



## Universal Shift Register in Digital logic

A Universal shift register is a register which has both the right shift and left shift with parallel load capabilities. Universal shift registers are used as memory elements in computers. A Unidirectional shift register is capable of shifting in only one direction. A bidirectional shift register is capable of shifting in both the directions. The Universal shift register is a combination design of **bidirectional** shift register and a **unidirectional** shift register with parallel load provision.



```
library ieee;  
use ieee. std_logic_1164.all;  
use ieee.  
std_logic_arith.all;  
use ieee.  
std_logic_unsigned.all;
```

```
entity SR_FF is  
PORT( S,R,CLOCK: in  
std_logic;  
Q, QBAR: out std_logic);  
end SR_FF;
```

```
Architecture behavioral of  
SR_FF is
```

```
begin  
PROCESS(CLOCK)  
variable tmp: std_logic;  
begin  
if(CLOCK='1' and CLOCK'EVENT)  
then  
if(S='0' and R='0') then  
tmp:=tmp;  
elsif(S='1' and R='1') then  
tmp:='Z';  
elsif(S='0' and R='1') then  
tmp:='0';  
else  
tmp:='1';  
end if;  
end if;  
Q <= tmp;  
QBAR <= not tmp;  
end PROCESS;  
end behavioral;
```

# VHDL Code for Serial In Parallel Out Shift Register

```
library ieee;  
use ieee.std_logic_1164.all;
```

```
entity sipo is  
  port(  
    clk, clear : in std_logic;  
    Input_Data: in std_logic;  
    Q: out std_logic_vector(3  
downto 0) );  
end sipo;
```

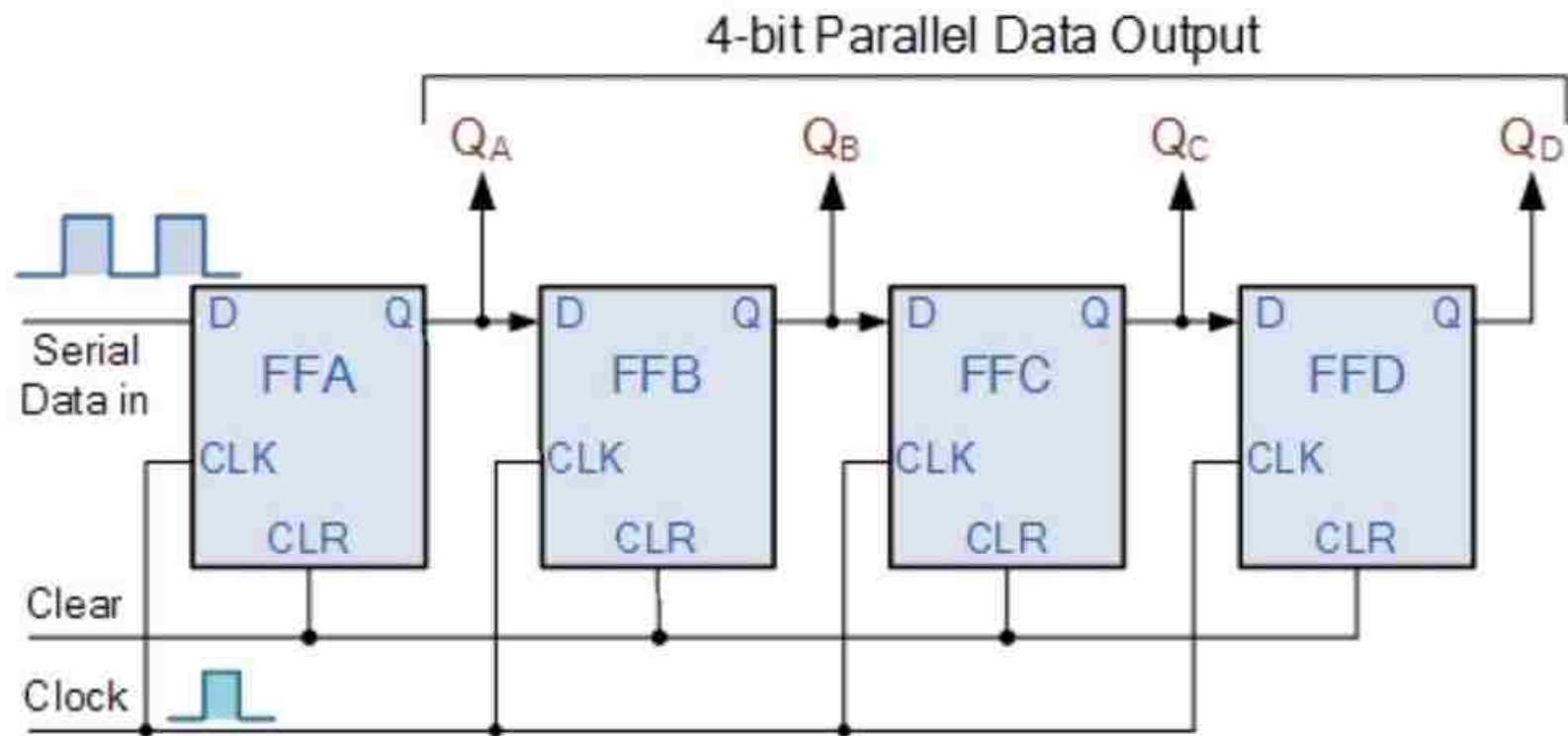
```
architecture arch of sipo is
```

```
begin
```

```
  process (clk)  
  begin  
    if clear = '1' then  
      Q <= "0000";  
    elsif (CLK'event and  
CLK='1') then  
      Q(3 downto 1) <= Q(2 downto  
0);  
      Q(0) <= Input_Data;  
    end if;  
  end process;  
end arch;
```

