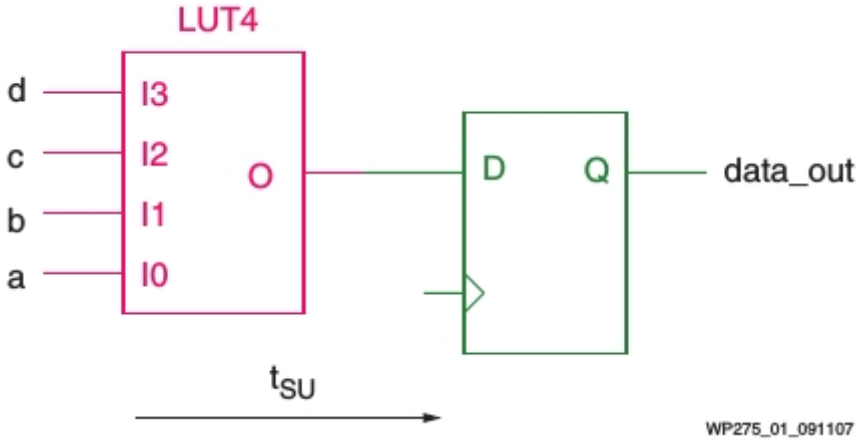
VHDL “coding” implications

Flip/Flops

Four signals ANDed and then captured with a flip/flop



VHDL Example

```
process (clk)
begin
  if clk'event and clk='1' then
    data_out <= a and b and c and d;
  end if;
end process;
```

