Flip-Flop HDL

Last updated 1/18/21

Flip-Flop HDL

- Latches are recognized by a pre-defined VHDL template

```
process (clock signal)
  begin
    if(clock signal) then
      actions
    end if;
  end process;
```

note:
- If statement without an else

```
process (i_clk, i_d)
  begin
    if(i_clk = '1') then
      o_q <= i_d;
    end if;
  end process;
```

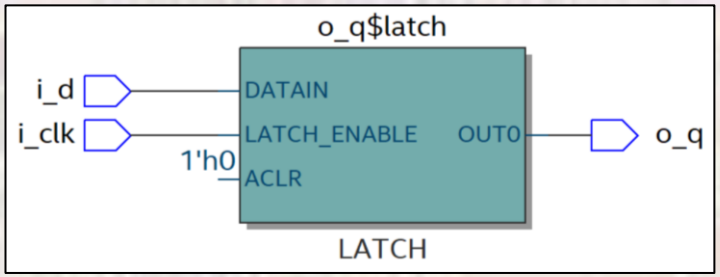
– what happens when the clock signal is false case is not defined
- the synthesizer must assume you meant to create a latch

Flip-Flop HDL

- Latches are recognized by a pre-defined VHDL template

```
--------------------------------
--
-- latches.vhdl
--
-- created: 1/26/18
-- by: johnsontimoj
-- rev: 0
--
-- file to synthesize latches for lecture slides
--
--------------------------------
library ieee;
use ieee.std_logic_1164.all;

entity latches is
   port(
      i_clk :  in std_logic;
      i_d   :  in std_logic;

      o_q   :  out std_logic
   );
end entity latches;
```

```
-- latch generated---------------
architecture behavioral of latches is
begin
   process(i_clk, i_d)
   begin
      if(i_clk = '1') then
         o_q <= i_d;
      end if;
   end process;

end architecture;
--------------------------------
```



Type	ID	Message
⚠	10631	VHDL Process Statement warning at latches.vhdl(27): inferring latch(es) for signal or variable "o_q", which holds its previous value in one or more paths through the process
ℹ	10041	Inferred latch for "o_q" at latches.vhdl(27)
ℹ		Quartus Prime Analysis & Elaboration was successful. 0 errors, 2 warnings

Flip-Flop HDL

- ~~Latches~~ are recognized by a pre-defined VHDL template

```
process (clock signal)
  begin
    if(clock signal) then
      actions
    else
      actions
    end if;
  end process;
```

```
process (i_clk)
  begin
    if(i_clk = 1) then
      o_q <= i_d;
    else
      o_q <= 1;
    end if;
  end process;
```

if a valid else statement is included then something other than a latch will be created

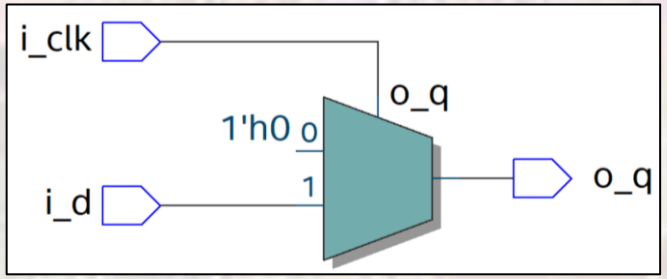
Flip-Flop HDL

- ~~Latches~~ are recognized by a pre-defined VHDL template

```
--------------------------------
--
-- latches.vhdl
--
-- created: 1/26/18
-- by: johnsontimoj
-- rev: 0
--
-- file to synthesize latches for lecture slides
--
--------------------------------
library ieee;
use ieee.std_logic_1164.all;

entity latches is
   port(
      i_clk :  in std_logic;
      i_d   :  in std_logic;

      o_q   :  out std_logic
   );
end entity latches;
```

```
-- no latch generated------------
architecture behavioral of latches is
begin
   process(i_clk, i_d)
   begin
      if(i_clk = '1') then
         o_q <= i_d;
      else
         o_q <= '0';
      end if;
   end process;
end architecture;
-----------------------------------
```

note:
- if a valid else statement is included then something other than a latch will be created




```
18236 Number of processors has not been specified which may cause overloading on shared machines.  Set the global assignment NUM_P
      Quartus Prime Netlist Viewers Preprocess was successful. 0 errors, 1 warning
```

Flip-Flop HDL

- Warning – warning – warning
  - Most (maybe all) of the times you do not complete an if-else a latch will be created
  - **We do not want latches - EVER**

Flip-Flop HDL

- Flip-Flops are recognized by a pre-defined VHDL template

```
process (clock signal)
  begin
    if(clock edge detection) then
      actions
    end if;
  end process;
```

note:
- here the else is not required because the synthesizer recognizes the edge detection

- you can include an else for clarity

```
process (i_clk)
  begin
    if(i_clk'event and i_clk = 1) then
      o_q <= i_d;
    end if;
  end process;
```

```
process (i_clk)
  begin
    if(rising_edge(i_clk)) then
      o_q <= i_d;
    end if;
  end process;
```

