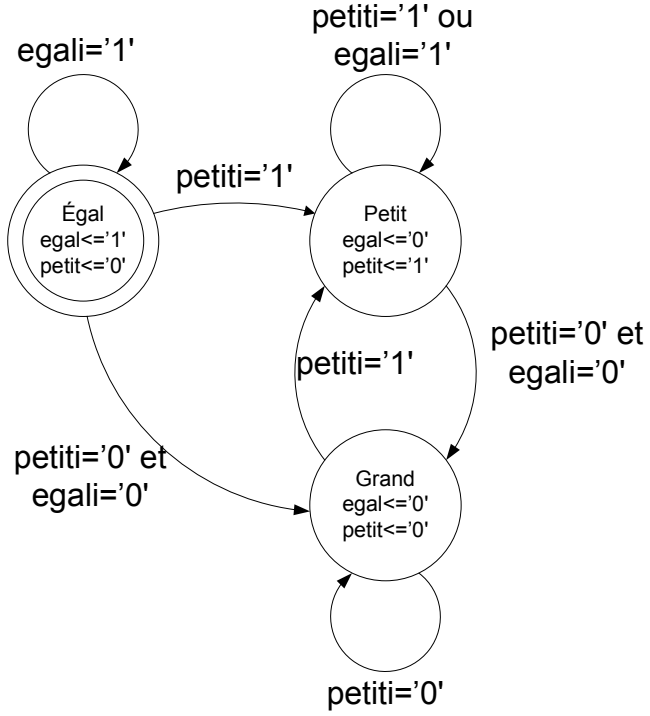
1.
a)
```
architecture arch of compareBit is
begin
   egali <= '1' when Xi = Yi else '0';                    (ou    egali <= not (Xi xor Yi);
   petiti <= '1' when Xi = '0' and Yi = '1' else '0';     (ou    petiti <= (not Xi) and Yi;
end arch;
```

b)




c)
```
architecture arch of compareSerie is
   component compareBit
      port( Xi, Yi : in std_logic; egali, petiti : out std_logic );
   end component;

   signal egali, petiti : std_logic;
begin
   cb: compareBit  port map(Xi, Yi, egali, petiti);

   process(clk,init)
   begin
      if init='1' then
         egal <= '1';
         petit <= '0';
      elsif rising_edge(clk) then
         if petiti='1' then
```

```
          petit <= '1';
          egal <= '0';
        elsif egali='0' then
          petit <= '0';
          egal <= '0';
        end if;
      end if;
  end process;
end arch;
```

d)
Poids faible, avantage : La comparaison peut commencer avant avoir fini une opération arithmétique précédente (les opérations d'aditions et de multiplications ne se font pas à partir des bits de poids fort).
Poids fort, avantage : On peut souvent savoir la réponse avant d'avoir passé tous les bits.

2.
a)

```
architecture arch of johnsonAbinaire is
begin
  process(J)
    variable val : integer;
  begin
    val := 0;
    for k in n-1 downto 1 loop
      if J(k) /= j(0) then
        val := k;
      end if;
    end loop;

    if (J(n-1) = '1') then
      val := val+n;
    end if;

    B <= std_logic_vector(to_unsigned(val, nB));
  end process;
end arch;
```

b)

```
architecture arch of johnsonAbinaireTB is
  component johnsonAbinaire
    generic ( n : positive; nB : positive );
    port( J : in std_logic_vector(n-1 downto 0); B : out std_logic_vector(nB-1 downto 0) );
  end component;

  constant nB : positive := integer(ceil(log2(real(n*2-1)))); -- nombre de bits de sortie
  signal J : std_logic_vector(n-1 downto 0);
```

```
  signal B : std_logic_vector(nB-1 downto 0);
begin
  UUT : johnsonAbinaire generic map (n => n, nB => nB) port map (J, B);

  process
  begin
    J <= (others => '0');
    wait for 10 ns;

    for k in 0 to n*2-1 loop
      if to_integer(unsigned(B)) /= k then
        report "Erreur pour " & integer'image(k) severity error;
      end if;
      J <= J(n-2 downto 0) & not(J(n-1));
      wait for 10 ns;
    end loop;
    report "fini" severity failure;
  end process;
end arch;
```

3.
(section 7.5 des notes) Il est possible que la bascule qui échantillonne le signal tombe dans un état métastable, si le moment d'échantillon arrive trop proche d'un moment où le signal échantillonné change. Il faut utiliser par exemple un circuit à double tampon pour faire un « synchroniseur ». Il faut aussi faire attention que la fréquence d'échantillonnage soit assez grande.