Now add an asynchronous reset

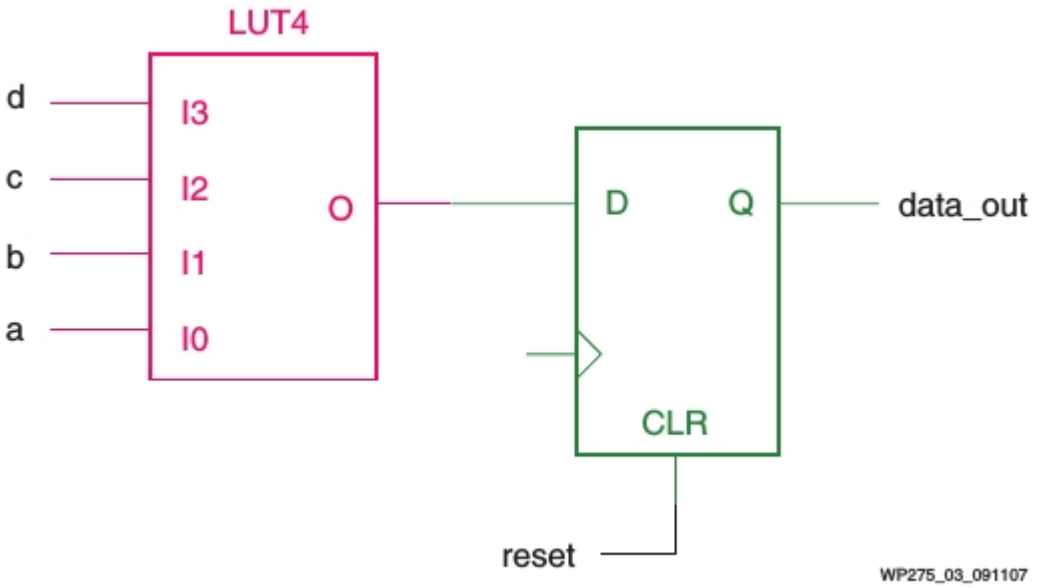


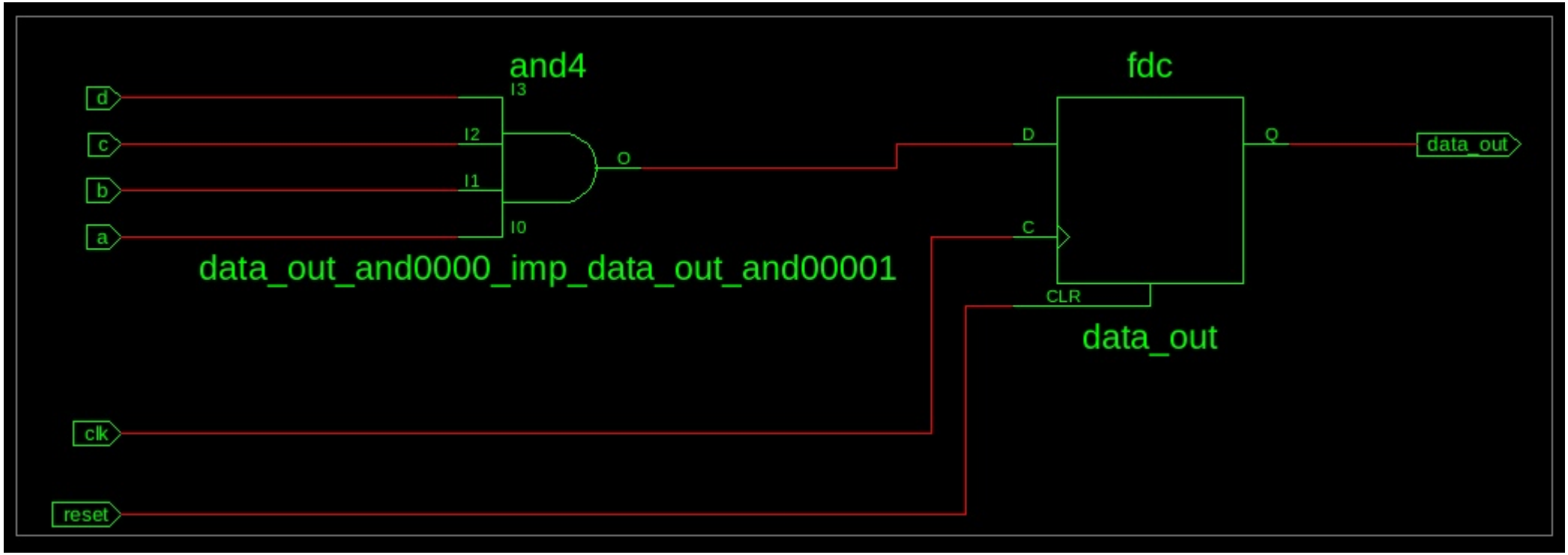
Figure 3: **Addition of Reset**

VHDL Example

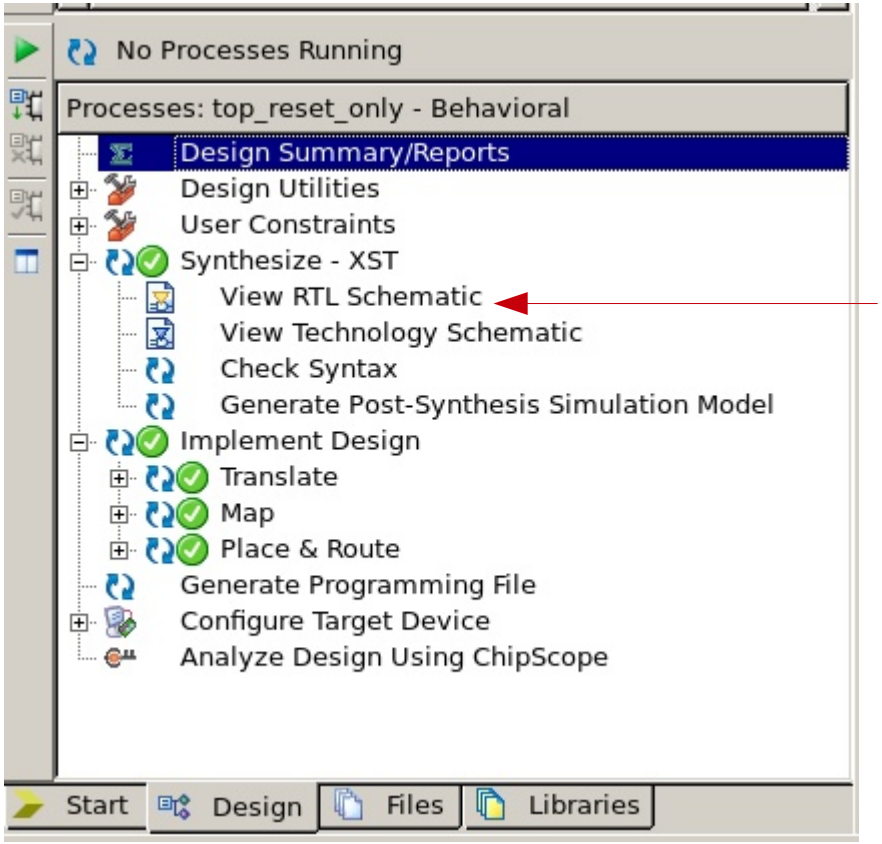
```
process (clk, reset)
begin
  if reset='1' then
    data_out <= '0';
  elsif clk'event and clk='1' then
    data_out <= a and b and c and d;
  end if;
end process;
```

```
entity top_reset_only is
    Port ( clk : in  STD_LOGIC;
           reset : in  STD_LOGIC;
           a,b,c,d : in  STD_LOGIC;
           data_out : out  STD_LOGIC);
end top_reset_only;

architecture Behavioral of top_reset_only is

begin
   process (clk, reset)
   begin
      if reset='1' then
         data_out <= '0';
      elsif clk'event and clk='1' then
         data_out <= a and b and c and d;
   end if;
end process;
```

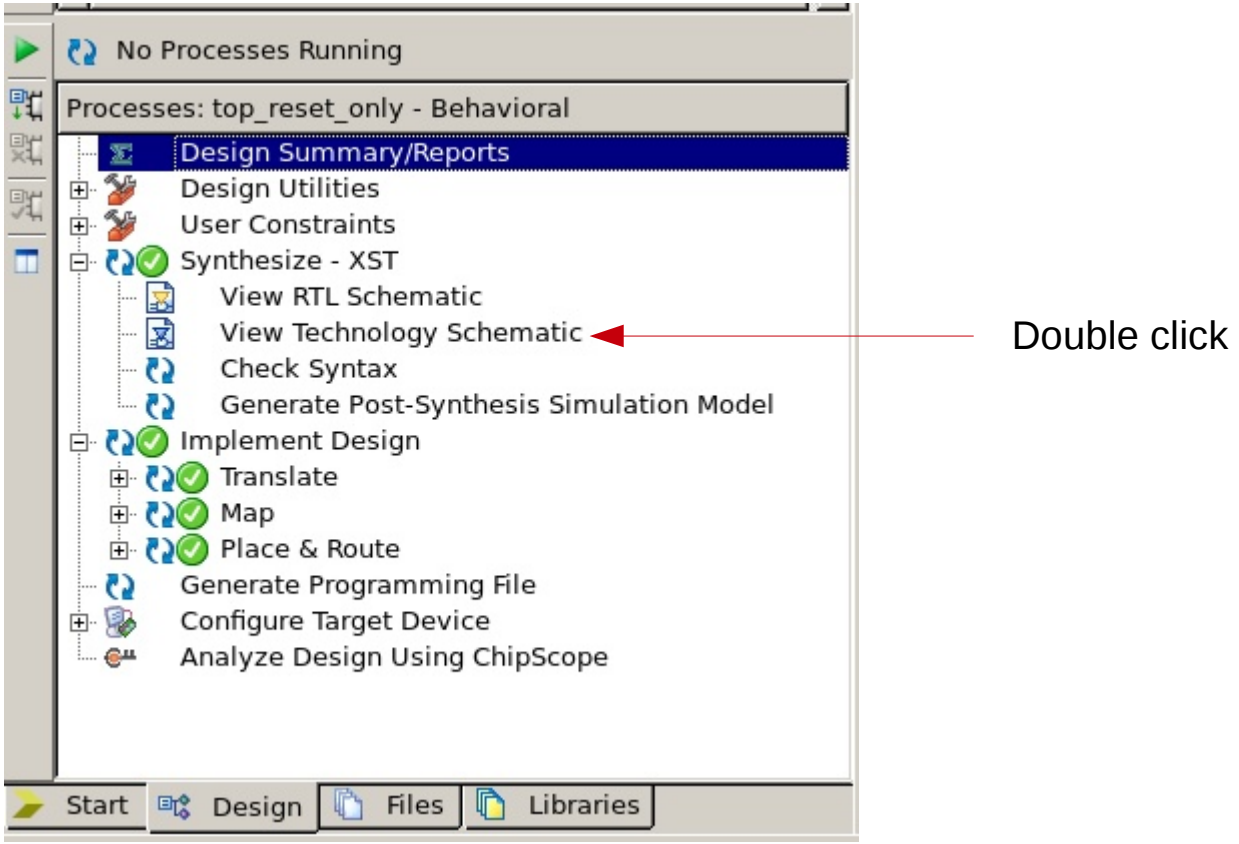
RTL schematic generated by Xilinx ISE tools