# Serial-in to Parallel-out (SIPO) Shift Register

## 4-bit Serial-in to Parallel-out Shift Register



The operation is as follows. Lets assume that all the flip-flops ( FFA to FFD ) have just been RESET ( CLEAR input ) and that all the outputs  $Q_A$  to  $Q_D$  are at logic level "0" ie, no parallel data output.

If a logic "1" is connected to the DATA input pin of FFA then on the first clock pulse the output of FFA and therefore the resulting  $Q_A$  will be set HIGH to logic "1" with all the other outputs still remaining LOW at logic "0".

Assume now that the DATA input pin of FFA has returned LOW again to logic "0" giving us one data pulse or 0-1-0.

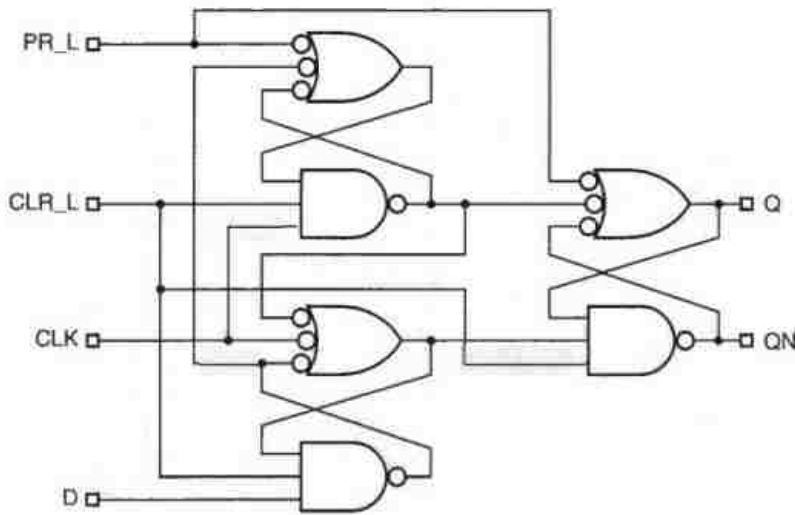
The second clock pulse will change the output of FFA to logic "0" and the output of FFB and  $Q_B$  HIGH to logic "1" as its input D has the logic "1" level on it from  $Q_A$ . The logic "1" has now moved or been "shifted" one place along the register to the right as it is now at  $Q_A$ .

When the third clock pulse arrives this logic “1” value moves to the output of FFC ( $Q_C$ ) and so on until the arrival of the fifth clock pulse which sets all the outputs  $Q_A$  to  $Q_D$  back again to logic level “0” because the input to FFA has remained constant at logic level “0”.

The effect of each clock pulse is to shift the data contents of each stage one place to the right, and this is shown in the following table until the complete data value of 0-0-0-1 is stored in the register. This data value can now be read directly from the outputs of  $Q_A$  to  $Q_D$ .

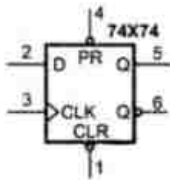
Then the data has been converted from a serial data input signal to a parallel data output. The truth table and following waveforms show the propagation of the logic “1” through the register from left to right as follows.

1. Draw the logic diagram of 74X74 IC and explain the operation.



**Figure 7-20**  
Commercial circuit for  
a positive-edge-  
triggered D flip-flop  
such as 74LS74.

A commonly desired function in D flip-flops is the ability to hold the last value stored, rather than load a new value, at the clock edge. This is accomplished by adding an enable input, called EN or CE (clock enable). While the name "clock enable" is descriptive, the extra input's function is not obtained by controlling the clock in any way whatsoever. If EN is asserted, the external D input is selected; if EN is negated, the flip-flop's current output is used.



