TD 1 : Exercices simples de synthèse logique

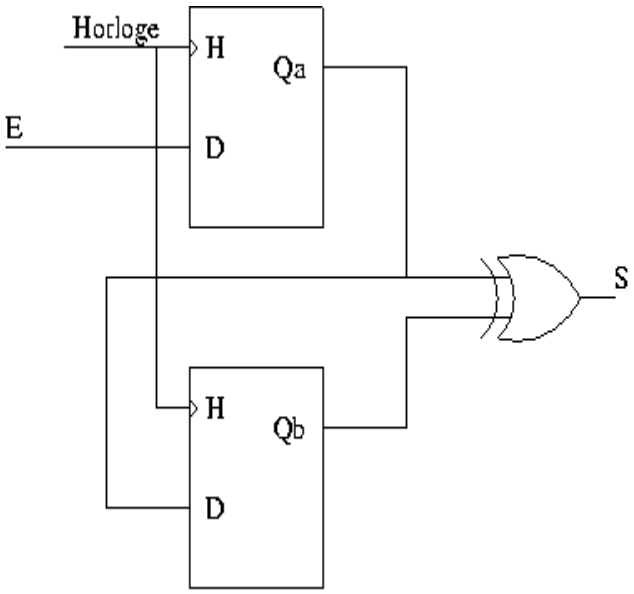
TD – EE341

Exercice 1.1 :

On considère le programme ci-dessous :

1. **Déduire de ce programme, par une construction méthodique, un schéma (bascules et portes logiques).**
2. **Compléter le chronogramme Fig 1.1.**

```
entity transitm is
  port (hor, e : in  bit;
        s      : out bit);
end transitm;

architecture quasi_struct of transitm is
  signal qa, qb : bit;
begin

  s <= qa xor qb;

  schem : process(hor)
  begin
    if hor'event and hor = '1' then
      qa <= e;
      qb <= qa;
    end if;
  end process schem;
end quasi_struct;
```