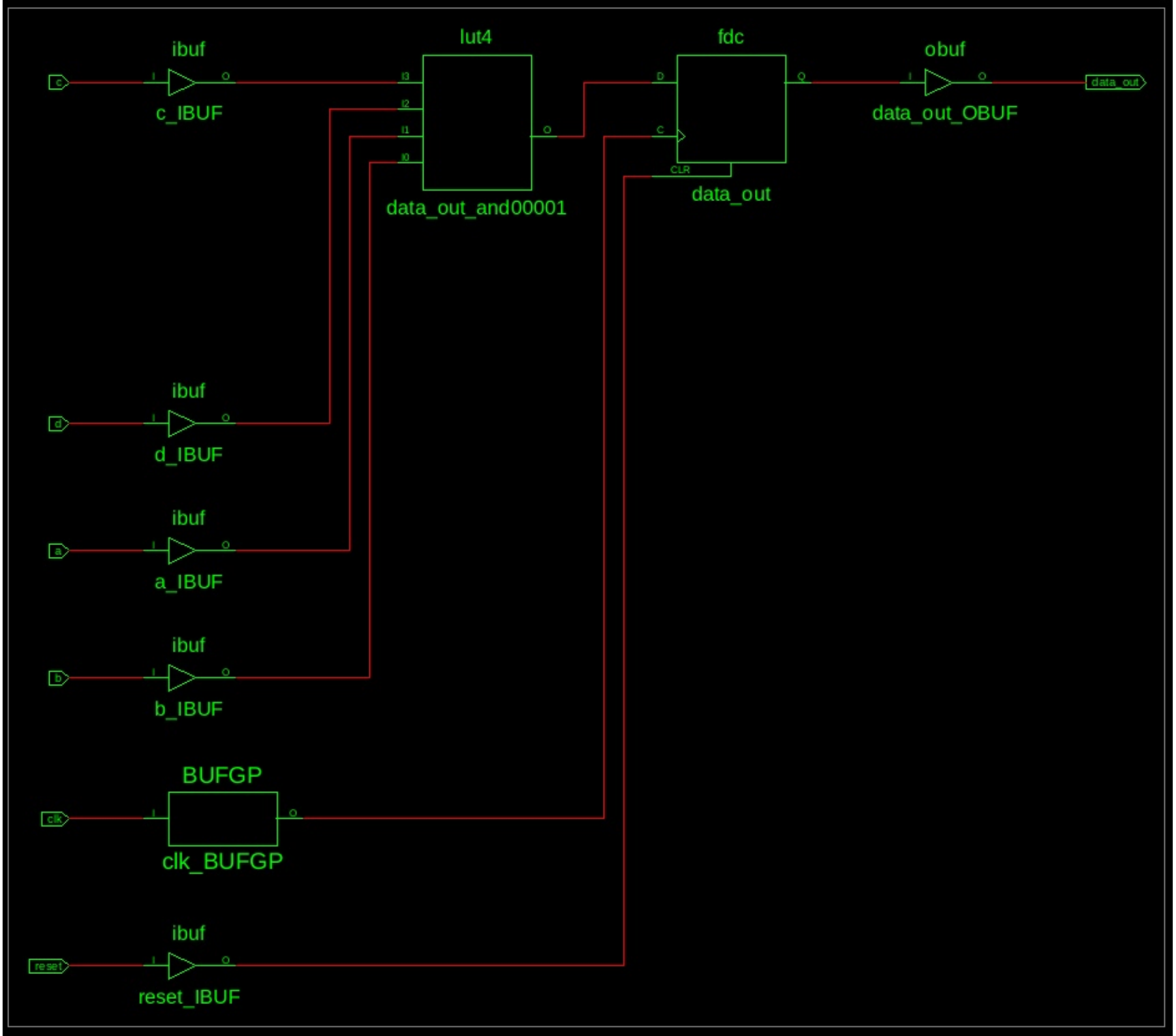


Generate an RTL schematic of a design



Generate and view a Technology schematic of a design




ibuf
c
I
O
c_IBUF
lut4
I3
I2
I1
I0
O
data_out_and00001
fdc
D
C
CLR
Q
data_out
obuf
I
O
data_out
data_out_OBUF
ibuf
d
I
O
d_IBUF
ibuf
a
I
O
a_IBUF
ibuf
b
I
O
b_IBUF
BUFGP
clk
I
O
clk_BUFGP
ibuf
reset
I
O
reset_IBUF

Now lets add Enable and Preset (i.e. Set) inputs

```
process (clk,reset)
begin
  if reset='1' then                                  } Reset
     data_out <= '0';
   elsif clk'event and clk='1' then
     if enable='1' then                              } Enable
       if force_high='1' then                        } Set
          data_out <= '1';
        else
          data_out <= a and b and c and d;           } Logic
       end if;
     end if;
  end if;
end process;
```

```
process (clk,reset)
begin
  if reset='1' then                                  } Reset
     data_out <= '0';
   elsif clk'event and clk='1' then
     if enable='1' then                              } Enable
      if force_high='1' then                         } Set
         data_out <= '1';
       else
         data_out <= a and b and c and d;            } Logic
      end if;
     end if;
  end if;
end process;
```

