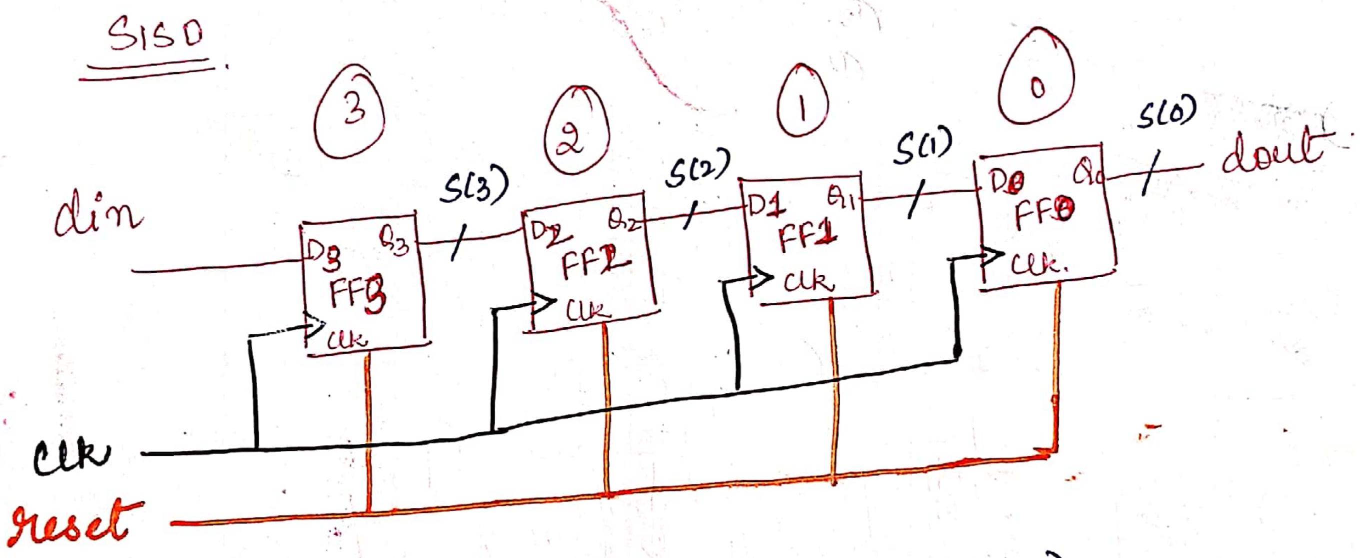
Registers

A Register is a group of flip flops, each of which shares a common clock and is capable of storing one bit of information. An 'n' bit register consists of a group of 'n' flip-flops capable of storing n bits of binary information.

SISO



```
module SISO ( din, clk, reset, dout);
input din, clk, reset;
output dout;
reg dout;
reg [3:0] s;
always @ (posedge clk)
begin
   if (reset)
      s <= 4'b0000;
   else
```

