TP Chapitre 03 : Construction de la Mémoire

Objectifs pratiques

- Implémenter le registre 1-bit
- Construire un registre multi-bits
- Assembler le compteur de programme
- Créer une RAM adressable

Durée estimée : 2h

Prérequis : TP Chapitre 01 terminé (Mux, DMux)

Préparation

Accès au Simulateur

👉 **Ouvrir le Simulateur HDL**

Allez dans **HDL Progression → Projet 4 : Séquentiel**

Note : La DFF est fournie comme primitive.

Exercice 1 : Registre 1-bit (BitReg)

Objectif : Créer une cellule mémoire persistante

Spécification

- Si `load = 1` : stocker `d` au prochain front
- Si `load = 0` : conserver la valeur

Schéma

👉 **Ouvrir l'exercice BitReg**

Code à compléter

```
entity BitReg is
  port(
    d    : in bit;
    load : in bit;
    q    : out bit
  );
end entity;

architecture rtl of BitReg is
  component Mux port(a,b,sel : in bit; y : out bit); end component;
  component DFF port(d : in bit; q : out bit); end component;
  signal mux_out, dff_out : bit;
begin
  -- TODO: Compléter
  -- u_mux: Mux port map (a => ?, b => ?, sel => ?, y => ?);
  -- u_dff: DFF port map (d => ?, q => ?);
  -- q <= ?;
end architecture;
```

▸ Solution

Exercice 2 : Registre 16-bits

Objectif : Paralléliser les BitReg

Spécification

- Entrée : d[15:0], load
- Sortie : q[15:0]
- Tous les bits partagent le même signal `load`

Structure

16 instances de BitReg en parallèle.

👉 **Ouvrir l'exercice Register16**

Code avec generate

```
entity Register16 is
  port(
    d    : in bits(15 downto 0);
    load : in bit;
    q    : out bits(15 downto 0)
  );
end entity;

architecture rtl of Register16 is
  component BitReg
    port(d, load : in bit; q : out bit);
  end component;
begin
  gen: for i in 0 to 15 generate
    bit_i: BitReg port map (
      d => d(i),
      load => load,
      q => q(i)
    );
  end generate;
end architecture;
```

Exercice 3 : Compteur de Programme (PC)

Objectif : Implémenter les 4 modes

Spécification

Priorité	Mode	Condition	Action
1	reset	reset=1	PC ← 0
2	load	load=1	PC ← in
3	inc	inc=1	PC ← PC+1
4	hold	sinon	PC ← PC

👉 **Ouvrir l'exercice PC**

Architecture suggérée

Code à compléter

```
entity PC is
  port(
    in_val : in bits(15 downto 0);
    reset  : in bit;
    load   : in bit;
    inc    : in bit;
    out_val: out bits(15 downto 0)
  );
end entity;

architecture rtl of PC is
  component Register16 port(d : in bits(15 downto 0); load : in bit; q : out bits(15 downto 0)); end component;
  component Inc16 port(a : in bits(15 downto 0); y : out bits(15 downto 0)); end component;
  component Mux16 port(a,b : in bits(15 downto 0); sel : in bit; y : out bits(15 downto 0)); end component;

  signal reg_out, incremented, after_inc, after_load, to_reg : bits(15 downto 0);
  constant ZERO : bits(15 downto 0) := (others => '0');
begin
  -- TODO: Compléter
end architecture;
```

Solution PC

▸ Solution

Exercice 4 : RAM8

Objectif : Mémoire de 8 mots

Spécification

- 8 mots de 16 bits
- Adresse sur 3 bits
- `load` = 1 : écrire à l'adresse
- Lecture toujours active

Structure

👉 **Ouvrir l'exercice RAM8**

Code à compléter

```
entity RAM8 is
  port(
    input   : in bits(15 downto 0);
    address : in bits(2 downto 0);
    load    : in bit;
    output  : out bits(15 downto 0)
  );
end entity;

architecture rtl of RAM8 is
  component Register16 port(d : in bits(15 downto 0); load : in bit; q : out bits(15 downto 0)); end component;
  component DMux8Way port(input, sel : in bit; a,b,c,d,e,f,g,h : out bit); end component;
  component Mux8Way16 port(a,b,c,d,e,f,g,h : in bits(15 downto 0); sel : in bits(2 downto 0); y : out bits(15 downto 0)); end component;

  signal load0,load1,load2,load3,load4,load5,load6,load7 : bit;
  signal r0,r1,r2,r3,r4,r5,r6,r7 : bits(15 downto 0);
begin
  -- TODO: Compléter
end architecture;
```

Solution RAM8

▸ Solution

Exercice 5 : RAM64

Objectif : Composition hiérarchique

Spécification

- 64 mots = 8 × RAM8
- Adresse sur 6 bits :
  - `address[5:3]` : quelle RAM8
  - `address[2:0]` : quel mot dans RAM8

👉 **Ouvrir l'exercice RAM64**

Code à compléter

```
entity RAM64 is
  port(
    input   : in bits(15 downto 0);
    address : in bits(5 downto 0);
    load    : in bit;
    output  : out bits(15 downto 0)
  );
end entity;

architecture rtl of RAM64 is
  component RAM8 port(input : in bits(15 downto 0); address : in bits(2 downto 0); load : in bit; output : out bits(15 downto 0)); end component;
  component DMux8Way port(input : in bit; sel : in bits(2 downto 0); a,b,c,d,e,f,g,h : out bit); end component;
  component Mux8Way16 port(a,b,c,d,e,f,g,h : in bits(15 downto 0); sel : in bits(2 downto 0); y : out bits(15 downto 0)); end component;

  signal load0,load1,load2,load3,load4,load5,load6,load7 : bit;
  signal o0,o1,o2,o3,o4,o5,o6,o7 : bits(15 downto 0);
begin
  -- Bits de poids fort pour sélectionner la RAM8
  -- Bits de poids faible pour sélectionner le mot dans la RAM8
  -- TODO: Compléter
end architecture;
```

Solution RAM64

▸ Solution

Récapitulatif

Composants implémentés

Composant	Complexité	Utilise
BitReg	★	Mux, DFF
Register16	★	16× BitReg
PC	★★	Register16, Inc16, Mux16
RAM8	★★	DMux8Way, 8× Register16, Mux8Way16
RAM64	★★	DMux8Way, 8× RAM8, Mux8Way16

Validation Finale

Checklist

- [ ] BitReg maintient sa valeur quand load=0
- [ ] Register16 capture tous les bits simultanément
- [ ] PC respecte les priorités (reset > load > inc)
- [ ] RAM8 lit/écrit à la bonne adresse
- [ ] RAM64 décompose correctement l'adresse

Prochaine étape

➡️ **Chapitre 04 : Architecture Machine** — Définir l'ISA du CPU !

📚 **Référence :** Livre Seed, Chapitre 03 - Mémoire