Architecture des ordinateurs - TD 4 : Automates

Sur un exemple, nous allons voir comment coder le fonctionnement synchrone d'un composant en utilisant un automate à états.

1 Composant d'émission d'un octet

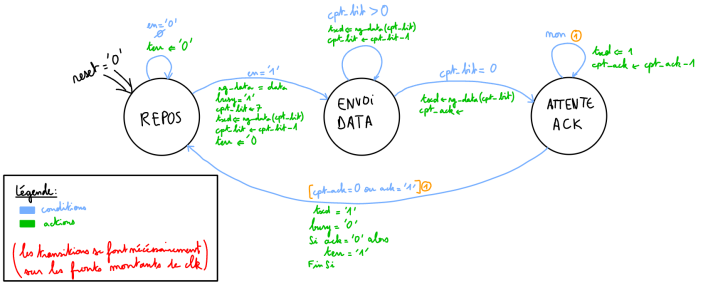
Nous souhaitons développer un composant synchrone sur une horloge **clk** dont le rôle est d'émettre un octet selon un protocole série.
Ce composant est par défaut au repos (ligne série **txd** à '1'), en attente d'un ordre d'émission. Sur le premier front montant de **clk** où il voit le signal **en** (pour *enable*), à '1',

1. il stocke l'octet donnée en entrée sur le bus **data**,
2. il indique qu'il est occupé (signal **busy** à '1')
3. il envoie sur la ligne série **txd**, le bit de poids fort (7) de l'octet

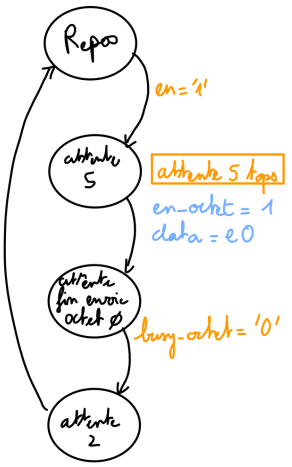
puis, à chaque front montant de **clk**, il envoie les bits suivants de 6 à 0.
Une fois qu'il a envoyé les 8 bits, il remet la ligne série à '1' puis il attend, au plus 5 fronts montant de **clk**, la confirmation de la réception (signal **ack** actif à '1') pour se remettre au repos (**busy** à '0').
Si le confirmation de la réception n'arrive pas, il prévient le composant qui donne les ordres en positionnant un signal **terr** (erreur de transmission) à '1' pendant une période de l'horloge.
A tout moment, et de manière asynchrone, un **reset** (actif à '0') remet le composant au repos.

Schéma

Antoine Rey le crack des schémas grâce à sa tablette de fou 📝




```
entity envoiOctet is
    port (
        clk, reset, en: in std_logic;
        data: in std_logic_vector(7 downto 0);
        txd, busy, terr: out std_logic;
    );
end envoiOctet;


architecture behavioral of envoiOctet is

    type t_etat is (repos, Envoi_data, attente_ack);
    signal etat: t_etat;

begin

    process(clk, reset)
        -- VARIABLES
        variable rg_data: std_logic_vector(7 downto 0);
        variable cpt_bit: natural;
        variable cpt_ack: natural;
    begin

        if (reset = '0') then
            -- RESET
            busy <= '0';
            terr <= '0';
            txd <= '1';
            etat <= repos;
            cpt_bit := 7;
            cpt_ack := 5;
            rg_data := (others => 'U');
        elsif (rising_edge(clk)) then
            -- MAIN
            case (etat) is
                when repos =>
                    terr <= '0';
                    if (en = '1') then
                        rg_data := data;
                        busy <= '1'
                        cpt_bit <= 7;
                        txd <= rg_data(cpt_bit);
                        etat <= envoi_data;
                        cpt_bit := cpt_bit - 1;
                    end if;
                when envoi_data =>
                    if (cpt_bit > 0) then
                        txd <= rg_data(cpt_bit);
                        cpt_bit := cpt_bit-1;
                        -- inutile : etat <= envoi_data;
                    else
                        cpt_ack := 5;
                        txd <= rg_data(cpt_bit);
                        etat <= attente_ack;
                    end if;

                when attente_ack =>
                    busy <= '0';
                    txd <= '1';
                    if (cpt_ack > 0 and ack /= '1') then
                        cpt_ack := cpt_ack - 1;
                    else
                        terr := not (ack = '1');
                        etat <= repos;
                    end if;
            end case;
        end if;

    end process;

end behavioral;
```



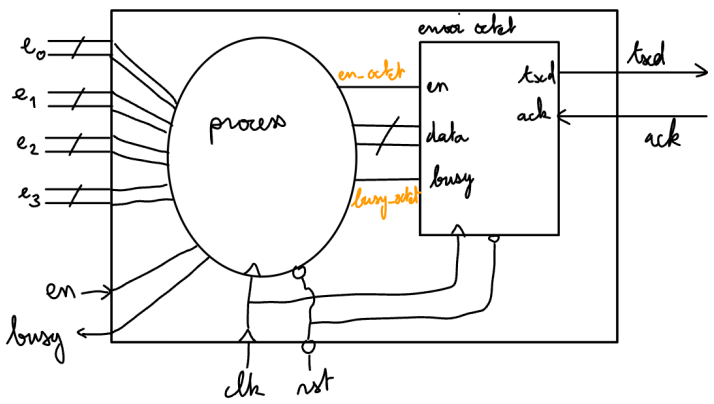
Code VHDL

2 Composant émettant 4 octets

On souhaite utiliser l'émetteur d'un octet dans un nouveau composant qui, lui, permettra d'émettre 4 octets à la suite [1] (https://sanik.inpt.fr/pU-RySqURHSb6U5ALM87Lg?view#fn1). Avant l'émission du premier octet, il y aura un temps d'attente de 5 tops d'horloge et entre chaque octet, un temps d'attente de 2 tops (quel est le moment où un octet est considéré comme envoyé ?).

[Question] : Développez un tel composant qui sera utile pour le mini-projet.

Schémas



```
entity envoi4octets is
    port (
        o1, o2, o3, o4: in std_logic_vector(7 downto 0);
        clk, reset, en : in std_logic;
        busy, txd : out std_logic;
    );
end envoi4octets;


architecture ... of envoi4octets is

    component envoiOctet
        port (
            clk, reset, en: in std_logic;
            data: in std_logic_vector(7 downto 0);
            txd, busy, terr: out std_logic;
        );
    end component;

    type t_etat is (repos, attente5, envoi, attente2);
    signal etat : t_etat;
    signal busy_octet : std_logic;
    signal en_octet : std_logic;
    signal terr_octet : std_logic;
    signal data_octet : std_logic_vector(7 downto 0);
begin

    Serial: envoiOctet port map(clk, reset, en_octet, data, txd, busy_octet, terr_o

    process(clk, reset)
        variable cpt_octet : natural; --indique quel octet on doit envoyer
    begin

        if(reset = '0') then
            -- RESET
            -- @TODO
        elsif (rising_edge(clk)) then
            -- MAIN
            case (etat)
                when repos =>

                when attente5 =>

                when attente2 =>

                when envoi =>

            end case;
        end if;
    end process;

end ...;
```

1. on considérera qu'il n'y a guerre d'erreur de transmission et que le signal **terr** ne sera jamais actionné ↩ (https://sanik.inpt.fr/pU-RySqURHSb6U5ALM87Lg?view#fnref1)