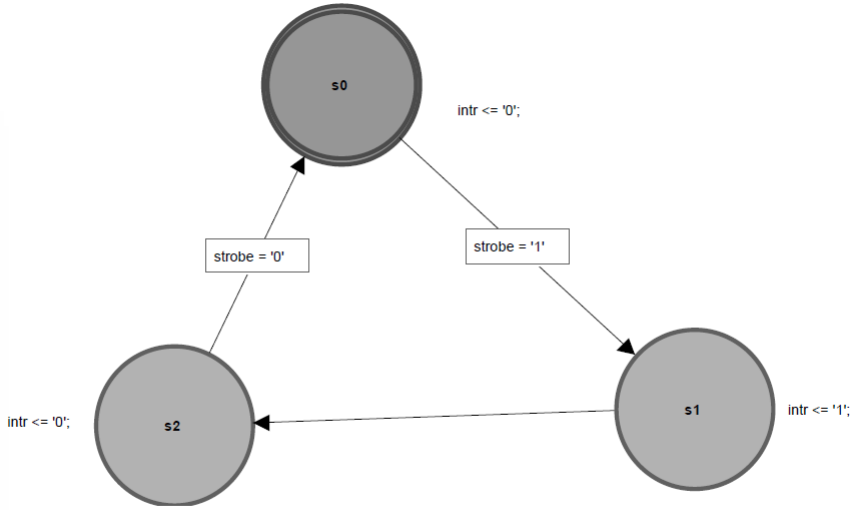


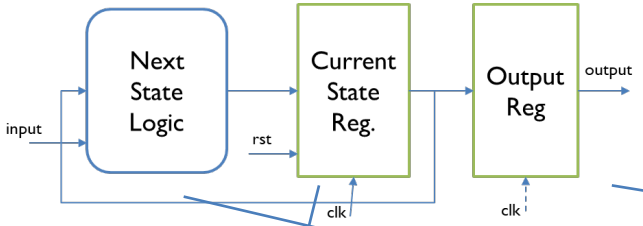
FSM examples: debouncer

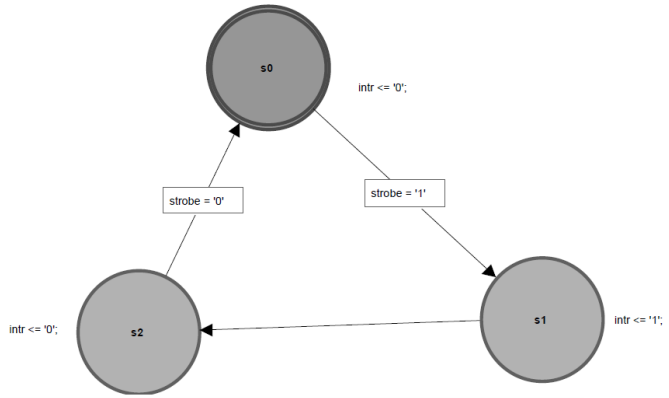
```
ENTITY fsm_intr IS
    PORT(
        clk     : IN      std_logic;
        rst     : IN      std_logic;
        strobe  : IN      std_logic;
        intr    : OUT     std_logic
    );

-- Declarations

END fsm_intr ;
```




```
ARCHITECTURE fsm OF fsm_intr IS

    -- Architecture Declarations
    TYPE STATE_TYPE IS (s0, s1, s2); -- Define the states




    -- Declare current and next state signals
    SIGNAL current_state : STATE_TYPE ; -- Create a signal that uses the different states
    SIGNAL next_state : STATE_TYPE ;

BEGIN
```



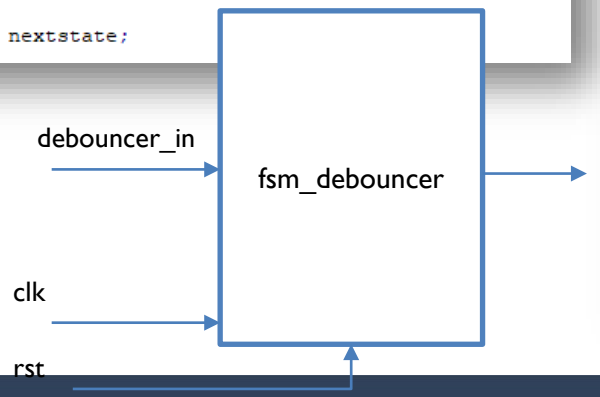





```
-------------------------------------------------------------
clocked : PROCESS(clk, rst)
-------------------------------------------------------------
BEGIN
   IF (rst = '1') THEN
      current_state <= s0;
      -- Reset Values
   ELSIF (rising_edge(clk)) THEN
      current_state <= next_state;
      -- Default Assignment To Internals

      -- Combined Actions for internal signals only
--    CASE current_state IS
--    WHEN s0 =>
--       reg<= data_in & '1';
--    WHEN s2 =>
--       count<= count +1;
--    WHEN OTHERS =>
--       NULL;
--   END CASE;
   END IF;

END PROCESS clocked;
```

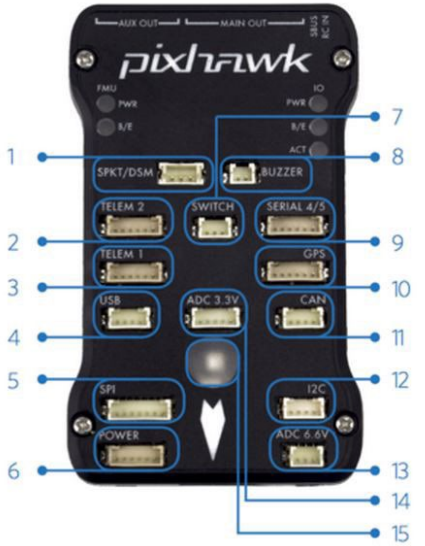
```
-------------------------------------------------------------
nextstate : PROCESS (current_state, strobe)
-------------------------------------------------------------
BEGIN
   CASE current_state IS
   WHEN s0 =>
      IF (strobe = '1') THEN
         next_state <= s1;
      ELSE
         next_state <= s0;
      END IF;
   WHEN s1 =>
         next_state <= s2;
   WHEN s2 =>
      IF (strobe = '0') THEN --IF (count = 10) THEN
         next_state <= s0;
      ELSE
         next_state <= s2;
      END IF;
   WHEN OTHERS =>
      next_state <= s0;
   END CASE;

END PROCESS nextstate;
```

```
   -------------------------------------------
   output : PROCESS (current_state)
   -------------------------------------------
   BEGIN
      -- Default Assignment
      intr <= '0';
      -- Default Assignment To Internals

      -- Combined Actions
      CASE current_state IS
      WHEN s0 =>
         intr <= '0';
      WHEN s1 =>
         intr <= '1';
      WHEN s2 =>
         intr <= '0';
      WHEN OTHERS =>
         NULL;
      END CASE;

   END PROCESS output;

   -- Concurrent Statements

END fsm;
```



