

# SOMMAIRE

- 6 Mémoires internes au FPGA
  - Mémoires internes au FPGA

# ROM

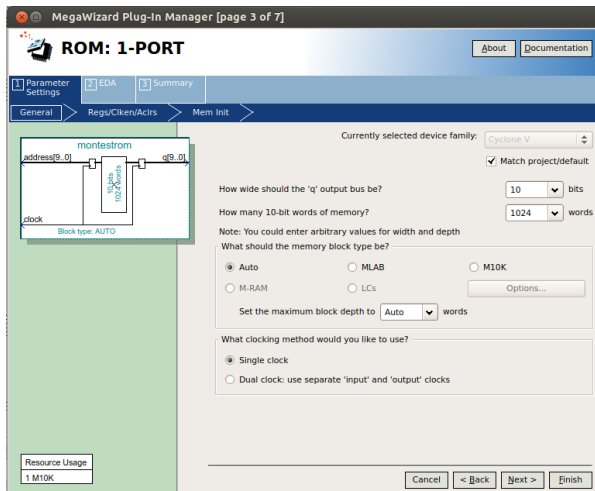
- ❑ Les FPGA des familles Cyclone (I) étaient les derniers à inclure des ROM totalement asynchrones (adresses et données)
- ❑ Depuis les Cyclones II, les adresses sont nécessairement synchrones
- ❑ Les données peuvent être synchrones ou asynchrones
- ❑ Le contenu est précisé par un fichier .mif ou .hex explicite ou généré par une fonction VHDL
- ❑ Il existe une version double port, permettant de lire simultanément le contenu à deux adresses distinctes

## ROM : CRÉATION PAR ASSISTANT GRAPHIQUE

- Pour pouvoir créer une ROM de façons graphique il faut au préalable créer un fichier d'initialisation du contenu (requis par l'outil)
- Créer un nouveau fichier d'initialisation **File** → **New** → **Memory Initialization File**
- Choisir 1024 mots de 10 bits
- Enregistrer sous **montestrom.mif**
- Clic droit dans le tableau crée **Custom Fill Cells**
- Remplir par exemple avec un compteur partant de 0
- On peut alors démarrer l'assistant de création de composants **Tools** → **MegaWizard Plug-in Manager**
- Choisir une mémoire de type ROM: **1-PORT** et créer un nouveau composant VHDL nommé **montestrom.vhd**

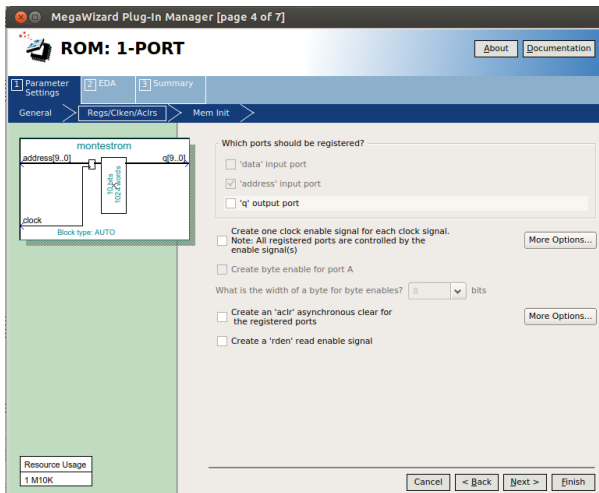
# ROM : CRÉATION PAR ASSISTANT GRAPHIQUE

## CONFIGURATION GÉNÉRALE



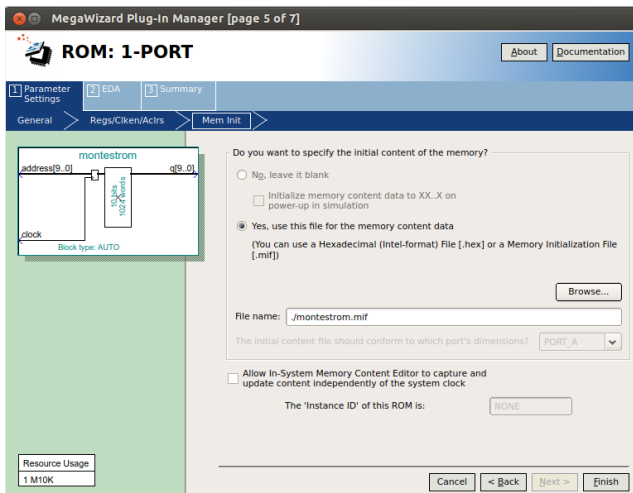
# ROM : CRÉATION PAR ASSISTANT GRAPHIQUE

## CONFIGURATION



# ROM : CRÉATION PAR ASSISTANT GRAPHIQUE

## CONFIGURATION



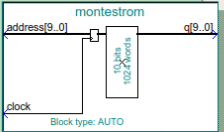
# ROM : CRÉATION PAR ASSISTANT GRAPHIQUE

## CONFIGURATION

MegaWizard Plug-In Manager [page 7 of 7]

**ROM: 1-PORT** [About](#) [Documentation](#)

1 Parameter Settings 2 EDA 3 Summary



Turn on the files you wish to generate. A gray checkmark indicates a file that is automatically generated, and a green checkmark indicates an optional file. Click Finish to generate the selected files. The state of each checkbox is maintained in subsequent MegaWizard Plug-In Manager sessions.