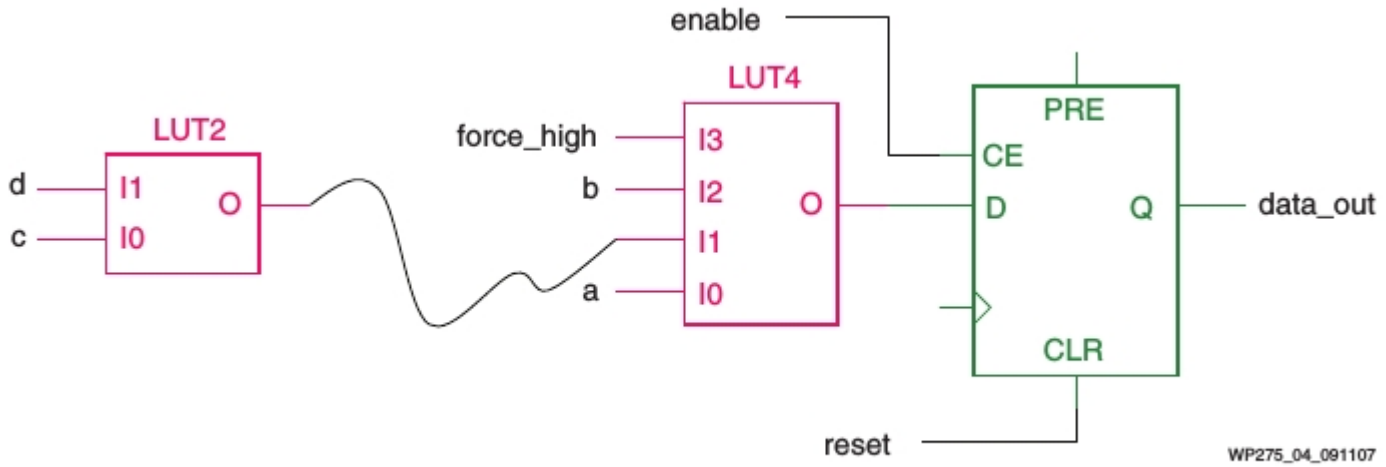
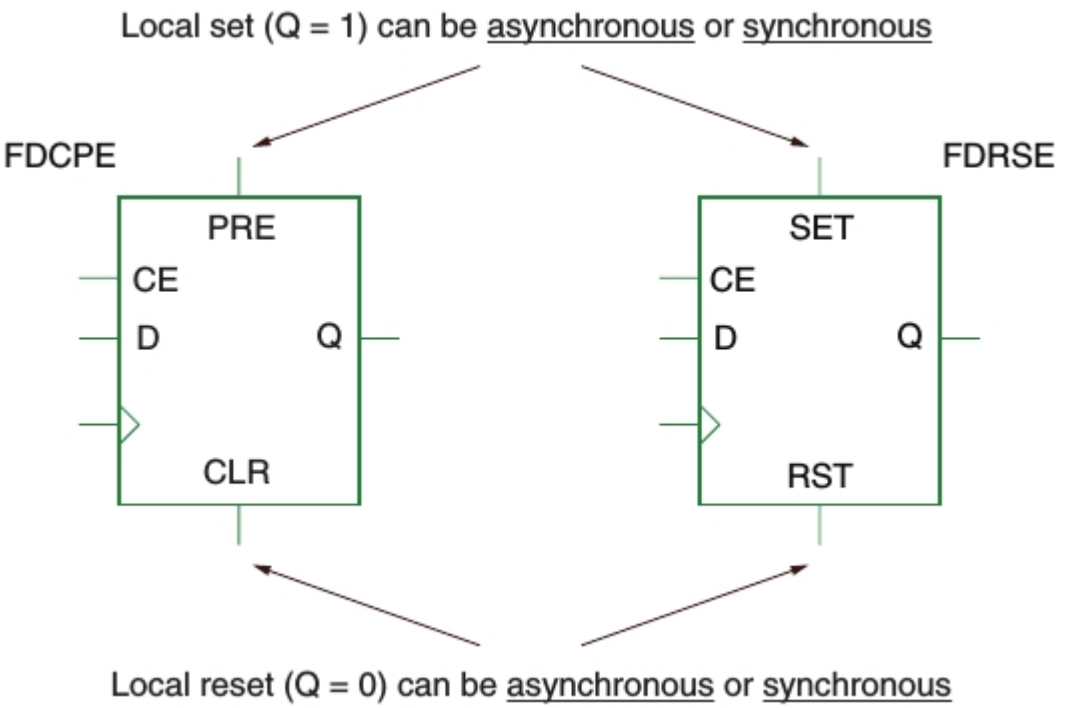


Figure 4: **Additional Controls**



Set & Reset can be Asynchronous or Synchronous with clock

BUT

<u>When both set & reset are used :</u>

both have to be asynchronous

OR

both have to be synchronous

So, lets make both set and reset synchronous

```
process (clk)
begin
  if clk'event and clk='1' then
    if reset='1' then                                    } Synchronous Reset
       data_out <= '0';
     else
       if enable='1' then                                } Enable
         if force_high='1' then                          } Synchronous Set
            data_out <= '1';
          else
            data_out <= a and b and c and d;             } Logic
         end if;
       end if;
    end if;
  end if;
end process;
```

So, lets make both set and reset synchronous

```
process (clk)
begin
  if clk'event and clk='1' then
    if reset='1' then                                  } Synchronous Reset
       data_out <= '0';
     else
       if enable='1' then                              } Enable
         if force_high='1' then                        } Synchronous Set
            data_out <= '1';
          else
            data_out <= a and b and c and d;           } Logic
         end if;
       end if;
    end if;
  end if;
end process;
```