Digital interfaces
(PWM, UART, SPI)

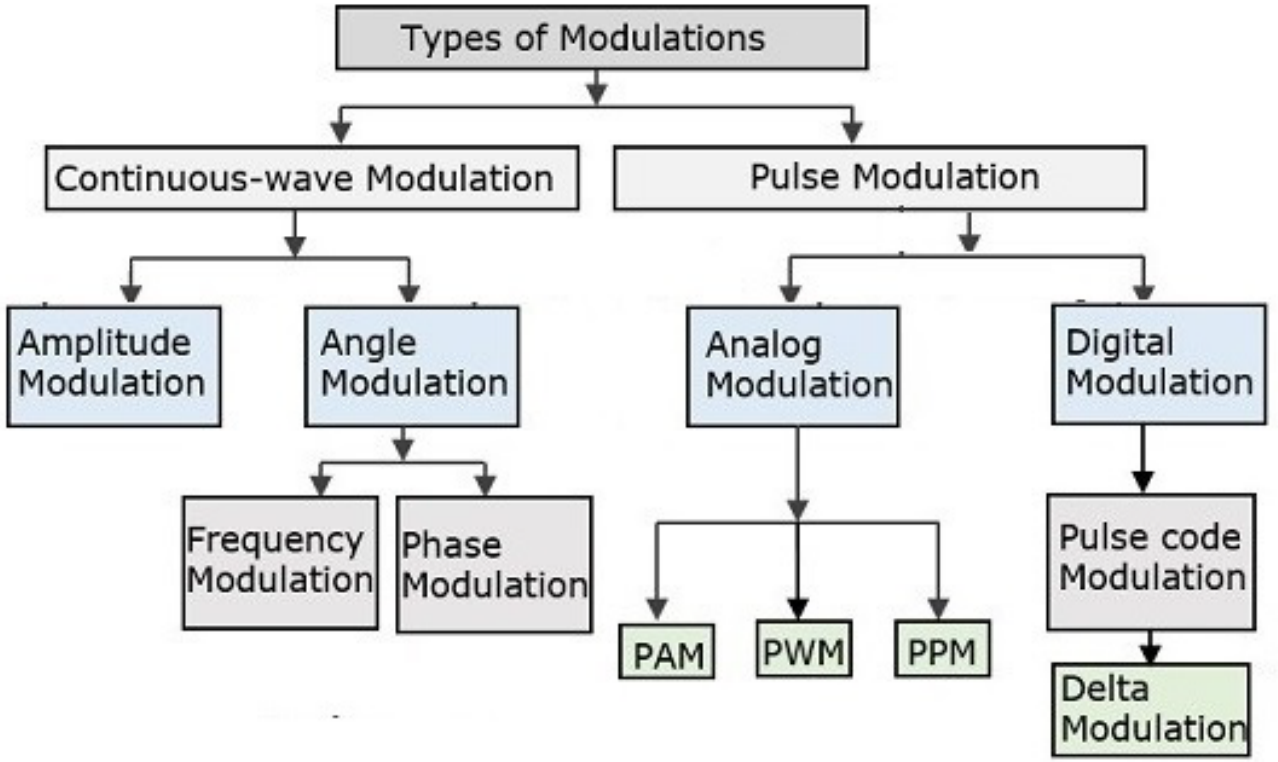
Communication interfaces



1 Spektrum DSM receiver
2 Telemetry (radio telemetry)
3 Telemetry (on-screen display)
4 USB
5 SPI (serial peripheral interface) bus
6 Power module
7 Safety switch button
8 Buzzer
9 Serial
10 GPS module
11 CAN (controller area network) bus
12 I²C splitter or compass module
13 Analog to digital converter 6.6 V
14 Analog to digital converter 3.3 V
15 LED indicator