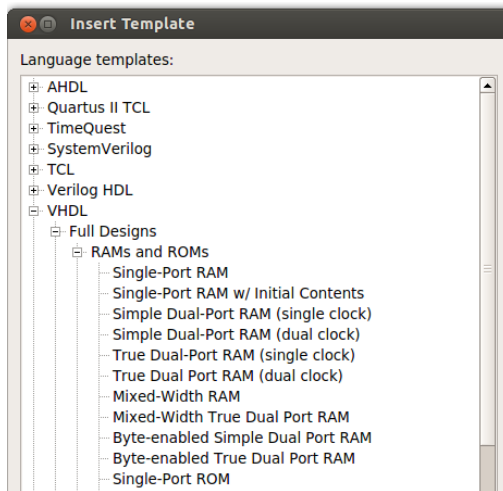
The MegaWizard Plug-In Manager creates the selected files in the following directory:  
/home/berville5/tmp/Quartus/TelecomNancy/exemple1/

File	Description
<input checked="" type="checkbox"/> montestrom.vhd	Variation file
<input type="checkbox"/> montestrom.inc	AHDL Include file
<input type="checkbox"/> montestrom.cmp	VHDL component declaration file
<input type="checkbox"/> montestrom.bsf	Quartus II symbol file
<input checked="" type="checkbox"/> montestrom_inst.vhd	Instantiation template file

Resource Usage  
1 M10K

Cancel < Back Next > Finish

# ROM : CRÉATION PAR MODÈLE VHDL





# ROM : CRÉATION PAR MODÈLE VHDL

```
entity single_port_rom is
  generic
  (
    DATA_WIDTH : natural := 8;
    ADDR_WIDTH  : natural := 8
  );
  port
  (
    clk      : in std_logic;
    addr     : in natural range 0 to 2**ADDR_WIDTH - 1;
    q        : out std_logic_vector((DATA_WIDTH - 1) downto 0)
  );
end entity;

architecture rtl of single_port_rom is
  -- Build a 2-D array type for the RoM
  subtype word_t is std_logic_vector((DATA_WIDTH-1) downto 0);
  type memory_t is array(2**ADDR_WIDTH-1 downto 0) of word_t;

  function init_rom
    return memory_t is
```

# ROM : CRÉATION PAR MODÈLE VHDL

```
variable tmp : memory_t := (others => (others => '0'));
begin
  for addr_pos in 0 to 2**ADDR_WIDTH - 1 loop
    -- Initialize each address with the address itself
    tmp(addr_pos) := std_logic_vector(to_unsigned(addr_pos,
DATA_WIDTH));
  end loop;
  return tmp;
end init_rom;

-- Declare the ROM signal and specify a default value. Quartus II
-- will create a memory initialization file (.mif) based on the
-- default value.
signal rom : memory_t := init_rom;
begin
  process(clk)
  begin
    if(rising_edge(clk)) then
      q <= rom(addr);
    end if;
  end process;
end rtl;
```

# ROM : CRÉATION PAR MODÈLE VHDL

# SOMMAIRE

- 7 Processeur
  - Processeur

# PROCESSEUR

## INTRODUCTION

Un processeur minimaliste de caractéristiques suivantes :

- Unité arithmétique et logique (ALU) d'une largeur de 16 bits (addition, chargement et maintien)
- Les instructions sont codées sur 16 bits et leurs adresses sur 7 bits.
- Les données sont codées sur 16 bits et occupent 128 adresses.

Le jeu d'instruction :

Mnémonique	Opcode	Description
load MEM	00000010 $d_7d_6...d_1d_0$	charge l'accu avec le mot d'adresse d[7..0]
store MEM	00000001 $d_7d_6...d_1d_0$	écrit le contenu de l'accu à l'adresse d[7..0]
add MEM	00000000 $d_7d_6...d_1d_0$	ajoute l'accu au contenu de d[7..0]
jump ADR	00000011 $d_6d_5...d_1d_0$	charge le compteur PC avec l'adresse d[6..0]