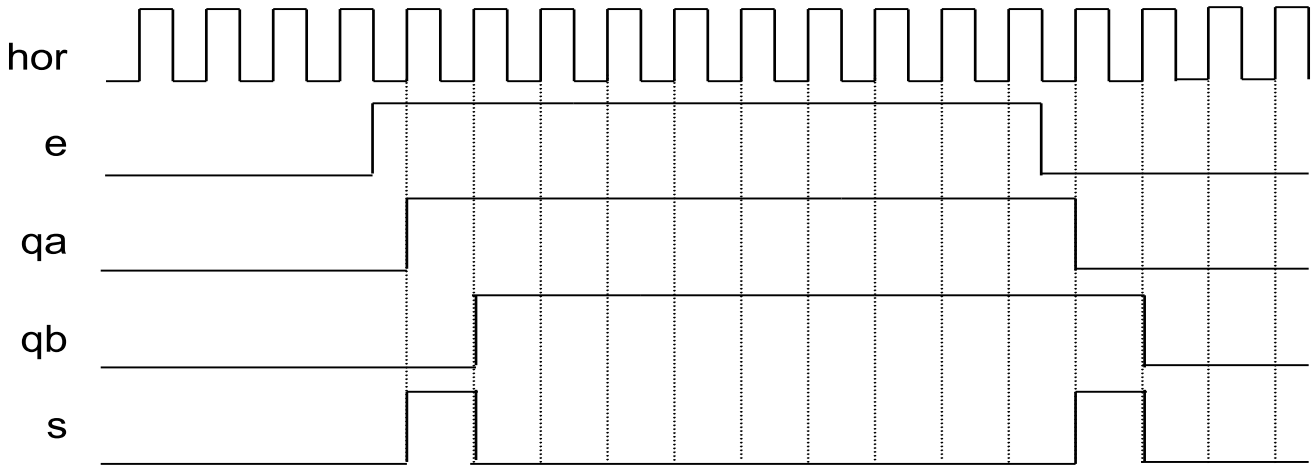


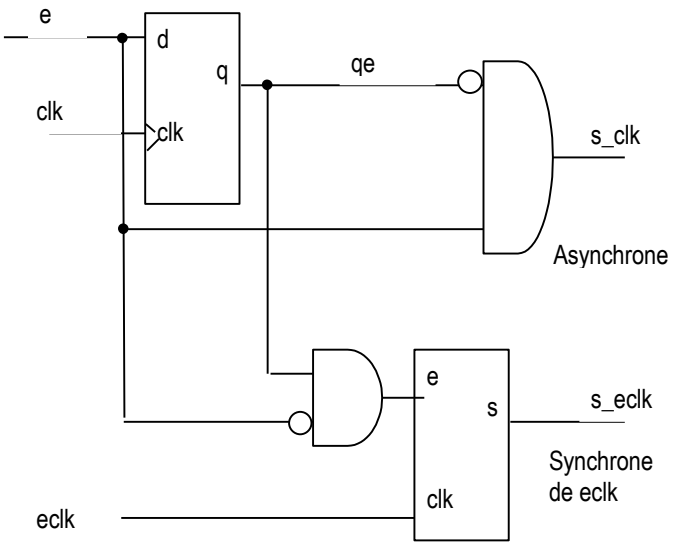
Fig 1.1 : Chronogramme à compléter

Exercice 1.2 :

On considère le programme ci-dessous :

1. **Compléter le code manquant ;**
2. **Déduire un schéma (bascules et portes logiques) de ce programme ;**
3. **Caractériser les sorties : synchrones ou asynchrones ?**
4. **Compléter le chronogramme Fig 1.2.**

```
entity condit is
  port (
    clk    : in  bit;
    eclk   : in  bit;
    e      : in  bit;
    s_clk  : out bit;
    s_eclk : out bit);
end condit;

architecture reg_lvl of condit is
    signal qe : bit ;
begin

  s_clk <= e and not(qe);

  ech1 : process (clk)
  begin
    if clk'event and clk = '1' then
      qe <= e;
    end if;
  end process ech1;

  ech : process (eclk)
  begin
    if eclk'event and eclk = '1' then
      s_eclk <= (qe and not(e));
    end if;
  end process ech;

end reg_lvl;
```