Communication technologies

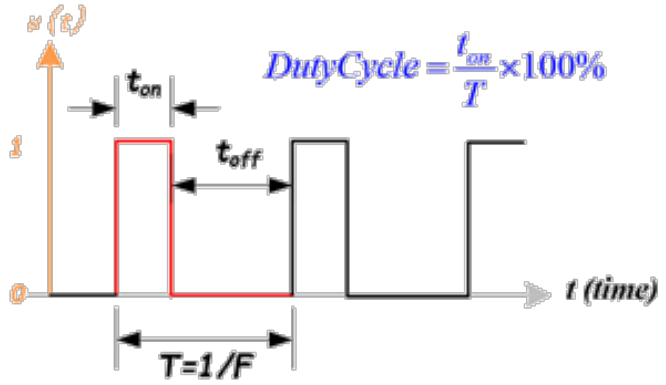
- Analog and digital communication

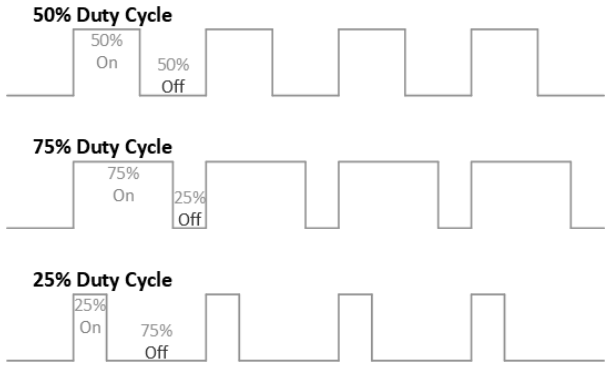


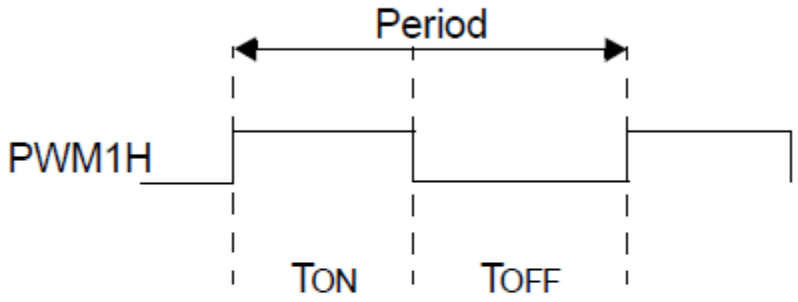
Pulse-width modulation (PWM)

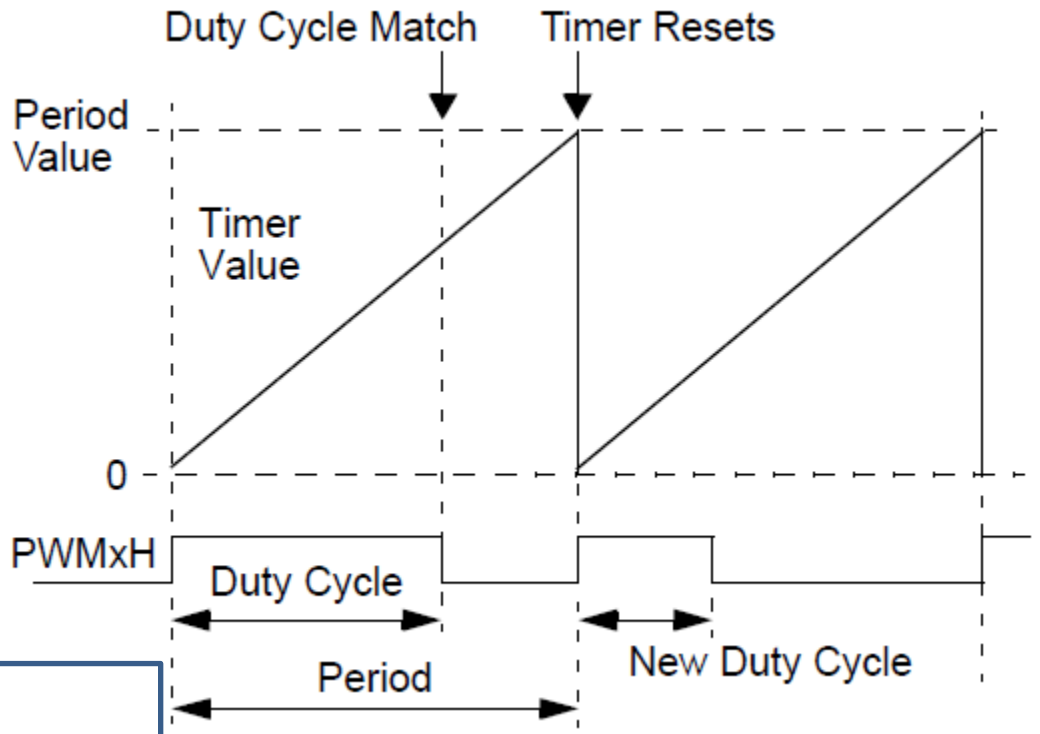
PWM signal

- PWM shows how much time the signal is high in an analog fashion.
- The signal "t_{on}" is considered when the signal is high.
- To describe the amount of "t_{on}", the concept of duty cycle is used.
- The duty cycle is the percentage of time a digital signal is on over an interval or period of time.





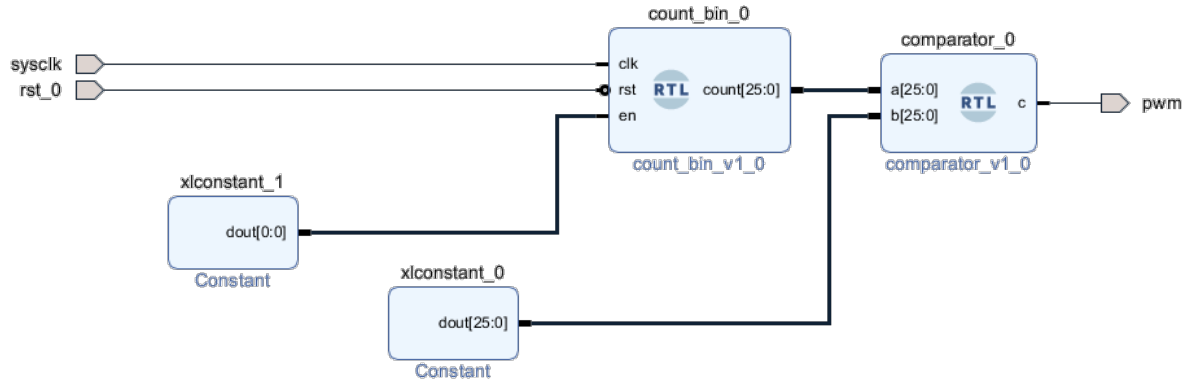




10 minutes to think about how to build PWM circuit.

Excercise

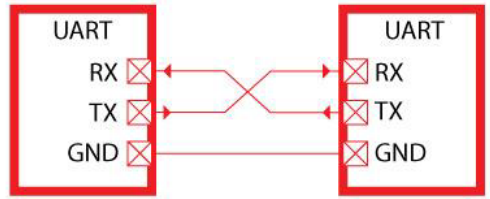
- How many counter bits are needed to generate a PWM signal that has a 1Hz frequency out of a 125MHz clock signal?
- Which is the comparator threshold value for generating a PWM with 50% duty cycle?
- Which is the comparator threshold value for generating a PWM with 10% duty cycle?