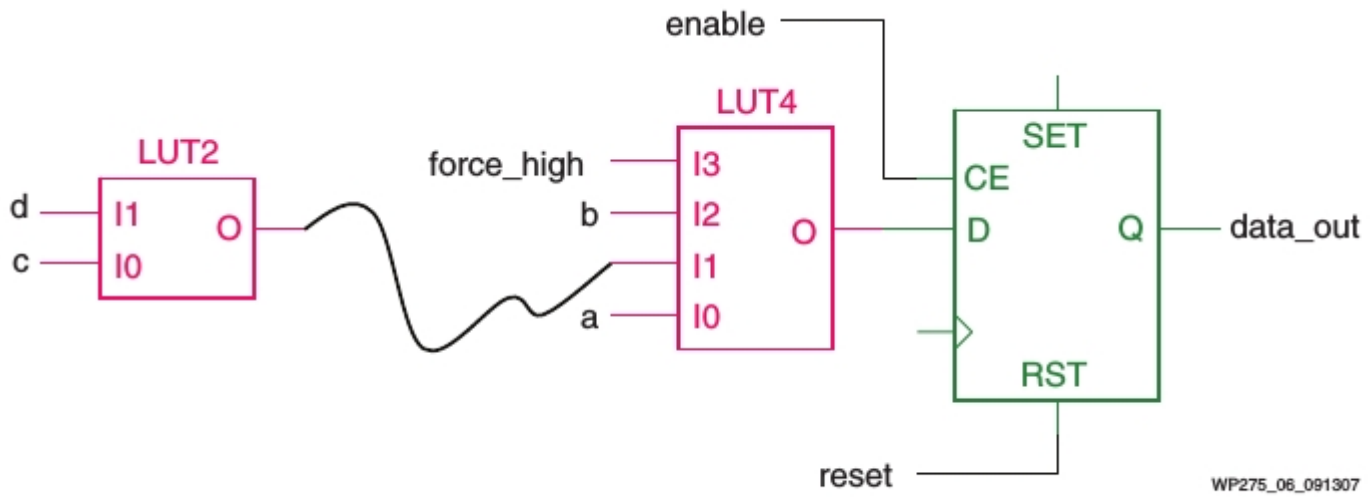


Figure 6: **XST Synchronous Implementation**

So, lets make both set and reset synchronous

```
process (clk)
begin
  if clk'event and clk='1' then
    if reset='1' then                                  } Synchronous Reset
       data_out <= '0';
     else
       if enable='1' then                              } Enable
         if force_high='1' then                        } Synchronous Set
            data_out <= '1';
          else
            data_out <= a and b and c and d;           } Logic
         end if;
       end if;
    end if;
  end if;
end process;
```

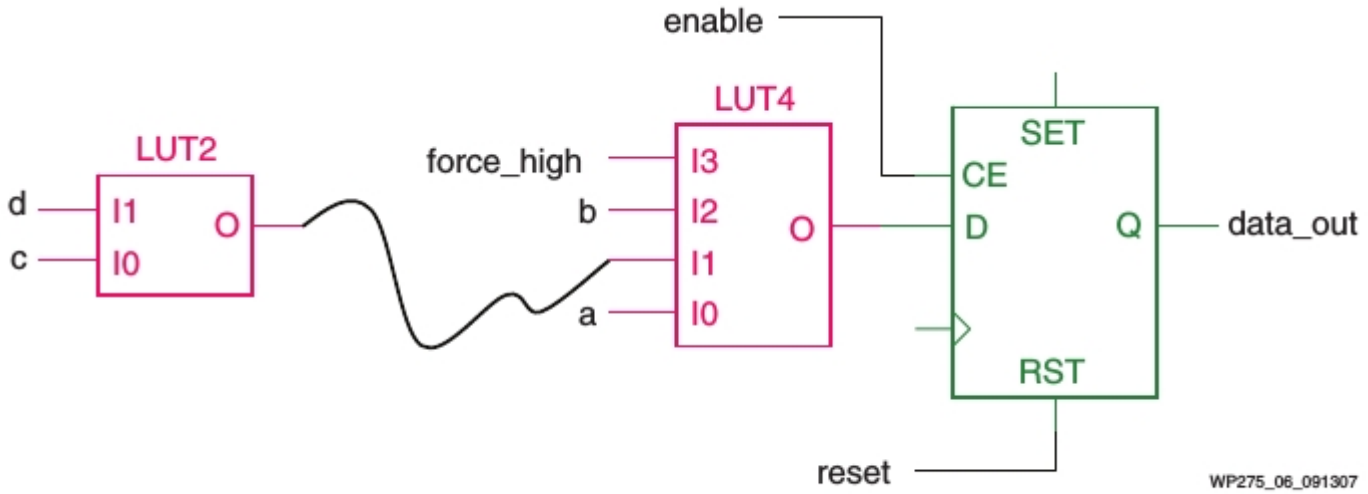
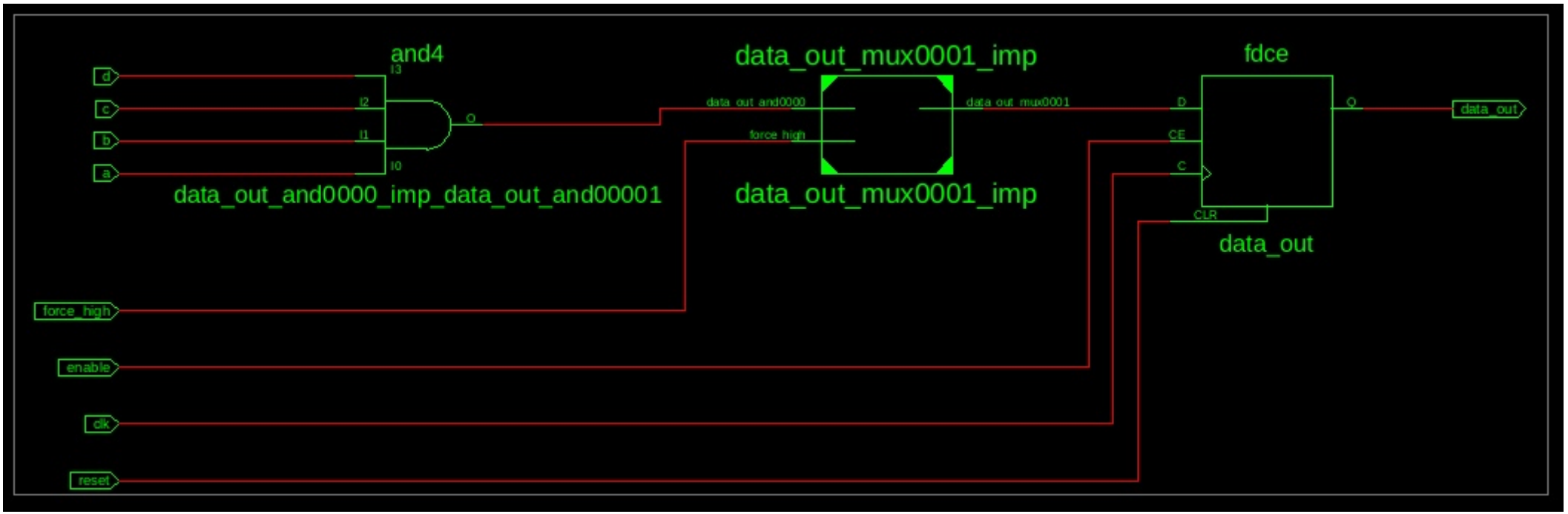


