SHIFT REGISTERS

Shift registers load the data present on its inputs and then moves or shifts it to its output once in every clock cycle.

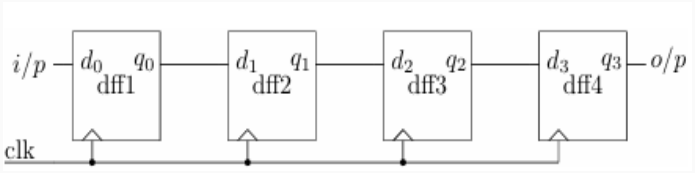
1. Serial Input Serial Output (SISO)
2. Serial Input Parallel Output (SIPO)
3. Parallel Input Serial Output (PISO)
4. Parallel Input Parallel Output (PIPO)

Serial Input Serial Output (SISO)

SISO is a type of shift register which receives a serial stream of bits or bytes and shifts the stream serially.

$T_{input} = n\ t_{ff}$ $T_{output} = n\ t_{ff}$

$T_{latency} = (n-1) \cdot t_{ff}$



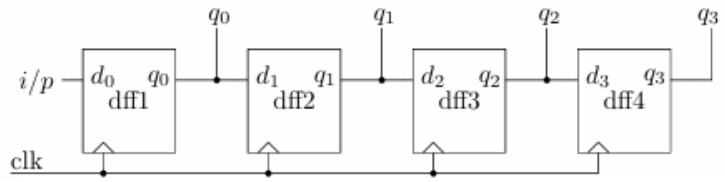

```
module SISO #(parameter WIDTH = 4)(
   input wire clk,        // Clock signal
   input wire rst,        // Asynchronous reset
   input wire serial_in,    // Serial data input
   output reg serial_out    // Serial data output
);
 // Internal register to hold the shift register values
   reg [WIDTH-1:0] shift_reg;

   always @(posedge clk or posedge rst)
 begin
      if (rst) begin
         shift_reg <= 0;       // Reset all bits to 0
        serial_out <= 0;       // Reset output to 0
      end else
         begin
         serial_out <= shift_reg[WIDTH-1];
                // Output the MSB
        shift_reg <= {shift_reg[WIDTH-2:0], serial_in};
             // Shift left and insert new input
    end
   end
endmodule
```

Serial Input Parallel Output (SIPO)

SIPO is a type of shift register which is used to convert a serial data stream into a parallel data stream.

$T_{input} = nt_{ff}$ $Toutput = t_{ff}$

$T_{latency} = n \cdot t_{ff}$



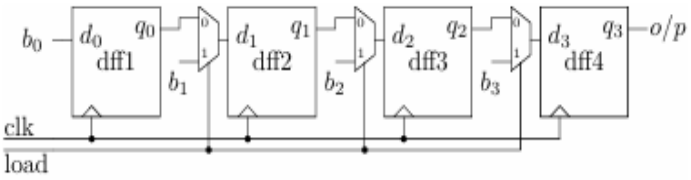

```
module SIPO #(parameter WIDTH = 4)(
   input wire clk,        // Clock signal
   input wire rst,        // Asynchronous reset (active
high)
   input wire serial_in,    // Serial data input
   output reg [WIDTH-1:0] parallel_out // Parallel data
output
);
 // Internal register to hold the shift register values
   reg [WIDTH-1:0] shift_reg;
always @(posedge clk or posedge rst)
   begin
      if (rst) begin
         shift_reg <= 0;        // Reset all bits to 0
      parallel_out <= 0;      // Reset parallel output to 0
     end
 else begin
 shift_reg <= {shift_reg[WIDTH-2:0], serial_in};
    // Shift left
  parallel_out <= shift_reg;   // Update parallel output
      end
   end
endmodule
```

PISO (Parallel-In Serial-Out)

PISO takes a parallel data stream as input and outputs a serial stream of data. In integrated circuits, due to the limitation of input and output ports parallel data stream is converted to the serial data stream.