PWM application: Clock divider

- It provides an output clock signal that is a divided frequency of the input.
- It is a PWM signal with a 50% duty cycle.
- The previous exercise is a clock divider. It divided a clock of125MHz to 1Hz.
- Clock dividers are needed to align the input frequency with the desire operating frequency of an application or an interface.
  - For the digital clock assignment, 1Hz is the desired operating frequency of the circuit.

Serial interface: Universal Asynchronous Receiver/Transmitter

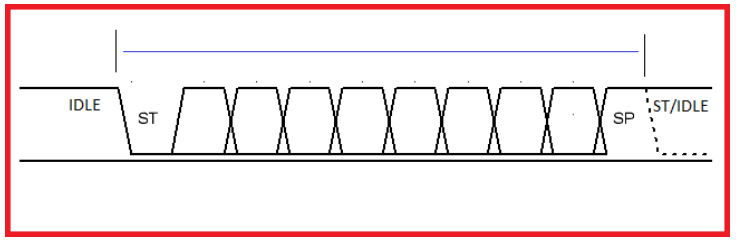


Serial communication introduction

- Serial port is a serial communication physical interface through which information transfers in or out one bit at a time. Data transfer through serial ports connected the computer to devices such as terminals and various peripherals.

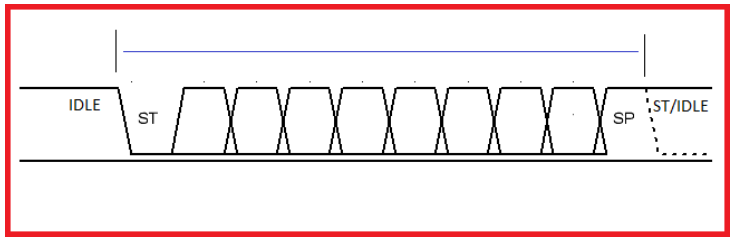
- Some computers, such as the IBM PC, used an integrated circuit called a UART, that converted characters to (and from) asynchronous serial form, and automatically looked after the timing and framing of data. A universal asynchronous receiver/transmitter (usually abbreviated UART) is a type of "asynchronous receiver/transmitter", a piece of computer hardware that translates data between parallel and serial forms

Baudrate



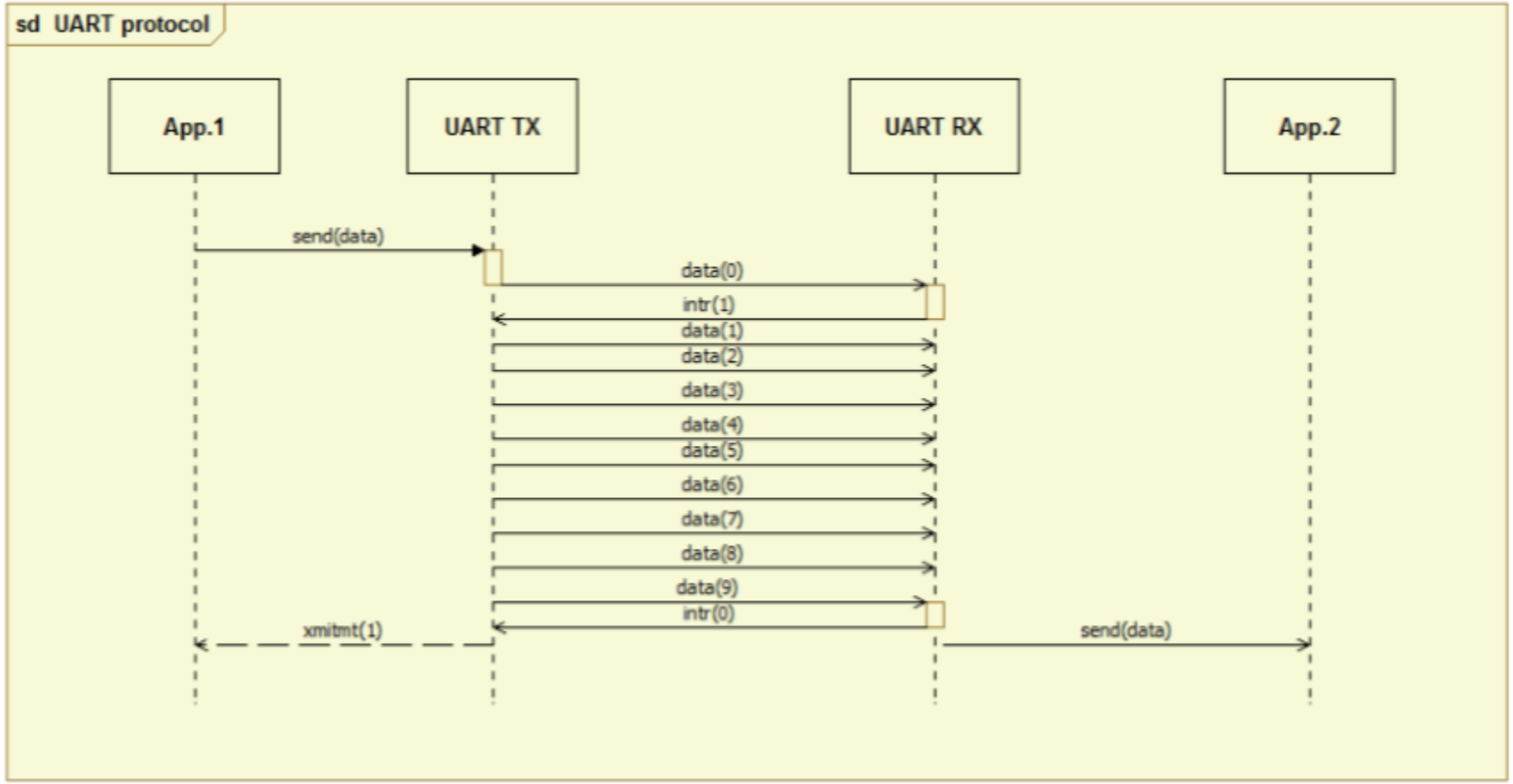
- **Baudrate**: In embedded designs, it is necessary to choose a proper oscillator to get the correct baud rate with little or no error. Some examples of common crystal frequencies and baud rates with no errors are: 300, 600, 1200, 1800, 2400, 4800, 7200, 9600, 14400, 19200, 38400, 57600, 115200 Bd
- **Data bits:** The number of data bits in each character can be 5 (for Baudot code), 6 (rarely used), 7 (for true ASCII), 8 (for any kind of data, as this matches the size of a byte), or 9 (rarely used). 8 data bits are almost universally used in newer applications. 5 or 7 bits generally only make sense with older equipment such as teleprinters. Most serial communications designs send the data bits within each byte LSB (Least Significant Bit) first. This standard is also referred to as "little endian". Also, possible, but rarely used, is "big endian" or MSB (Most Significant Bit) first serial communications. The order of bits is not usually configurable, but data can be byte-swapped only before sending.