Figure 6: **XST Synchronous Implementation**

Why two LUTs now? What is going on? Actual circuit?

```
entity reset_en_set is
    Port ( clk : in  STD_LOGIC;
           reset : in  STD_LOGIC;
           enable : in  STD_LOGIC;
           force_high : in  STD_LOGIC;
           a,b,c,d : in  STD_LOGIC;
           data_out : out  STD_LOGIC);
end reset_en_set;

architecture Behavioral of reset_en_set is

begin
   process (clk,reset,enable,force_high)
   begin
      if reset='1' then
         data_out <= '0';
      elsif clk'event and clk='1' then
         if enable='1' then
            if force_high='1' then
               data_out <= '1';
            else
               data_out <= a and b and c and d;
            end if;
         end if;
      end if;
   end process;
end Behavioral;
```


and4
I3
d
I2
c
I1
b
I0
a
O
data_out_and0000_imp_data_out_and00001
data_out_mux0001_imp
data_out_and0000
force_high
data_out_mux0001
data_out_mux0001_imp
fdce
D
CE
C
CLR
Q
data_out
force_high
enable
clk
reset

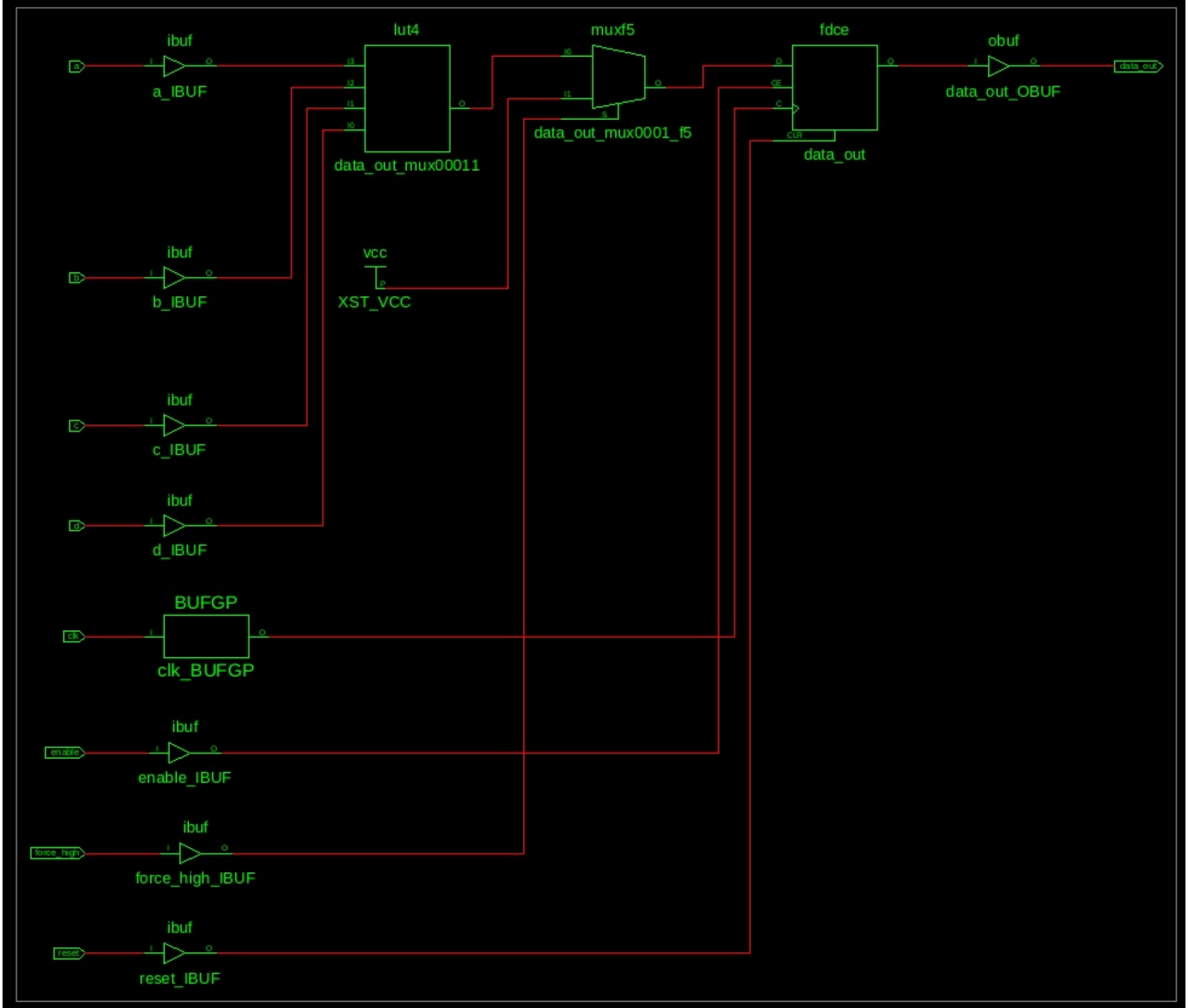

ibuf
a_IBUF
lut4
data_out_mux00011
muxf5
data_out_mux0001_f5
fdce
data_out
obuf
data_out_OBUF
data_out
ibuf
b_IBUF
vcc
XST_VCC
ibuf
c_IBUF
ibuf
d_IBUF
BUFGP
clk_BUFGP
ibuf
enable_IBUF
ibuf
force_high_IBUF
ibuf
reset_IBUF