$T_{input}=t_{ff}$ $T_{output}=n\cdot t_{ff}$

$t_{latency}=t_{ff}$



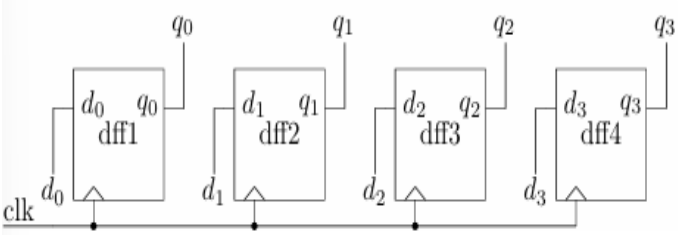

```
module PISO (
  input clk,          // Clock signal
  input rst_n,        // Active-low reset
  input load,         // Load signal (to load parallel data)
  input [7:0] parallel_in,// 8-bit parallel input
  output reg serial_out  // Serial output            );
 reg [7:0] shift_reg;     // 8-bit shift register
always @(posedge clk or negedge rst_n)
 begin
  if (~rst_n) begin
    shift_reg <= 8'b0;  // Reset the shift register
    serial_out <= 0;    // Reset serial output
  end else if (load) begin
    shift_reg <= parallel_in; // Load parallel data
    serial_out <= shift_reg[7]; // Output MSB initially
  end
else begin
    serial_out <= shift_reg[7]; // Output MSB of the register
    shift_reg <= shift_reg << 1; // Shift left
  end
end
endmodule
```

PIPO (Parallel-In Parallel-Out)

In PIPO a parallel data stream is input and a parallel data stream is output. PIPO is most popularly known as pipeline register or simply as register. PIPO is used for storing a data for one clock cycle or delaying a data by one clock cycle.

$T_{input}=t_{ff}$ $Toutput=t_{ff}$

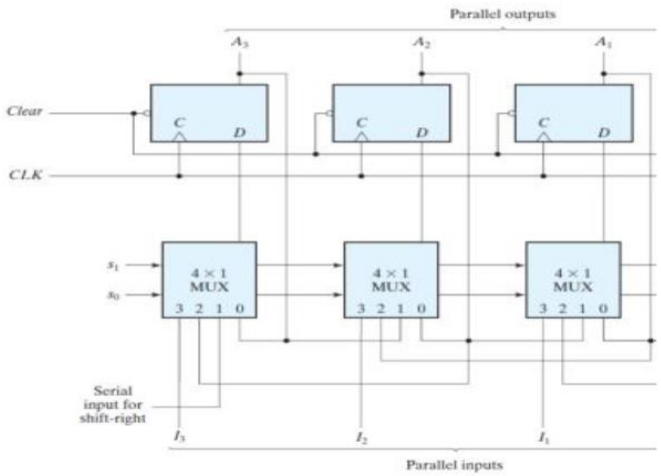
$T_{latency}=t_{ff}$




```
module PIPO (
  input clk,          // Clock signal
  input rst_n,        // Active-low reset
  input load,         // Load signal (to load parallel data)
  input [7:0] parallel_in,// 8-bit parallel input
  output reg [7:0] parallel_out // 8-bit parallel output
);reg [7:0] shift_reg;     // 8-bit shift register
always @(posedge clk or negedge rst_n) begin
  if (~rst_n) begin
    shift_reg <= 8'b0;  // Reset the shift register
    parallel_out <= 8'b0; // Reset parallel output
  end else if (load) begin
    shift_reg <= parallel_in; // Load parallel data
  end else begin
    parallel_out <= shift_reg;
  end
end
endmodule
```

Universal Shift Register

```
module UniversalShiftRegister (
  input clk,          // Clock signal
  input rst_n,        // Active-low reset
  input shift_left,     // Shift left control
  input shift_right,    // Shift right control
  input load,         // Load control
  input [3:0] parallel_in,// 4-bit parallel input
  output reg [3:0] q    // 4-bit parallel output
);
always @(posedge clk or negedge rst_n) begin
  if (~rst_n) begin
    q <= 4'b0;  // Reset the shift register
  end else if (load) begin
    q <= parallel_in; // Load parallel data into register
  end else if (shift_left) begin
    q <= q << 1; // Shift left
  end else if (shift_right) begin
    q <= q >> 1; // Shift right
  end
end
endmodule
```



- ✓ IIT KHARAGPUR **: PROF. INDRANIL SENGUPTA** (HARDWARE MODELING USING VERILOG)

  https://www.youtube.com/watch?v=NCrlyaXMAn8&list=PLJ5C_6qdAvBELELTSPgzYkQg3HgclQh-5

- ✓ **ANKIT GOYAL SIR**: DIGITAL ELECTRONICS
  https://youtube.com/playlist?list=PLs5_Rtf2P2r41iuDKULDHHnIwfXyTAxBH&si=PhMhOvo8_rT1a53y

- ✓ BOOKS:

  - DIGITAL ELECTRONICS: **Digital Logic and Computer Design by M. Morris Mano.**
  - VERILOG : **Verilog HDL - Samir Palnitkar**
  - Digital Design With an Introduction to the Verilog HDL, VHDL, and SystemVerilog by **M. Morris Mano and Michael D. Ciletti**