# PROCESSEUR

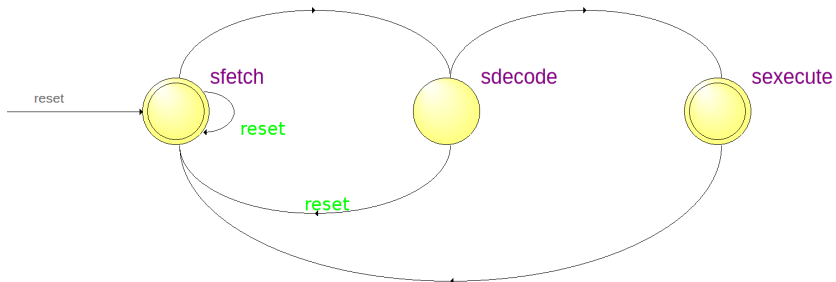
## INTRODUCTION

Chaque instruction se décompose en 3 phases (trois cycles machine) :

- **Recherche d'instructions (Fetch)** : lecture de l'instruction pointée par PC et stockage dans le registre d'instructions (IR).
- **Décodage d'instructions (Decode)** : détermination du type d'instruction et recherche des opérandes.
- **Exécution (Execute)** : réalisation des opérations et stockage éventuel des résultats.

# PROCESSEUR

## INTRODUCTION



# PROCESSEUR

## INTRODUCTION

Ressources nécessaires :

- La mémoire programme (ROM) comporte 128 mots de 16 bits.
- La mémoire de données (RAM) comporte 128 mots de 16 bits.
- Le compteur programme (PC) 7 bits stocke l'adresse de l'instruction suivante à traiter.
- Le registre d'instructions (IR) 16 bits stocke l'instruction (et l'opérande) en cours.
- Le PC qui est un registre qui peut être incrémenté ou chargé avec IR[6..0].



# PROCESSEUR

## INTRODUCTION

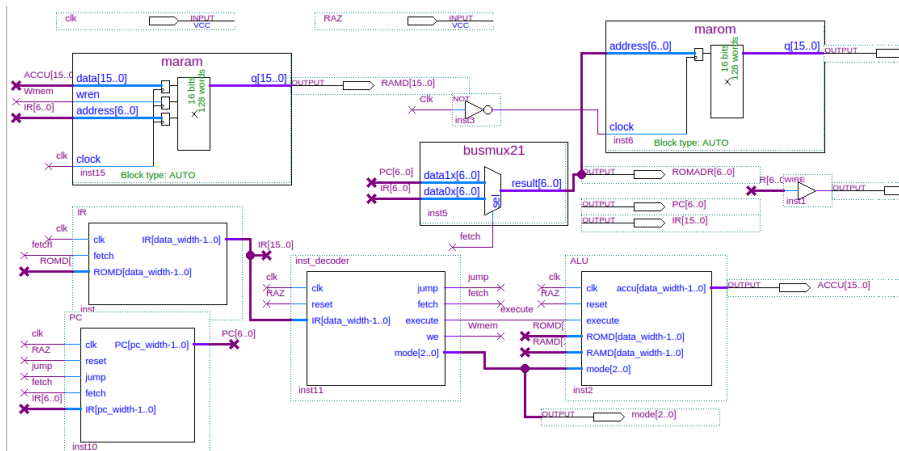
- Une unité arithmétique et logique (ALU) qui traite 2 mots de 16 bits et fournit un résultat sur 16 bits.
- Un accumulateur (registre 16 bits) qui stocke les résultats de l'ALU.
- L'ALU opère sur le contenu de l'accumulateur et les 16 bits de la RAM ou de la ROM.
- La seule source d'écriture des données est l'accumulateur.

Le décodeur d'instructions :

- A partir des opcodes (IR) et de la phase d'exécution il fournit les signaux de contrôle permettant aux ressources d'exécuter l'instruction.

# PROCESSEUR

## SCHEMA COMPLET



# ROM

## CONTENU

```

tmp(0) := x"0210"; -- CHARGE l'accu avec MEM(0x10)
tmp(1) := x"0011"; -- AJOUTE MEM(0x11) à l'accu
tmp(2) := x"0185"; -- STOCKE l'accu à l'adresse 0x85
tmp(3) := x"0285"; -- CHARGE l'accu avec MEM(0x85) pour
    vérifier le résultat: 0xFFFF
tmp(4) := x"0304"; -- JUMP à l'adresse 4 (boucle
    infinie)
tmp(16#10#) := x"AAAA"; -- Donnée à l'adresse 0x10
tmp(16#11#) := x"5555"; -- Donnée à l'adresse 0x11
    return tmp;
    
```

# EXERCICE

## MICROPROCESSEUR MINIMALISTE

- 1 Réalisez le décodeur d'instruction de ce processeur.  
Déterminez le type de la machine à utiliser (justifiez votre réponse).
- 2 Testez votre décodeur d'instruction pour chacune des instructions de ce processeur.
- 3 Réalisez complètement le processeur présenté
- 4 Initialisez la ROM avec le contenu du programme ci-dessus
- 5 Vérifiez en simulation le bon fonctionnement du processeur