Table 1: **Example Discrete Device Characteristics**

Inputs					Outputs
R	**S**	**CE**	**D**	**C**	**Q**
1	X	X	X	↑	0
0	1	X	X	↑	1
0	0	0	X	X	No Change
0	0	1	1	↑	1
0	0	1	0	↑	0

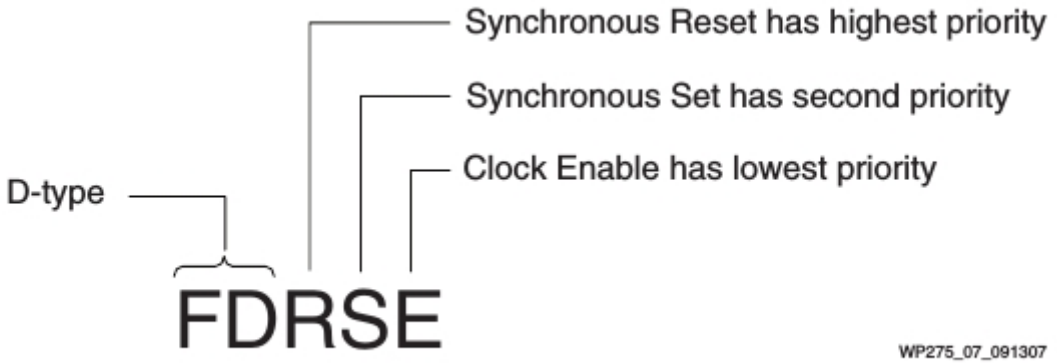


Figure 7: **FDRSE Definition**

Check out the priority implied by the ordering of the if/else statements:

```
process (clk,reset)
begin
  if reset='1' then                                    } Reset
     data_out <= '0';
   elsif clk'event and clk='1' then
     if enable='1' then                                } Enable
       if force_high='1' then                          } Set
          data_out <= '1';
        else
          data_out <= a and b and c and d;             } Logic
       end if;
     end if;
  end if;
end process;
```

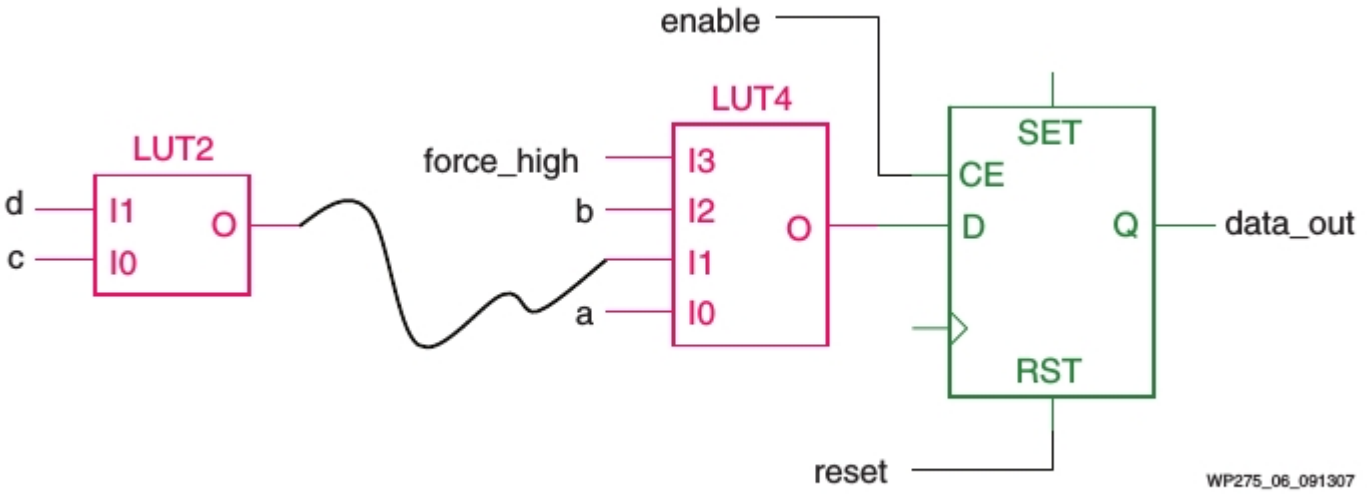


Figure 6: **XST Synchronous Implementation**

Reorder set and enable within the if/else statements

```
process (clk)
begin
  if clk'event and clk='1' then
    if reset='1' then                                  } Synchronous Reset
       data_out <= '0';
     else
       if force_high='1' then                          } Synchronous Set
          data_out <= '1';
        else
        if  enable='1' then                            } Enable
           data_out <= a and b and c and d;            } Logic
        end if;
       end if;
    end if;
  end if;
end process;
```