Parity and stop bits

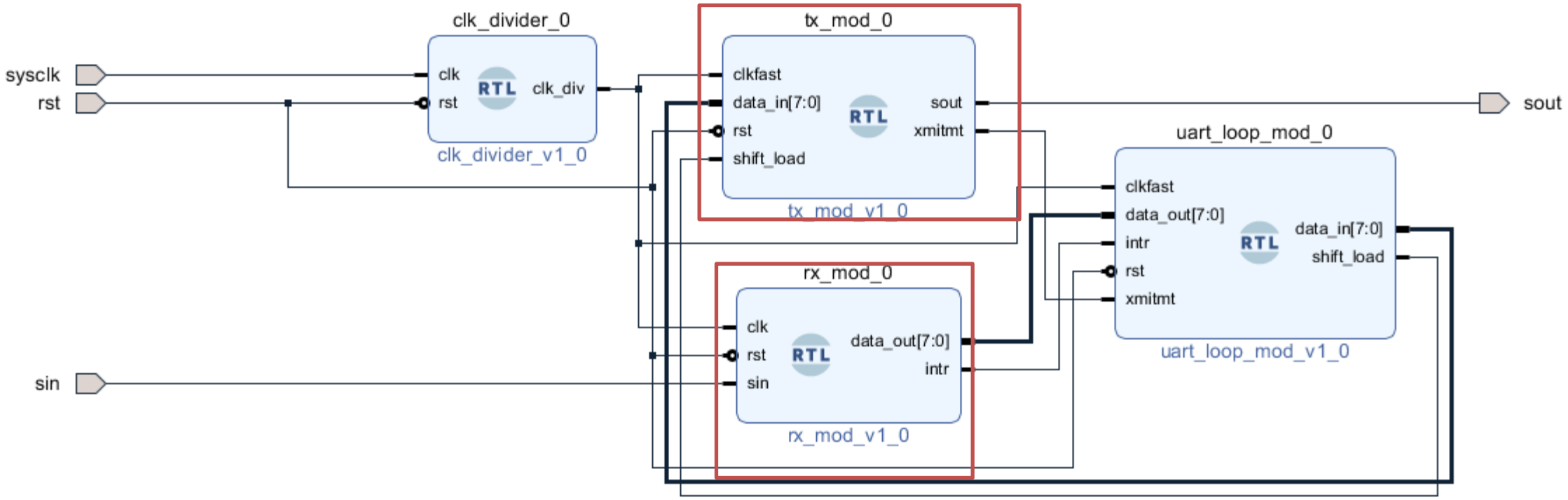


- **Parity:** Parity is a method of detecting errors in transmission. When parity is used with a serial port, an extra data bit is sent with each data character, arranged so that the number of 1 bits in each character, including the parity bit, is always odd or always even. If a byte is received with the wrong number of 1's, then it must have been corrupted. However, an even number of errors can pass the parity check.
- **Stop bits:** Stop bits sent at the end of every character allow the receiving signal hardware to detect the end of a character and to resynchronise with the character stream. Electronic devices usually use one stop bit. If slow electromechanical teleprinters are used, one-and-one half or two stop bits are required.

