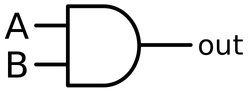
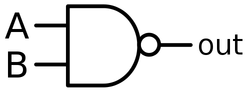
**Logic diagram and logic equation of AND gate**

[](https://i0.wp.com/www.technobyte.org/wp-content/uploads/2018/10/and-logic-gate-symbol.png?ssl=1)

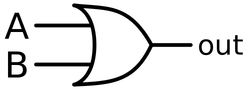
Y(A and B) = A.B

**Logic diagram and logic equation of NAND gate**

[](https://i2.wp.com/www.technobyte.org/wp-content/uploads/2018/10/nand-logic-gate-symbol.png?ssl=1)

Y (A nand B) =  \overline {A.B}  = A|B =  A\uparrow B 

**Logic diagram and logic equation of OR gate**

[](https://i2.wp.com/www.technobyte.org/wp-content/uploads/2018/10/or-logic-gate-symbol.png?ssl=1)

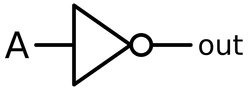
Y (A or B) = A + B

**Logic diagram and logic equation of NOR gate**

[](https://i2.wp.com/www.technobyte.org/wp-content/uploads/2018/10/nor-logic-gate-symbol.png?ssl=1)

Y (A nor B) =  \overline {A+B}  =  A\downarrow B 

**Logic diagram and logic equation of NOT gate**

[](https://i2.wp.com/www.technobyte.org/wp-content/uploads/2018/10/not-logic-gate-symbol.png?ssl=1)

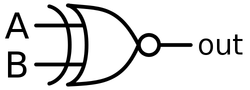
Y (not A) = A’ =  \overline { A }  

**Logic diagram and logic equation of XOR gate**

[](https://i1.wp.com/www.technobyte.org/wp-content/uploads/2018/10/exor-logic-gate-symbol.png?ssl=1)

Y (A exor B) = A  \oplus B = AB’ + A’B

**Logic diagram and logic equation of XNOR gate**

[](https://i2.wp.com/www.technobyte.org/wp-content/uploads/2018/10/exnor-xnor-logic-gate-symbol.png?ssl=1)

Y (A exnor B) =  \overline {A \oplus B}  = A’B’ + AB

**VHDL code for all logic gates using dataflow method**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity ALLGATES\_SOURCE is

Port ( A,B : in  STD\_LOGIC;

P, Q, R, S, T, U, V : out  STD\_LOGIC);

end ALLGATES\_SOURCE;

architecture dataflow of ALLGATES\_SOURCE is

begin

--- you have to remember the commands for boolean logic  in VHDL as shown below

P < = A and B;

Q < = A nand B;

R <= A or B;

S <= A nor B;

T <= not A;

U <= A xor B;

V <= A xnor B;

end dataflow;

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity and\_or\_top is

Port ( INA1 : in STD\_LOGIC; -- AND gate input

INA2 : in STD\_LOGIC; -- AND gate input

OA : out STD\_LOGIC; -- AND gate output

end and\_or\_top;

architecture Behavioral of and\_or\_top is

begin

OA <= INA1 and INA2; -- 2 input AND gate

end Behavioral;

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity and\_or\_top is

Port ( INO1 : in STD\_LOGIC; -- OR gate input

INO2 : in STD\_LOGIC; -- OR gate input

OO : out STD\_LOGIC); -- OR gate output

end and\_or\_top;

architecture Behavioral of and\_or\_top is

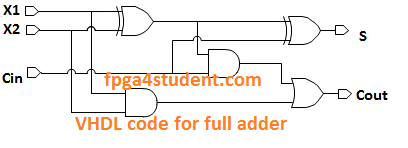
begin

OO <= INO1 or INO2; -- 2 input OR gate

end Behavioral;

# VHDL code for Full Adder

#### The full adder has three inputs X1, X2, Carry-In Cin and two outputs S, Carry-Out Cout as shown in the following figure:



### The VHDL code for the full adder using the structural model:

-- fpga4student.com

-- [FPGA projects](https://www.fpga4student.com/p/fpga-projects.html), [VHDL projects](https://www.fpga4student.com/p/vhdl-project.html), [Verilog projects](https://www.fpga4student.com/p/verilog-project.html)

-- VHDL code for full adder

-- Structural code for full adder

library ieee;