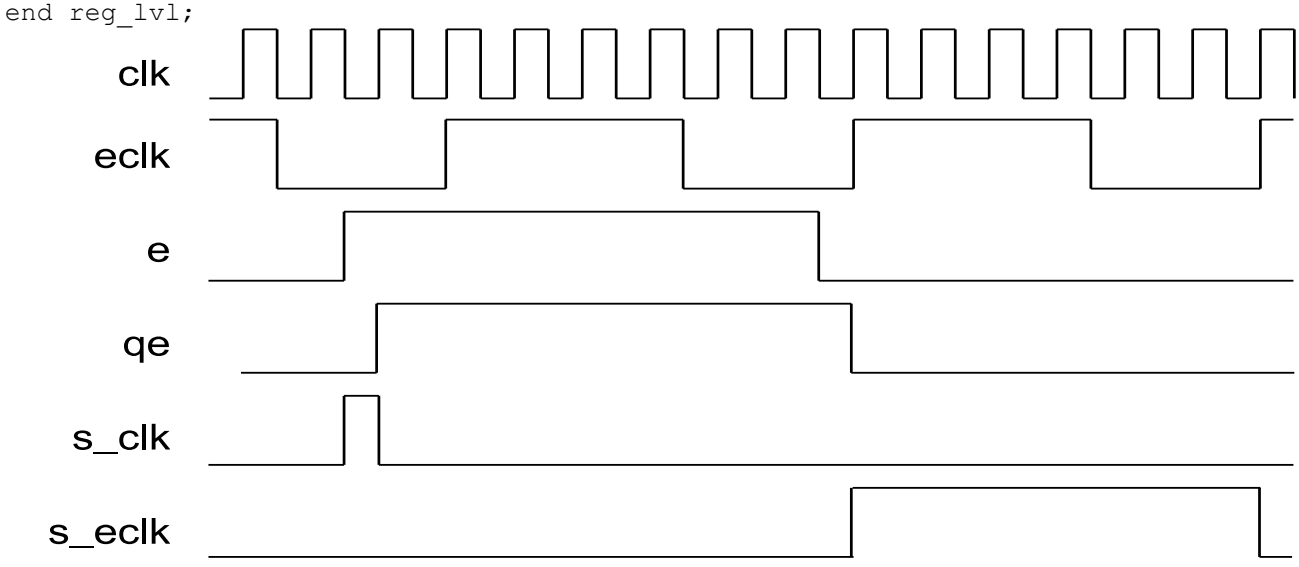


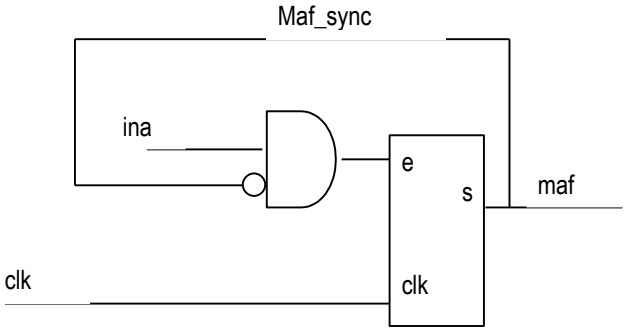
Fig 1.2 : Chronogramme à compléter

Exercice 1.3 :

On considère le programme ci-dessous :

1. **Ce code est il correct ?**
2. **Déduire un schéma (bascules et portes logiques) de ce programme ;**
3. **Compléter le chronogramme Fig 1.3.**

```
entity mise_feu is
  port (
    clk : in  bit;
    ina : in  bit;
    maf : out bit);
end mise_feu;

architecture detect of mise_feu is
    signal maf_sync : bit;
begin
   maf <= maf_sync;
  sync : process (clk)
  begin
    if clk'event and clk = '1' then
      maf_sync <= ina and not maf_sync;
    end if;
  end process sync;
end detect;
```



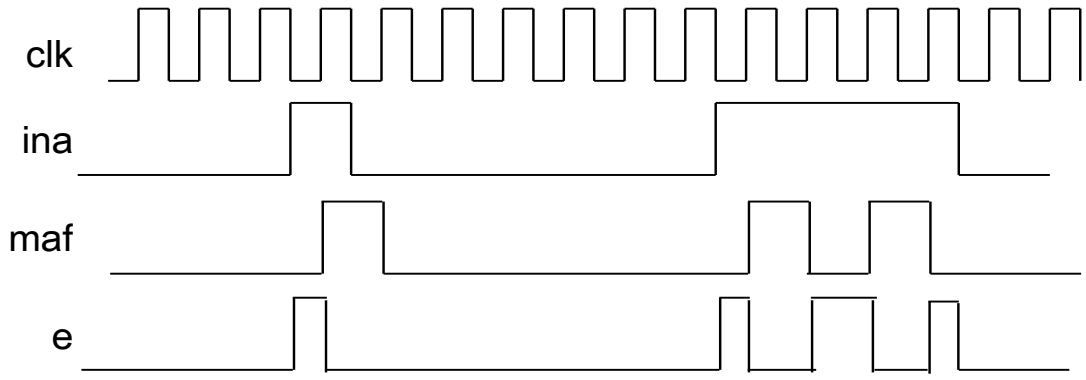


Fig 1.3 : Chronogramme à compléter

Exercice 2.1 :

On considère la fonction logique de la figure 2.1:

1. **Compléter le programme pour réaliser cette fonction ;**
2. **Réaliser un chronogramme de test de cette fonction (compléter le chronogramme fig 2.2).**

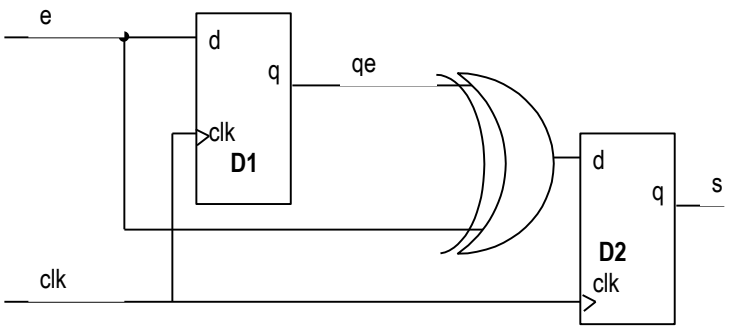


Fig 2.1 Bloc fonctionnel

```
entity detect is
  port (
    clk : in  bit;
    e   : in  bit;
    s   : out bit);
end detect;

architecture reg of detect is

begin

  sync : process (clk)
  begin
    if clk'event and clk = '1' then
      qe <= e;                          --D1
      s  <= qe xor e;                   --D2
    end if;
  end process sync;

end reg;
```