UART protocol



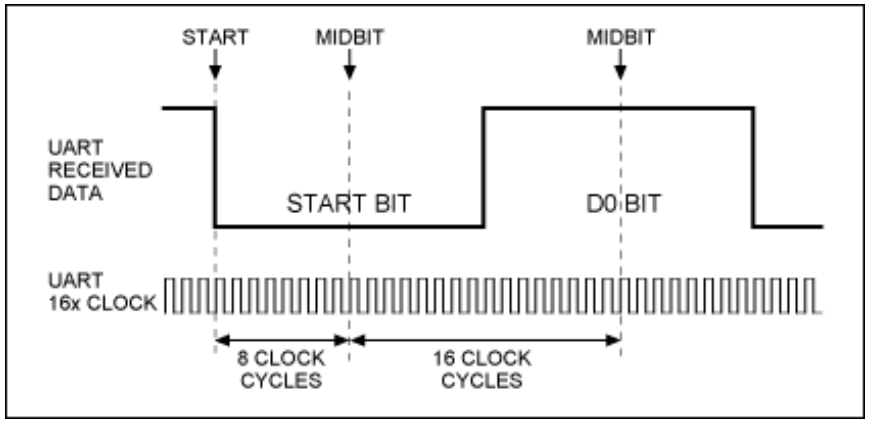
Overall system



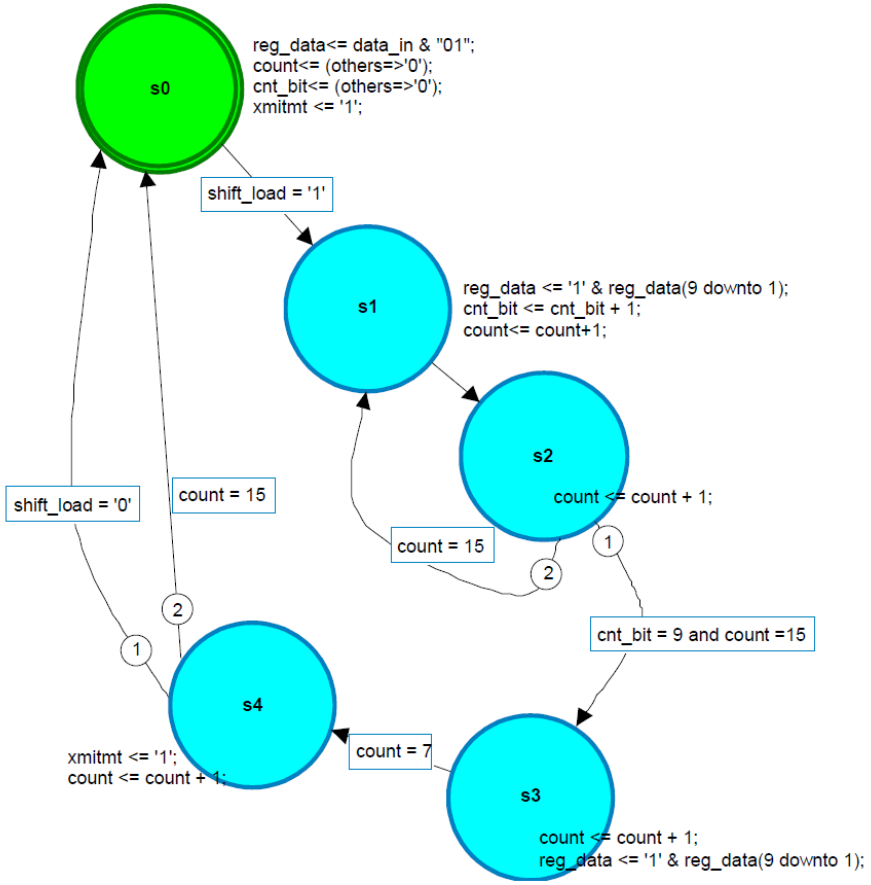
- The code is in BB

Receiver

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;


ENTITY rx_mod IS
   PORT(
      clk      : IN     std_logic;
      rst      : IN     std_logic;
      sin      : IN     std_logic;
      data_out : OUT    std_logic_vector (7 DOWNTO 0);
      intr     : OUT    std_logic);

END rx_mod ;

