

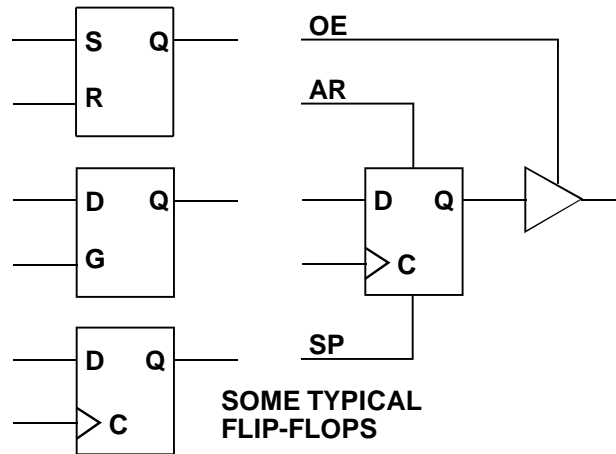
## FLIP-FLOPS AND MORE VHDL

CPE319 deals with sequential logic, more specifically referred to as **FINITE STATE MACHINES (FSM)** or **DIGITAL CONTROLLERS**. This is the subject of Part III of the Sandige text (Chapter 8).

Some typical flip-flops are shown in the diagram to the right. The first is the basic **SET/RESET (SR) FLIP-FLOP**.

According to IEEE Std. 91, no name need be placed in the box since the S and R identify the device. Technically, the Q at the output does not need to be listed.

The second one is the **GATED D LATCH** or **TRANSPARENT LATCH**. The later name comes about since  $Q = D$  as long as G is enabled. When G is not enabled, Q retains its current state.



The third flip-flop is the **RISING-EDGE TRIGGERED D FLIP-FLOP** which is the major building block of many FSM's. The flip-flop on the right is also a **RET DFF** which is typical of those found in PLD's. AR is an Asynchronous Reset and SP is a synchronous Preset.

For a more in-depth discussion of flip-flops you should refer to the Sandige text. Here we will look at these flip-flops as described in VHDL.

```

library ieee;
use ieee.std_logic_1164.all;
entity sr_ff is port(
    reset, set : in std_logic;
    q         : out std_logic
);
end sr_ff;
architecture sr of sr_ff is
begin
    process (reset, set) begin
        if reset='1' then q<='0'; end if;
        if set='1'   then q<='1'; end if;
    end process;
end sr;

```

This VHDL file describes, incompletely, the operation of the SRFF. Setting it causes the

output to be asserted (go high?). Resetting it causes the output to be deasserted. If it is neither set nor reset it, its state remains unchanged. But, what happens when it is set and reset at the same time? Generally speaking, this operation is not defined. What actually happens depends on the actual implementation of the device.

If this is programmed into a 22V10, the following (portion of the) report file is obtained.

```
PLD Compiler Software:      PLA2JED.EXE    01/MAR/97    [v4.00 ] 4 IR x77
DESIGN EQUATIONS          (20:16:22)
    q = /reset * q + set
Completed Successfully
```

This is the classical definition for such a flip-flop. Note that if both set and reset are asserted, Q will be asserted.

Below is the VHDL description of the D flip-flop found in the 22V10.

```
library ieee;
use ieee.std_logic_1164.all;
entity d_ff is port(
    d, clk, reset, set, oe : in std_logic;
    q : out std_logic
);
end d_ff;
architecture dee of d_ff is
    signal que: std_logic;
begin
    process (clk, reset, set) begin
        if reset='1' then que<='0';
        elsif (clk'event and clk='1')
            then if set='1' then que<='1'; else que<=d;
            end if;
        end if;
    end process;
    process (oe, que) begin
        if (oe='1') then q<=que; else q<='Z';
        end if;
    end process;
end dee;
```

Let's discuss this description. First, an internal signal, QUE is defined and is to be used as the direct output of the flip-flop. If RESET is asserted, QUE=0 regardless of anything else. Otherwise, nothing happens except on the rising edge of the clock. The term **clk'event** describes ANY change on the CLK signal. The **clk='1'** statement, anded with it, narrows it down to the rising edge. Actually there is another construct which may be used:

**rising\_edge(clk)**

On the rising edge, if SP is asserted, QUE goes to 1, otherwise it assumes the value currently present on the d input.

This module contains a second **PROCESS** clause. This one connect QUE to Q if the output enable (OE) is asserted. Otherwise Q is in the three-state mode. Compiling this produces almost what we might expect.

#### DESIGN EQUATIONS (20: 11: 02)

```
q. D = set + d
q. AR = reset
q. SP = GND
q. C = clk
q. OE = oe
```

Completed Successfully

What we really should have expected is shown below.

```
q. D = d
q. AR = reset
q. SP = set
q. C = clk
q. OE = oe
```

The items that are changed are underlined and in bold. For some reason the GALAXY compiler did not make use of the Synchronous Preset feature on the PAL and accomplished the same thing by sending the SET signal to the D input (correct, but not what we expected!). There may be some way to tell the compiler to use the SP input but it is not worth our time at this point to figure out what it might be.

Sandige covers several other types of flip-flops. We shall not provide the VHDL descriptions for them but make several practical comments.

The T (TOGGLE) Flip-Flop has been, since the beginning of time (or at least the beginning of flip-flops) an imaginary device. If you wanted such a flip-flop, you used a JK flip-flop and tied J to K and called the combination T. With the introduction to CPLD's, the T flip-flop has emerged as an actual device. Sometimes T flip-flops (especially in counters) are more desirable than D flip-flops. VHDL can be set to determine which type is best and select that type.

The JK Flip-flop (described by Sandige) has been the work horse of the industry (especially in FSM design) for many years. There are a few PLD's (mainly PLA's) which still use JK flip-flops but most PLDS and CPLDS use D, T or combination D/T flip-flops. (Some FPGA's may use JK's).

While we used to be concerned with the timing characteristics of flip-flops, today we are more concerned with the timing characteristics of PLD's. Actually, they are one in the same (as far as their definitions are concerned). These are all described on pages 44 and 45 of Sandiges'

text.