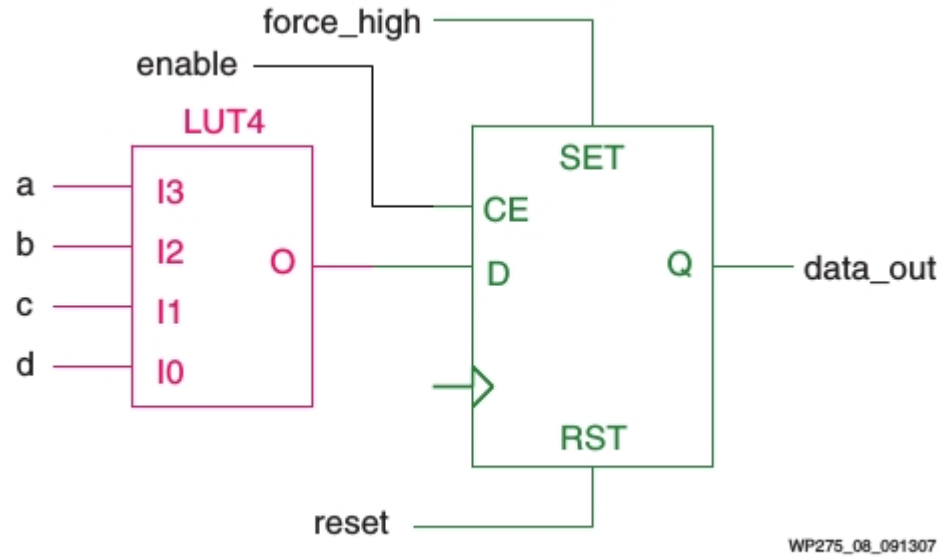
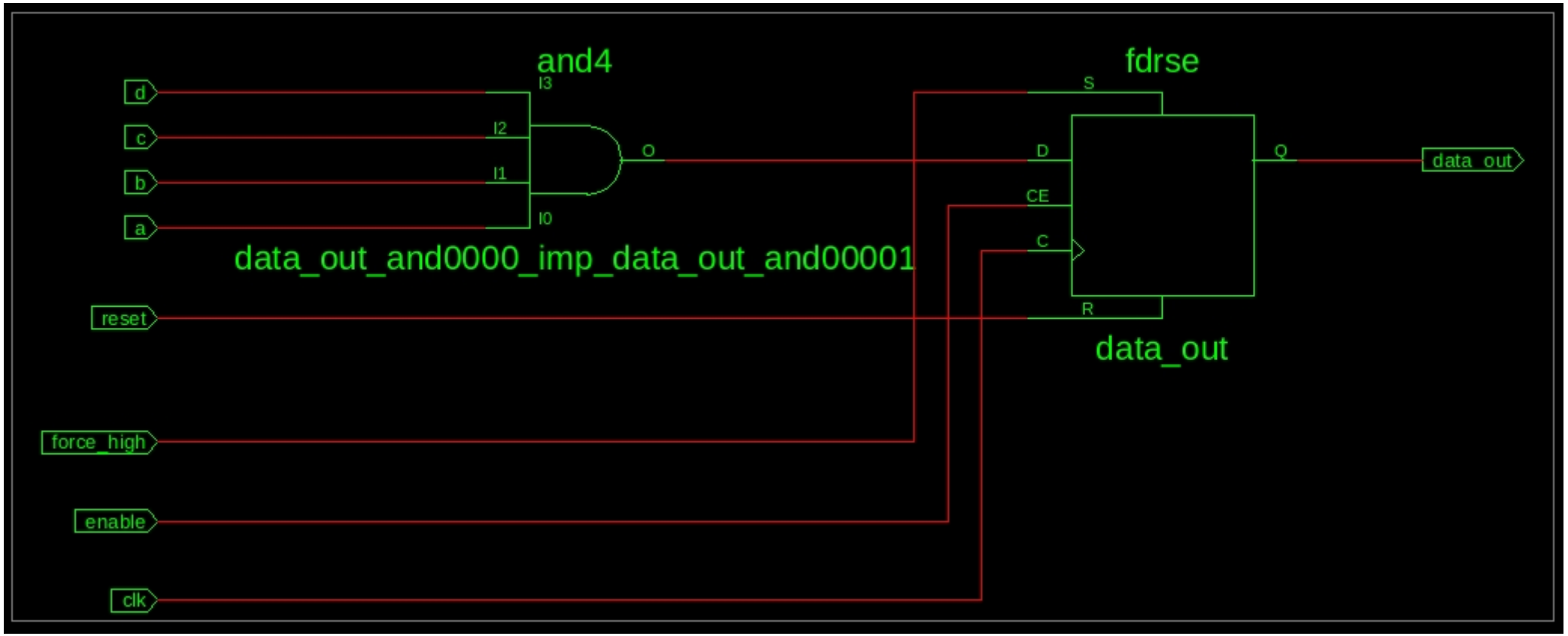


Figure 8: **Smaller, Higher Performance Implementation**

```
entity sync_reset_en_set is
    Port ( clk : in  STD_LOGIC;
           reset : in  STD_LOGIC;
           enable : in  STD_LOGIC;
           force_high : in  STD_LOGIC;
           a,b,c,d : in  STD_LOGIC;
           data_out : out  STD_LOGIC);
end sync_reset_en_set;

architecture Behavioral of sync_reset_en_set is
begin
   process (clk,reset,enable,force_high)
   begin
      if clk'event and clk='1' then
         if reset='1' then
            data_out <= '0';
         else
            if force_high='1' then
               data_out <= '1';
            else
               if enable='1' then
                  data_out <= a and b and c and d;
               end if;
            end if;
         end if;
      end if;
end process;
end Behavioral;
```

Voila - - - it works!!



Initial

Device Utilization Summary				[-]
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of 4 input LUTs	2	4,896	1%	
Number of occupied Slices	1	2,448	1%	
Number of Slices containing only related logic	1	1	100%	
Number of Slices containing unrelated logic	0	1	0%	
Total Number of 4 input LUTs	2	4,896	1%	
Number of bonded IOBs	9	66	13%	
IOB Flip Flops	1			
Number of BUFGMUXs	1	24	4%	
Average Fanout of Non-Clock Nets	0.88			

Revised

Device Utilization Summary				[-]
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of 4 input LUTs	1	4,896	1%	
Number of occupied Slices	1	2,448	1%	
Number of Slices containing only related logic	1	1	100%	
Number of Slices containing unrelated logic	0	1	0%	
Total Number of 4 input LUTs	1	4,896	1%	
Number of bonded IOBs	9	66	13%	
IOB Flip Flops	1			
Number of BUFGMUXs	1	24	4%	
Average Fanout of Non-Clock Nets	0.75			