TP 5 CORRIGE. VHDL

[Réf. Livre VHDL (Meaudre & all …)]

1. Programme VHDL d'une Bascule RS (asynchrone)

RS 1

```
-- rs bascule
entity rs1 is
        port (R,S : in bit;       -- entree
              Q : out bit);   -- sortie
end rs1;

architecture behav of rs1 is                    -- comportemental
signal etat : bit;
begin
q <= etat;
process (R,S)
begin
        if R='0' and S='0' then
                etat <= etat;                   -- mode memoire explicite (memo implicite possible aussi en omettant ce cas)
        elsif R='0' and S='1' then
                etat <= '1';                    -- set
        elsif R='1' and S='0' then
                etat <= '0';                    -- reset
        elsif R='1' and S='1' then
                etat <= '0';                    -- indesirable
        end if;
end process;
end behav;
```

RS 2

```
-- rs bascule
entity rs2 is
        port (R,S : in bit;       -- entree
              Q : out bit);   -- sortie
end rs2;

architecture behav of rs2 is                    -- comportemental
signal etat : bit;
begin
q <= etat;
process (R,S)
begin                                                   -- mode memoire R='0' and S='0' implicite (memorisation)
        if R='0' and S='1' then
                etat <= '1';                    -- set
        elsif R='1' and S='0' then
                etat <= '0';                    -- reset
        elsif R='1' and S='1' then
                etat <= '0';                    -- indesirable
        end if;
end process;
end behav;
```

2. Programme VHDL d'une Bascule D (>0 edge triggered)

D edge 1

```
-- dedge                bascule D +edge (appelee communement bascule D)
entity dedge1 is
        port (D,clock : in bit;       -- entree + horloge
              Q : out bit);   -- sortie
end dedge1;

architecture primitive of dedge1 is
begin
        process
        begin
                wait until clock = '1';                 -- front montant
                Q <= D;
        end process;
end primitive;
```

D edge 2

```
-- dedge                bascule D +edge (appelee communement bascule D)
entity dedge2 is
        port (D,clock : in bit;       -- entree + horloge
              Q : out bit);   -- sortie
end dedge2;

architecture primitive of dedge2 is
begin
        process(clock)          -- le process ne reagit qu'au signal d'horloge
        begin
                if (clock 'event and clock ='1') then                   -- front montant
                        Q <= D;
                end if;
        end process;
end primitive;
```

3. Programme VHDL d'une Bascule T (>0 edge triggered)

La bascule T considérée a, en plus de l'entree d'horloge (clock), une entrée de validation T.
La sortie Q de la bascule T passe de 0 à 1 pour T=1 et clock active.
La sortie Q de la bascule T reste à 0 pour T=0 et clock active.
La sortie Q de la bascule T passe de 1 à 0 pour T=1 et clock active.
La sortie Q de la bascule T reste à 1 pour T=0 et clock active.
Si clock est inactive, la sortie Q de la bascule T reste à l'état antérieur.

```
-- Bascule T edge  (+edge triggered (front montant))
ENTITY tedge IS
        PORT (T, clock : IN BIT;       -- entree
              s : OUT BIT);     -- sortie
END tedge;

ARCHITECTURE primitive OF tedge IS
SIGNAL etat : BIT;
BEGIN
        s <= etat;
        PROCESS
        BEGIN
                WAIT UNTIL clock = '1';                 -- front montant d'horloge
                        IF (T = '1') THEN
                                etat <= not etat;
                        END IF;
        END PROCESS;
END primitive;
```

4. Programme VHDL d'une Bascule JK (>0 edge triggered)

JK Table de vérité

```
-- Bascule JK d'apres la table de verite (+edge triggered (front montant))
entity jk1 is
        port (j,k,clock : in bit;       -- entree
              q : out bit);     -- sortie
end jk1;

architecture primitive of jk1 is
signal etat : bit;
begin
process
        begin
        wait until clock = '1';             -- front montant
                if (j='1' and k='1') then
                                    etat <= not etat;
                elsif (j='1' and k='0') then
                                    etat <= '1';
                elsif (j='0' and k='1') then
                                    etat <= '0';
                -- le cas j='0' et k='0' n'est pas renseigne pour sequentiel (memorisation)
                end if;
        end process;
q <= etat;
end primitive;
```

JK Automate

```
-- Bascule JK d'apres l'automate  (+edge triggered (front montant))
entity jk2 is
    port (j,k,clock : in bit;       -- entree
          q : out bit);     -- sortie
end jk2;

architecture primitive of jk2 is
signal etat : bit;
begin
process
    begin
    wait until clock = '1'; -- front montant
      if (j='1' and etat='0') then
            etat <= '1';
      elsif (k='1' and etat='1') then
            etat <= '0';
      -- les autres cas de figure d'entrees ne sont pas renseignes pour sequentiel (memorisation)
      end if;
    end process;
q <= etat;
end primitive;
```

JK Etat interne

```
-- Bascule JK a partir de l'etat interne de la bascule et non plus des commandes j et k  (+edge triggered)
entity jk3 is
      port (j,k,clock : in bit;        -- entree
               q : out bit);      -- sortie
end jk3;

architecture primitive of jk3 is
signal etat : bit;
begin
q <= etat;
process
      begin
      wait until clock = '1';           --  front montant
         case etat is
         when '0'    =>
            if (j='1') then
                  etat <= '1';
            end if;
         when '1'    =>
            if (k='1') then
                  etat <= '0';
            end if;
         end case;               -- les autres cas d'entrees ne sont pas renseignes pour sequentiel (memorisation)
      end process;
end primitive;
```

JK Etat interne (variante)

```
-- Bascule JK a partir de l'etat interne de la bascule et non plus des commandes j et k (variante)  (+edge)
entity jk4 is
      port (j,k,clock : in bit;        -- entree
               q : out bit);      -- sortie
end jk4;

architecture primitive of jk4 is
signal etat : bit;
begin
q <= etat;
process(clock)
      begin
      if (clock = '1' and clock 'event) then                    -- front montant
         case etat is
         when '0'    =>
            if (j='1') then
                  etat <= '1';
            end if;
         when '1'    =>
            if (k='1') then
                  etat <= '0';
            end if;
         end case;               -- les autres cas d'entrees ne sont pas renseignes pour sequentiel (memorisation)
      end if;
      end process;
end primitive;
```