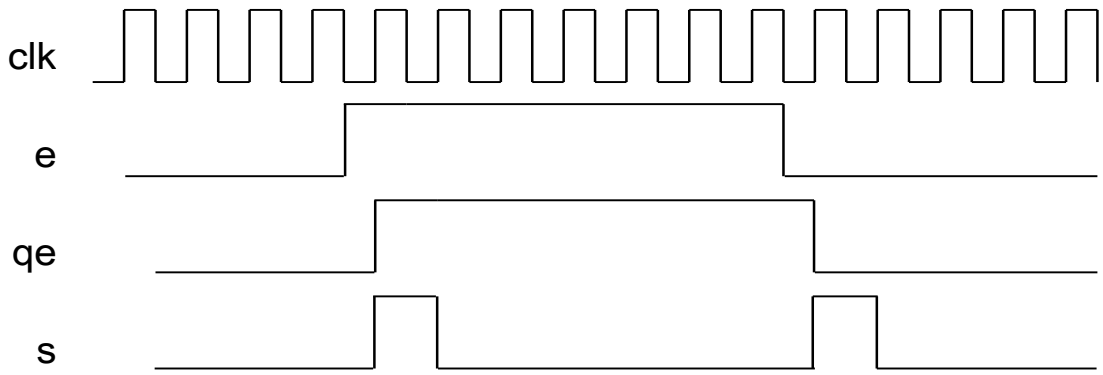


Fig 2.2 : Chronogramme à compléter

Exercice 2.2 :

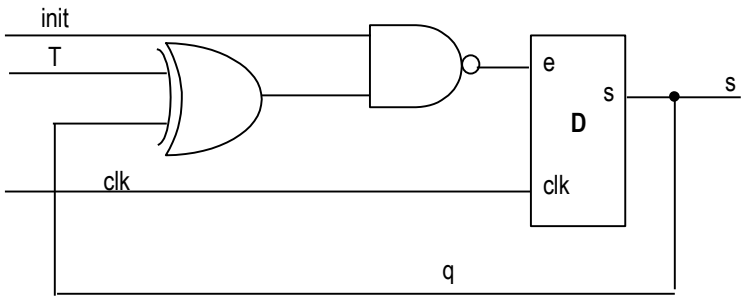
On considère l'équation logique ci-dessous :

$$Q_{(t-1)} = /(\text{init} \cdot (T \oplus Q_{(t)}))$$

1. **Concevoir le schéma de réalisation de cette équation.**
2. **Compléter le code VHDL pour réaliser cette fonction.**
3. **Le cahier des charges évolue : il faut transformer le signal init en une commande « actif bas » (si init='0'), de remise à zéro asynchrone.**
4. **Transformer le programme pour qu'il utilise le type std_logic. Rajouter une commande de haute impédance « oen » active basse, qui pilote la sortie uniquement lorsque « oen » est actif.**

```
entity basc is
  port (T, clk, init : in  bit;
        s            : out bit);
end basc;

architecture primitive of basc is
  signal q : bit;
begin
```

1. Schéma



2. Code VHDL

```
entity basc is
  port (T, clk, init : in  bit;
        s            : out bit);
end basc;

architecture primitive of basc is
  signal q : bit;
begin
  s <= q;
  process (clk)
  begin
    if clk'event and clk = '1' then
      q <= not (init and (T xor q));
    end if;
  end process;
end primitive;
```

3. Programme avec la commande init en remise à zéro :

```
entity basc is
  port (T, clk, init : in  bit;
        s            : out bit);
end basc;

architecture primitive of basc is
  signal q : bit;
begin
  s <= q;
  process (clk, init)
  begin
    if init = '0' then
      q <= '0';
    elsif clk'event and clk = '1' then
        q <= not (T xor q);
    end if;
  end process;
end primitive;
```

4. Programme avec le type std_logic et oe :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity basc is
   port (
         T, clk, init, oe: in std_logic;
         s : out std_logic );
end basc;

architecture primitive of basc is
   signal q : std_logic;
begin

   s <= q when oe = '0' else 'Z' ;

   process(clk, init)
   begin
      if init = '0' then
         q <= '0' ;
      elsif clk'event and clk ='1' then
        q <= not (T xor q);
      end if;
  end process;
end primitive;
```