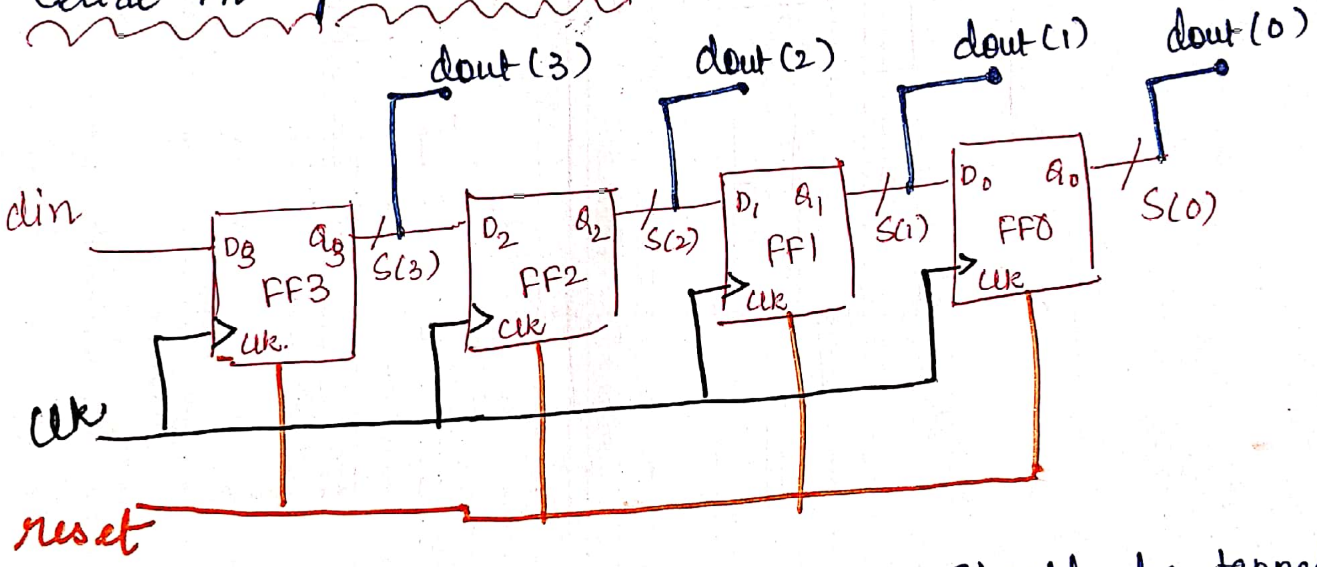
temporary variable, I am using 'reg' instead of wire because I will be using it inside always block.

```
begin

  S[3] <= din ;
  S[2] <= S[3];
  S[1] <= S[2];
  S[0] <= S[1];
end
end

assign dout = S[0];
endmodule
```

Serial in parallel out (SIPO)



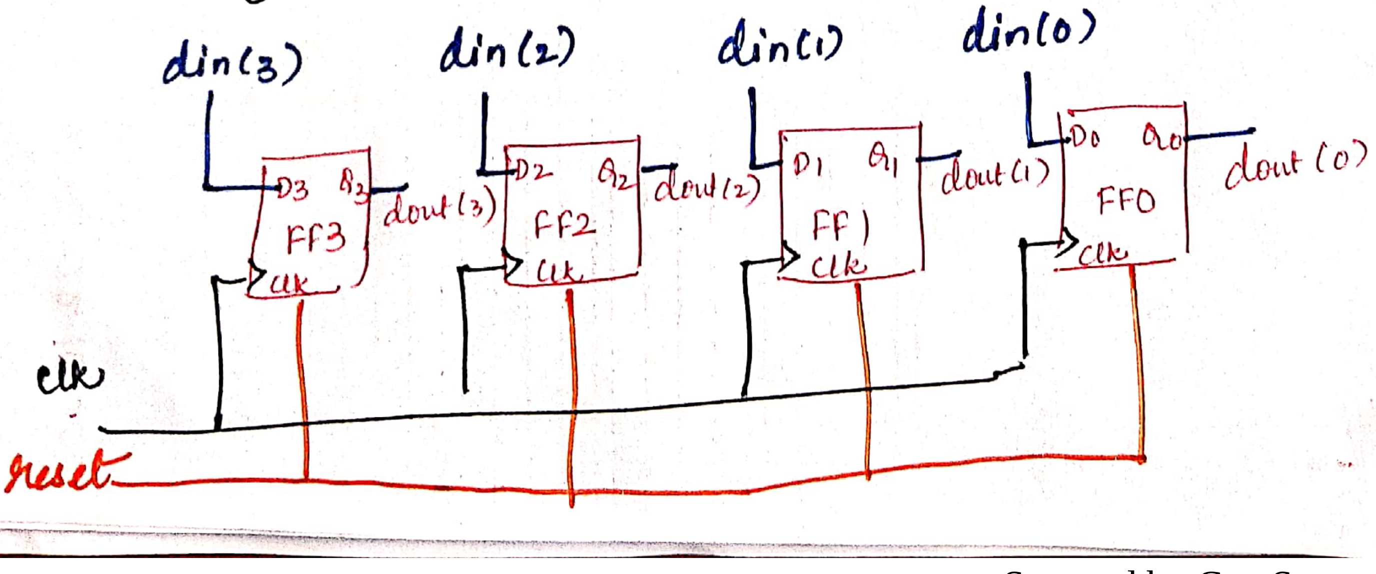
(The output should be tapped parallely too. & at the same time shifted serially)

```
Module SIPO (din, clk, reset, dout);
Input din, clk, reset;
Output [3:0] dout;
 reg [3:0] s;
 always @ (posedge clk)
 begin
   if (reset)
     s <= 4'b0000;
   else
     begin
        s[3] <= din;
        s[2] <= s[3];
        s[1] <= s[2];
        s[0] <= s[1];
     end
   end
   assign dout = s;
   endmodule
```