-- hds interface_end
ARCHITECTURE rtl OF rx_mod IS
    signal rxreg: std_logic_vector(9 downto 0);
    signal count: unsigned (3 downto 0);
    signal rxmt: std_logic;
    signal rxin,start_flag: std_logic;
    begin
        process (clk, rst)
            begin
                if (rst = '0') then
                    count <= (others => '0');
                    rxmt <= '1';
                    rxreg <= (others => '1');
                    intr <= '0';
                    rxin <= '1';
                    start_flag<='0';
                elsif (rising_edge(clk)) then
                    rxin<=sin;
                    if (rxmt = '1' and rxin = '0') then
                        count <= (others => '0');
                        rxmt <= '0';
                        rxreg <= (others => '1');
                        start_flag<='0';
                    elsif (count = 7 and rxmt = '0' and rxin = '0' and start_flag='0') then
                        rxreg <= rxin & rxreg(9 downto 1);
                        count <= (others => '0');
                        start_flag<='1';
                    elsif (count = 15 and rxmt = '0') then
                        rxreg <= rxin & rxreg(9 downto 1);
                        count <= count + 1;
                    else
                        count <= count + 1;
                    end if;
                    if (rxmt = '0' and rxreg(9) = '1' and rxreg(0) = '0') then
                        intr <= '1';
                        rxmt <= '1';
                    else
                        intr <= '0';
                    end if;
                end if;
        end process;
        data_out <= rxreg(8 downto 1);
END rtl;
```

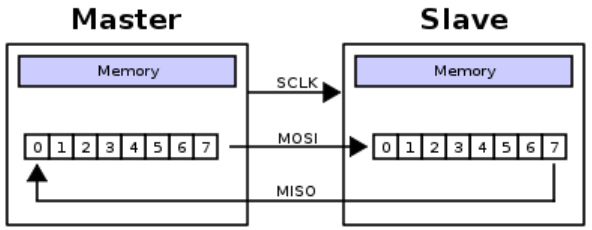


Transmitter



Serial Peripheral Interface (SPI)

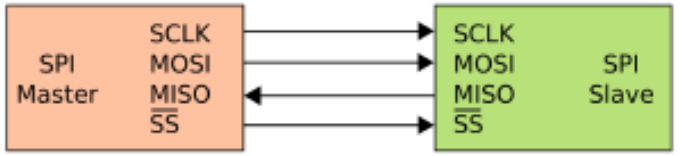
SPI

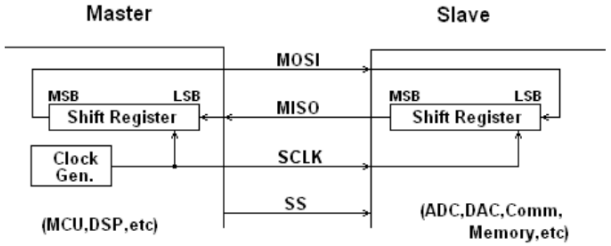


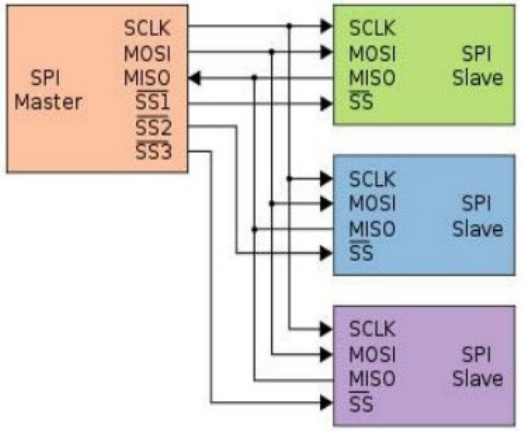
- Serial Peripheral Interface (SPI) is a synchronous serial communication interface for short-distance communication.
- Full duplex mode using a master-slave architecture with a single master.