D	EN	CLK	Q	QN
0	1		0	1
1	1		1	0
x	0		last Q	last QN
x	x	0	last Q	last QN
x	x	1	last Q	last QN



```
library ieee;
use ieee.std_logic_1164.all;
entity D_flip_flop is
    port (clk,Din : in std_logic;
          Q: out std_logic;
          Qnot : out std_logic);
end D_flip_flop;
architecture DFF_arch of D_flip_flop is
begin
    process (clk,Din)
    begin
        if(clk'event and clk='1') then
            Q <= Din;
            Qnot <= (not Din);
        end if;
    end process;
end DFF_arch;
```

```
library ieee;  
use ieee. std_logic_1164.all;  
use ieee.  
std_logic_arith.all;  
use ieee.  
std_logic_unsigned.all;
```

```
entity JK_FF is  
PORT( J,K,CLOCK: in  
std_logic;  
Q, QB: out std_logic);  
end JK_FF;
```

```
Architecture behavioral of  
JK_FF is  
begin  
PROCESS(CLOCK)  
variable TMP: std_logic;  
begin  
if(CLOCK='1' and CLOCK'EVENT)  
then  
if(J='0' and K='0') then  
TMP:=TMP;  
elsif(J='1' and K='1') then  
TMP:= not TMP;  
elsif(J='0' and K='1') then  
TMP:='0';  
else  
TMP:='1';  
end if;  
end if;  
Q<=TMP;  
Q <=not TMP;  
end PROCESS;  
end behavioral;
```

# FUNCTION TABLE/TRUTH TABLE FOR 74194

---

**TRUTH TABLE**

CLEAR	INPUTS									OUTPUTS			
	MODE		CLOCK	SERIAL		PARALLEL				QA	QB	QC	QD
	S1	S0		LEFT	RIGHT	A	B	C	D				
L	X	X	X	X	X	X	X	X	X	L	L	L	L
H	X	X	$\overline{\text{L}}$	X	X	X	X	X	X	QA0	QB0	QC0	QD0
H	H	H	$\overline{\text{L}}$	X	X	a	b	c	d	a	b	c	d
H	L	H	$\overline{\text{L}}$	X	H	X	X	X	X	H	QA <sub>n</sub>	QB <sub>n</sub>	QC <sub>n</sub>
H	L	H	$\overline{\text{L}}$	X	L	X	X	X	X	L	QA <sub>n</sub>	QB <sub>n</sub>	QC <sub>n</sub>
H	H	L	$\overline{\text{L}}$	H	X	X	X	X	X	QB <sub>n</sub>	QC <sub>n</sub>	QD <sub>n</sub>	H
H	H	L	$\overline{\text{L}}$	L	X	X	X	X	X	QB <sub>n</sub>	QC <sub>n</sub>	QD <sub>n</sub>	L
H	L	L	X	X	X	X	X	X	X	QA0	QB0	QC0	QD0

## VHDL code for a 74LS194 Universal Shift Register

Hi everyone!


I've got an assignment about writing the VHDL code for a 74LS194 shift register, but I'm not allowed to use the functional description. So far I've made this, but I'm not sure if it's ok (I've simulated it, but since I'm new to VHDL I'm not sure if it's working as it should).

Thanks a lot for your help, here's the code:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity IC74LS194 is
Port ( clear, S0, S1, clk, SL, SR, A, B, C, D : in STD_LOGIC;
Qa, Qb, Qc, Qd : out STD_LOGIC);
end IC74LS194;

architecture Schematic of IC74LS194 is
signal Qas, Qbs, Qcs, Qds : STD_LOGIC;
signal LogicA, LogicB, LogicC, LogicD : STD_LOGIC;
begin

LogicA <= (SR and not(S1) and S0) or (S0 and S1 and A) or (not(S0) and S1 and
Qbs) or (S0 and not(S1) and Qas);
LogicB <= (S0 and not(S1) and Qas) or (S0 and S1 and  or (not(S0) and S1 and
Qcs) or (not(S0) and not(S1) and Qbs);
LogicC <= (S0 and not(S1) and Qbs) or (S0 and S1 and C) or (not(S0) and S1 and
Qds) or (not(S0) and not(S1) and Qcs);
LogicD <= (S0 and not(S1) and Qcs) or (S0 and S1 and D) or (not(S0) and S1 and
SL) or (not(S0) and not(S1) and Qds);

FFA: process (clear, clk, Qas)
begin
if clear = '0' then
Qas <= '0';
elsif rising_edge (clk) then
Qas <= LogicA;
else Qas <= Qas;
end if;
end process;

FFB: process (clear, clk, Qbs)
begin
if clear = '0' then
Qbs <= '0';
elsif rising_edge (clk) then
Qbs <= LogicB;
else Qbs <= Qbs;
end if;
end process;

FFC: process (clear, clk, Qcs)
begin

if clear = '0' then
Qcs <= '0';
elsif rising_edge (clk) then
Qcs <= LogicC;
else Qcs <= Qcs;
end if;
end process;

FFD: process (clear, clk, Qds)
begin
if clear = '0' then
Qds <= '0';
elsif rising_edge (clk) then
Qds <= LogicD;
else Qds <= Qds;
end if;
end process;

Qa <= Qas;
Qb <= Qbs;
Qc <= Qcs;
Qd <= Qds;

end Schematic;
```