$\xrightarrow{}$ For parallel output (Output [3:0] dout;)

$\xrightarrow{}$ For parallel output. (assign dout = s;)

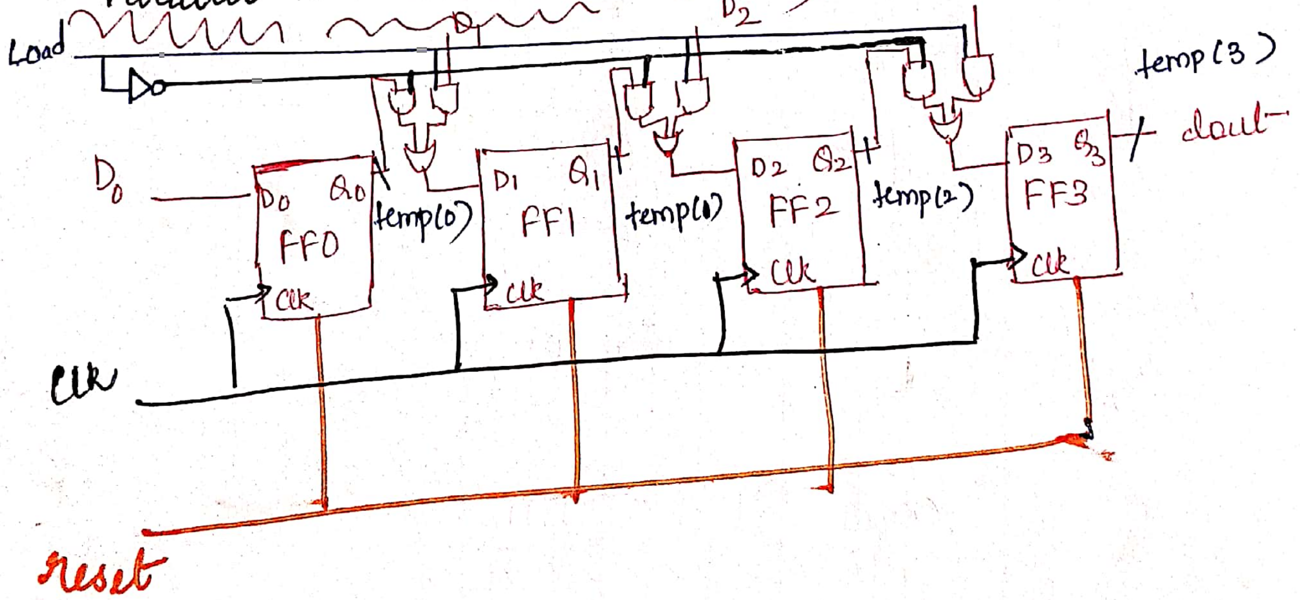
Parallel in Parallel Out (PIPO)



```
module PIPO (din, clk, reset, dout);
input [3:0] din;
input reset, clk;
output [3:0] dout;

always @ (posedge clk)
begin
    if (reset)
        dout <= 4'b0000;
    else
        dout <= din;
    end
endmodule
```

Parallel In Serial Out (PISO)



* When Load = 1, the parallel input D0, D1, D2, D3 get loaded parallelly into the flipflops.

* When Load = 0, the inputs gets shifted serially and you can collect the shifted bit at dout.

```
module PISO ( din, clk, reset, load, dout );
 input [3:0] din ;
  input  clk ; reset, load ;
  output  dout ;
  reg [3:0] temp ;
  always @ (posedge clk )
   begin
     if ( reset )
       temp <= 4'b0000 ;
    else  if  ( load )
       temp  <= din ;

    else
      begin
        dout  <= temp (3) ;
        temp  <= { temp [2:0], 1'b0} ;
       end
     end
   endmodule
```

This can also be written as

temp <= { temp[2], temp [1], temp[0], 1'b0 } ;

temp [3:0]
↳ temp(3), temp(2), temp(1), temp(0)

For this format