**use** **ieee.std\_logic\_1164.all**;

entity Full\_Adder\_Structural\_VHDL **is**

port(

X1, X2, Cin : **in** std\_logic;

S, Cout : **out** std\_logic

);

**end** **Full\_Adder\_Structural\_VHDL**;

architecture structural **of** Full\_Adder\_Structural\_VHDL **is**

signal a1, a2, a3: std\_logic;

**begin**

a1 <= X1 **xor** X2;

a2 <= X1 **and** X2;

a3 <= a1 **and** Cin;

Cout <= a2 **or** a3;

S <= a1 **xor** Cin;

**end** **structural**;

### The VHDL code for the full adder using the behavioral model:

**library ieee;**

**use ieee.std\_logic\_1164.all;**

**entity fulladder is**

**port(a:in std\_logic\_vector(2 downto 0);**

**s,ca:out std\_logic);**

**end fulladder;**

**architecture fulladder of fulladder is**

**begin**

**process(a)**

**begin**

**if a="000" then s<='0';ca<='0';**

**elsif a="001" then s<='1';ca<='0';**

**elsif a="010" then s<='1';ca<='0';**

**elsif a="011" then s<='0';ca<='1';**

**elsif a="100" then s<='1';ca<='0';**

**elsif a="101" then s<='0';ca<='1';**

**elsif a="110" then s<='0';ca<='1';**

**else s<='1';ca<='1';**

**end if;**

**end process;**

**end fulladder;**

**DECODER**

**Truth table for a 2:4 decoder**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **A** | **B** | **Y3** | **Y2** | **Y1** | **Y0** |
| 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 |

From this truth table, we get the following logic equations for the outputs.

Y0 = A’B’

Y1 = A’B

Y2 = AB’

Y3 = AB

**VHDL code for decoder using dataflow method**

**2:4 DECODER DATAFLOW**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity DECODER\_SOURCE is

   Port ( A,B : in  STD\_LOGIC;

          Y3,Y2,Y1,Y0 : out  STD\_LOGIC);

end DECODER\_SOURCE;

architecture dataflow of DECODER\_SOURCE is

begin

Y0 <= ((not A)and(not B));

Y1 <= ((not A) and B);

Y2 <= (A and (not B));

Y3 <= (A and B);

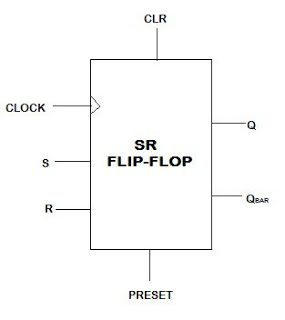
end dataflow;

**flipflops**

A Flip-flop is the basic element which is used to store information of one bit. Flip-flops have their content change either at the rising or falling edge of the enable signal(usually the controlling clock signal).  
  
There are basically four main types of  flip-flops in VHDL:  
 1. SR Flip-flop  
 2. D Flip-flop  
 3. JK Flip-flop  
 4. T Flip-flop.

1**. SR FLIP-FLOP VHDL Code:**

A SR flip flop used in digital electronics will provide the results in a similar manner to the JK flip flop and this is the reason why the vhdl codes for these two flipflops are similar in nature.   
  
Given below is a behavioral approach of writing the VHDL code for a SR Flip-flop**.**

[](http://4.bp.blogspot.com/-EBZx_rVty8g/UZzlO0M2sPI/AAAAAAAAAD8/luzfU9abCvI/s1600/Capture.JPG)

*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,CLR,PRESET: in std\_logic;*

*Q, QBAR: out std\_logic);*

*end**SR-FF;*

*Architecture behavioral of SR-FF is*

*begin*

*P1: PROCESS(CLOCK,CLR,PRESET)*

*variable x: std\_logic;*

*begin*

*if(CLR='0') then*

*x:='0';*

*elsif(PRESET='0')then*

*x:='1';*

*elsif(CLOCK='1' and CLOCK'EVENT) then*

*if(S='0' and R='0')then*

*x:=x;*

*elsif(S='1' and R='1')then*

*x:='Z';*

*elsif(S='0' and R='1')then*

*x:='0';*

*else*

*x:='1';*

*end if;*

*end if;*

*Q<=x;*

*QBAR<=not x;*

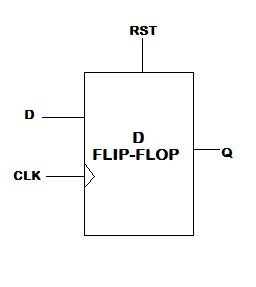
*end PROCESS;*

*end behavioral;*

**2. D FLIP-FLOP VHDL Code:**

A D flip flop or Delay flip flop gives the same output as the input provided and thus the vhdl code is much simpler.