- SPI communication was used to connect devices such as printers, cameras, scanners, etc. to a desktop computer; nowadays it is replaced by USB.
- In robotics, SPI is still utilized for connecting sensors, actuators, etc.
- SPI runs using a master/slave set-up and can run in full duplex mode (i.e., signals can be transmitted between the master and the slave simultaneously).
- When multiple slaves are present, SPI requires no addressing to differentiate between these slaves.
- There is no standard communication protocol for SPI.

SPI

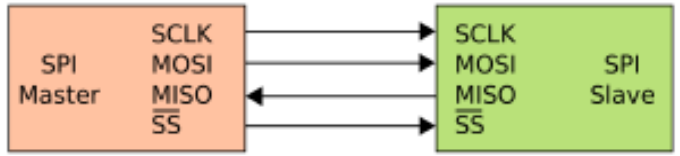
- SPI works by shifting bits between two registers of the same word size. Data is exchanged through the two connections, MOSI and MISO.
- One bit is shifted for each cycle of the serial clock.
- This yields a constant transfer rate. The first bit delivered is the most significant bit and the least significant bit is the last.
  - SCLK is the serial clock.
  - MOSI is the output from the master to the slave.
  - MISO is the output from the slave to the master.
  - SS is the slave select, which is used to wake up the slave.

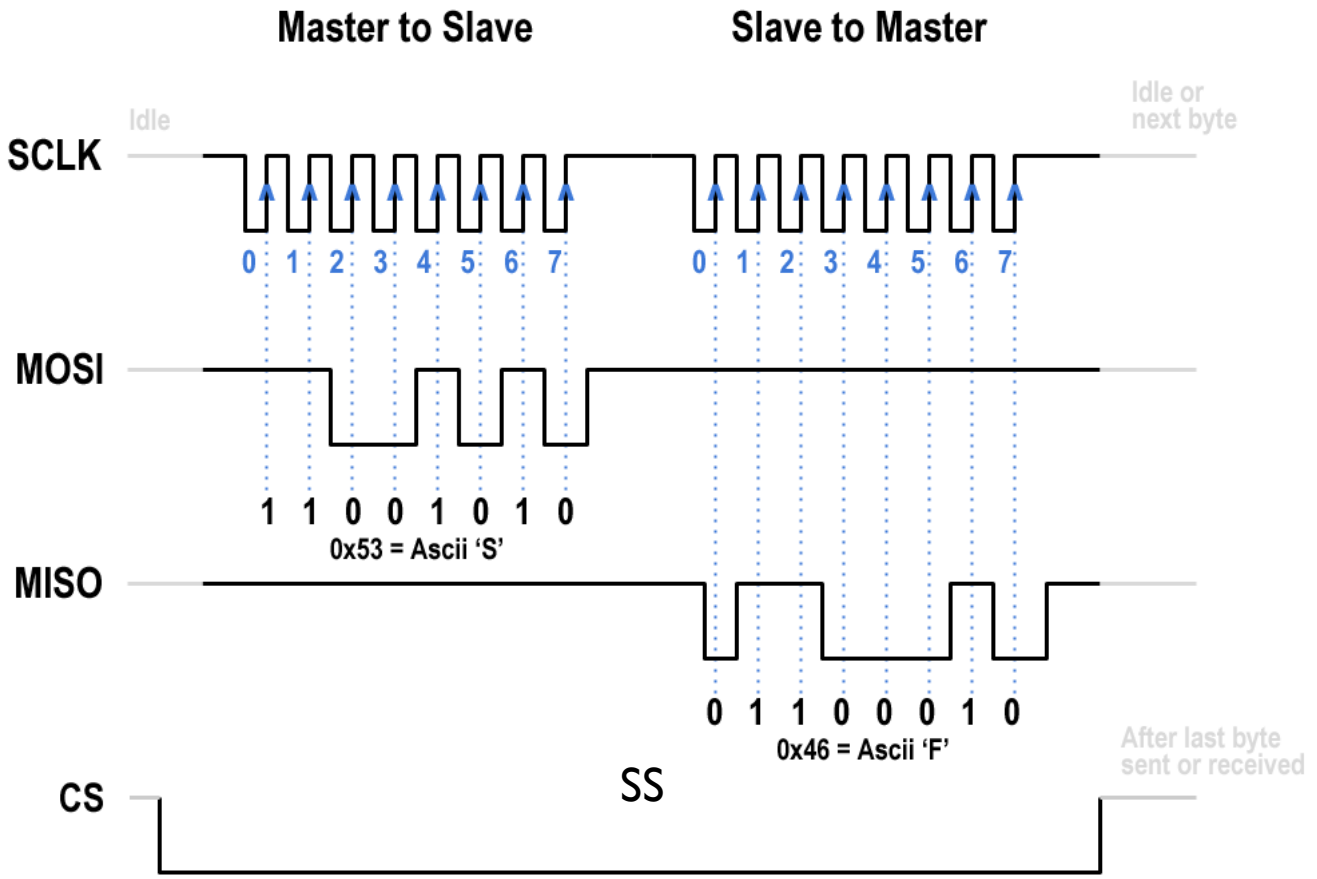






How it works

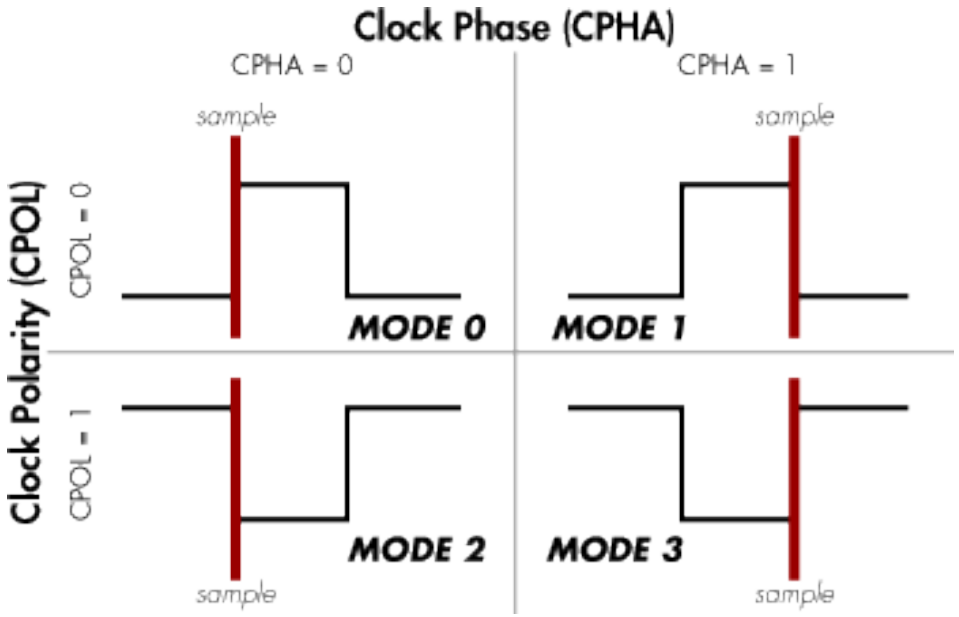

SPI
Master
SCLK
MOSI
MISO
SS
SCLK
MOSI
MISO
SS
SPI
Slave


Master to Slave
Slave to Master
Idle
Idle or
next byte
SCLK
0 1 2 3 4 5 6 7
0 1 2 3 4 5 6 7
MOSI
1 1 0 0 1 0 1 0
0x53 = Ascii 'S'
MISO
0 1 1 0 0 0 1 0
0x46 = Ascii 'F'
After last byte
sent or received
CS
SS

SPI Modes

- It aims at adding more flexibility to the communication channel between the master and slave.
- Its a combination of polarity and phases (CPOL, CPHA)
  - CPOL (**C**lock **POL**arity) and CPHA (**C**lock **PHA**se)

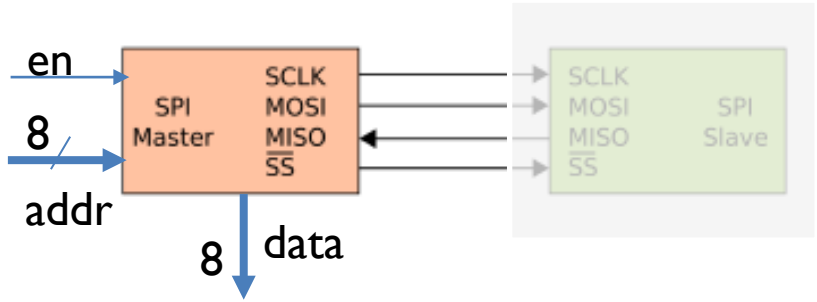
Mode	CPOL	CPHA
0	0	0
1	0	1
2	1	0
3	1	1

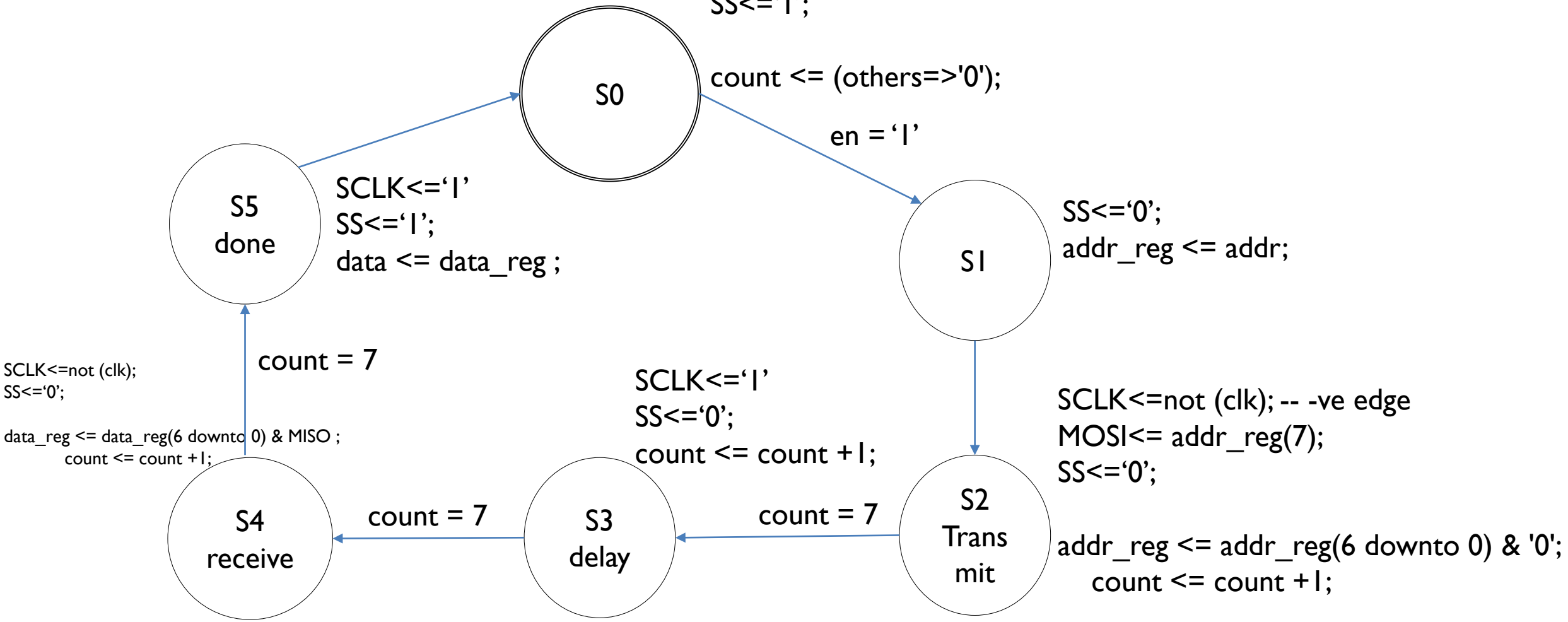


Internal signals:

addr_reg, data_reg: std_logic_vector(7 downto 0) ;
count: unsigned(2 downto 0) ;

SPI FSM: Master





Lab Excercise: PWM

- Design and simulate the following PWM signal generator in Vivado.
- Change the frequency and duty cycles and observe the signal outputs.