2008 release

Flip-Flop HDL

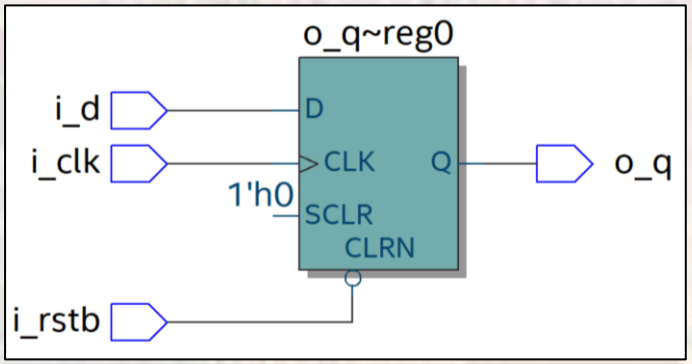
- D-FF w/ asynchronous rstb

```
--------------------------------
--
-- flops.vhdl
--
-- created: 1/26/18
-- by: johnsontimoj
-- rev: 0
--
-- file to synthesize flip-flops for lecture slides
--
--------------------------------
library ieee;
use ieee.std_logic_1164.all;

entity flops is
   port(
      i_clk:   in std_logic;
      i_d :    in std_logic;
      i_rstb:  in std_logic;

      o_q :    out std_logic
   );
end entity flops;
```

Note rstb IS in the sensitivity list

```
-- flop generated---------------
architecture behavioral of flops is
begin
   process (i_clk, i_rstb)
   begin
      if (i_rstb = '0') then
         o_q <= '0';
      elsif (rising_edge(i_clk)) then
         o_q <= i_d;
      end if;
   end process;

end architecture;
---------------------------------
```

Asynchronous inputs are outside the rising_edge



Most of our designs will use DFFs with an asynchronous reset

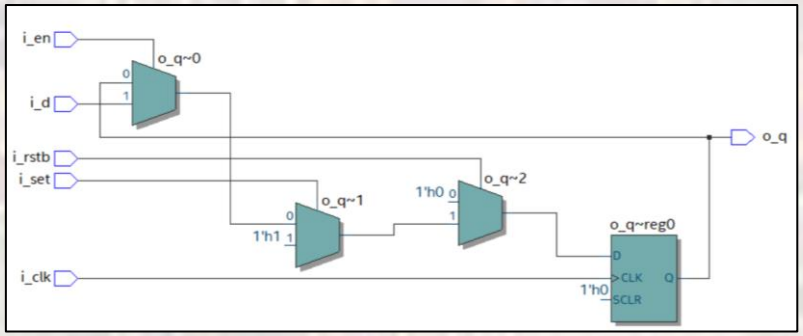
Flip-Flop HDL

- D-FF w/ synchronous set, rstb, en

```
--------------------------------
--
-- flops.vhdl
--
-- created: 1/26/18
-- by: johnsontimoj
-- rev: 0
--
-- file to synthesize flip-flops for lecture slides
--
--------------------------------
library ieee;
use ieee.std_logic_1164.all;

entity flops is
   port(
      i_clk:   in std_logic;
      i_d :    in std_logic;
      i_set:   in std_logic;
      i_en: in std_logic;
      i_rstb:  in std_logic;

      o_q :    out std_logic
   );
end entity flops;
```

```
-- fancy flop generated--------------
architecture behavioral of flops is
begin
   process (i_clk)
   begin
      if (rising_edge(i_clk)) then
         if (i_rstb = '0') then
            o_q <= '0';
         elsif (i_set = '1') then
            o_q <= '1';
         elsif (i_en = '1') then
            o_q <= i_d;
         end if;
      end if;
   end process;
end architecture;
```

Note rstb, set, and en are NOT in the sensitivity list



Synchronous inputs are inside the rising_edge

priority

rstb > set > en

Flip-Flop HDL

- Process concurrency

```
--------------------------------
--
-- process_concurrency.vhdl
--
-- created: 1/26/18
-- by: johnsontimoj
-- rev: 0
--
-- file to show concurrency of processes
--
--------------------------------
library ieee;
use ieee.std_logic_1164.all;

entity process_concurrency is
   port(
      i_clka:  in std_logic;
      i_clkb:  in std_logic;
      i_d1 :   in std_logic;
      i_d2 :   in std_logic;
      i_d3 :   in std_logic;
      i_rstb:  in std_logic;

      o_q1 :   out std_logic;
      o_q2 :   out std_logic;
      o_q3 :   out std_logic

   );
end entity process_concurrency;
```

```
architecture behavioral of process_concurrency is
begin
   process (i_clka, i_rstb)
   begin
      if (i_rstb = '0') then
         o_q1 <= '0';
      elsif (rising_edge(i_clka)) then
         o_q1 <= i_d1;
      end if;
   end process;

   process (i_clka, i_rstb)
   begin
      if (i_rstb = '0') then
         o_q2 <= '0';
      elsif (rising_edge(i_clka)) then
         o_q2 <= i_d2;
      end if;
   end process;

   process (i_clkb, i_rstb)
   begin
      if (i_rstb = '0') then
         o_q3 <= '0';
      elsif (rising_edge(i_clkb)) then
         o_q3 <= i_d3;
      end if;
   end process;

end architecture;
--------------------------------
```