Given below is a behavioral approach of writing the vhdl code for a D Flip-flop.

[](http://2.bp.blogspot.com/-AoatpdknKns/UZztN98ANjI/AAAAAAAAAEM/VWghgFNlIvU/s1600/Capture.JPG)

*library ieee;*

*use ieee. std\_logic\_1164.all;*

*use ieee. std\_logic\_arith.all;*

*use ieee. std\_logic\_unsigned.all;*

*entity D-FF is*

*PORT( D,CLK,RST: in std\_logic;*

*Q: out std\_logic);*

*end D-FF;*

*architecture behavioral of D-FF is*

*begin*

*P1: process(RST,CLK)*

*begin*

*if(RST='1')then*

*Q<='0';*

*elsif(CLK='1' and CLK'EVENT) then*

*Q<=D;*

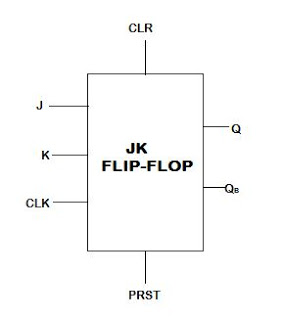
*end if;* *end process;*

*end behavioral;*

[Best VHDL books online](https://www.amazon.in/gp/search?ie=UTF8&tag=akki93-21&linkCode=ur2&linkId=f52ae4566eabcf2fc4af699090ca3828&camp=3638&creative=24630&index=books&keywords=vhdl)http://ir-in.amazon-adsystem.com/e/ir?t=akki93-21&l=ur2&o=31

**3. JK FLIP-FLOP VHDL Code:**

Given below is a behavioral approach of writing the VHDL code for a JK Flip-flop.

**[](http://3.bp.blogspot.com/-lNx-OS84Rvc/UZzyTXVkyBI/AAAAAAAAAEc/tXlmyWSO3S4/s1600/Capture.JPG)**

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,CLK,PRST,CLR: in std\_logic;

                              Q, QB: out std\_logic);

end JK-FF;

Architecture behavioral of JK-FF is

begin

P1: PROCESS(CLK,CLR,PRST)

variable x: std\_logic;

begin

if(CLR='0') then

 x:='0';

elsif(PRST='0')then

 x:='1';

elsif(CLK='1' and CLK'EVENT) then

    if(J='0' and K='0')then

       x:=x;

   elsif(J='1' and K='1')then

       x:= not x;

   elsif(J='0' and K='1')then

       x:='0';

   else

       x:='1';

   end if;

end if;

   Q<=x;

   QB<=not x;

end PROCESS;

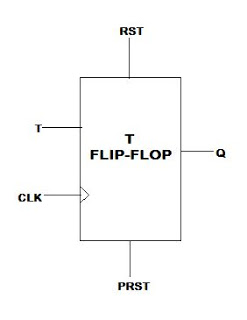
end behavioral;

http://ir-in.amazon-adsystem.com/e/ir?t=akki93-21&l=am2&o=31&a=0070499446

**4. T FLIP-FLOP VHDL Code:**

The T in a t flip flop stands for toggle and this is exactly what this digital component does. It simply toggles the value of a particular input. A basic not gate will solve the problem in the vhdl code for this element.

Given below is a behavioral approach of writing the VHDL code for a T Flip-flop.

**[](http://4.bp.blogspot.com/-r5TZPHLveLE/UZz3TlNl2pI/AAAAAAAAAE0/S1-dMs9yWCs/s1600/Capture.JPG)**

library ieee;

use ieee. std\_logic\_1164.all;

use ieee. std\_logic\_arith.all;

use ieee. std\_logic\_unsigned.all;

entity T-FF is

PORT( T,CLK,PRST,RST: in std\_logic;

                                    Q: out std\_logic);

end T-FF;

architecture behavioral of T-FF is

begin

P1: process(CLK,PRST,RST)

variable x: std\_logic;

begin

if(RST='0') then

x:='0';

elsif(RST='1' and PRST='0') then

x:='1';

elsif(CLK='1' and CLK'EVENT) then

if(T='1')then

x:= not x;

end if;

end if;

   Q<=x;

end process;

end behavioral;