

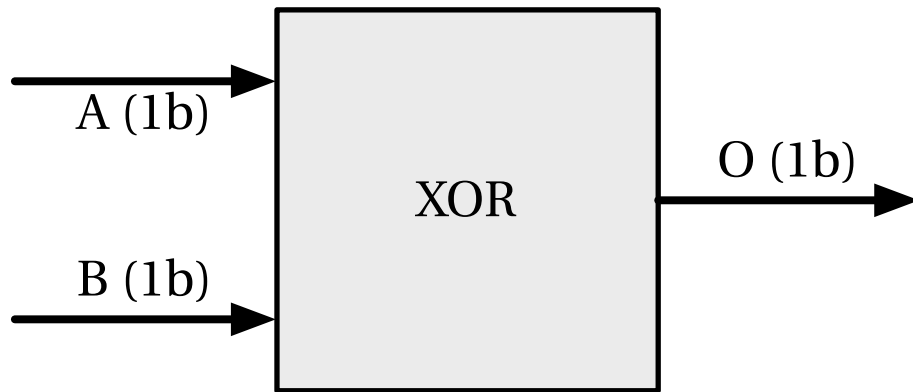
EN201 - Correction du TD n°1

Bertrand LE GAL
[bertrand.legal@ims-bordeaux.fr]

*Filière Electronique - 2^{ème} année
ENSEIRB-MATMECA - Bordeaux INP
Talence, France*



Question n°1 - Rappels du cours de VHDL



L'ordre de définition des ports n'est pas imposé mais peut avoir un impact sur le **PORT MAP** des E/S.

Le sens des ports est spécifié à l'aide des types **IN** & **OUT**

Le type **STD_LOGIC** est utilisé pour les données codées sur un bit.

Attention le dernier signal ne nécessite pas de « ; » après sa déclaration

```
ENTITY xor_v1 IS
  PORT (
    A : in  STD_LOGIC;
    B : in  STD_LOGIC;
    O : out STD_LOGIC
  );
END xor_v1;
```

Question n°1 - Description de l'entité du module

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;

ENTITY fsm IS
PORT (
    RESET      : IN  STD_LOGIC;
    CLOCK      : IN  STD_LOGIC;

    B_UP       : IN  STD_LOGIC;
    B_DOWN     : IN  STD_LOGIC;
    B_CENTER   : IN  STD_LOGIC;
    B_LEFT     : IN  STD_LOGIC;
    B_RIGHT    : IN  STD_LOGIC;

    PLAY_PAUSE : OUT STD_LOGIC;
    RESTART    : OUT STD_LOGIC;
    FORWARD    : OUT STD_LOGIC;
    VOLUME_UP  : OUT STD_LOGIC;
    VOLUME_DW  : OUT STD_LOGIC
);
END fsm;
```

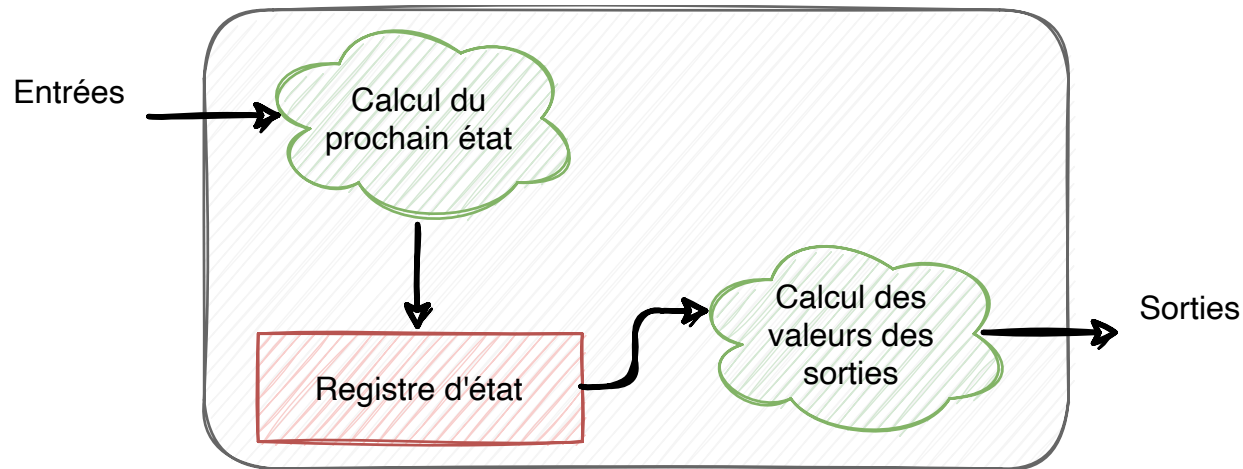


La description de l'entité VHDL est une transposition de l'exemple précédent.

Pour simplifier la relecture, les signaux d'horloge et de reset sont les premiers. Puis on déclare les entrées et enfin les sorties du module.

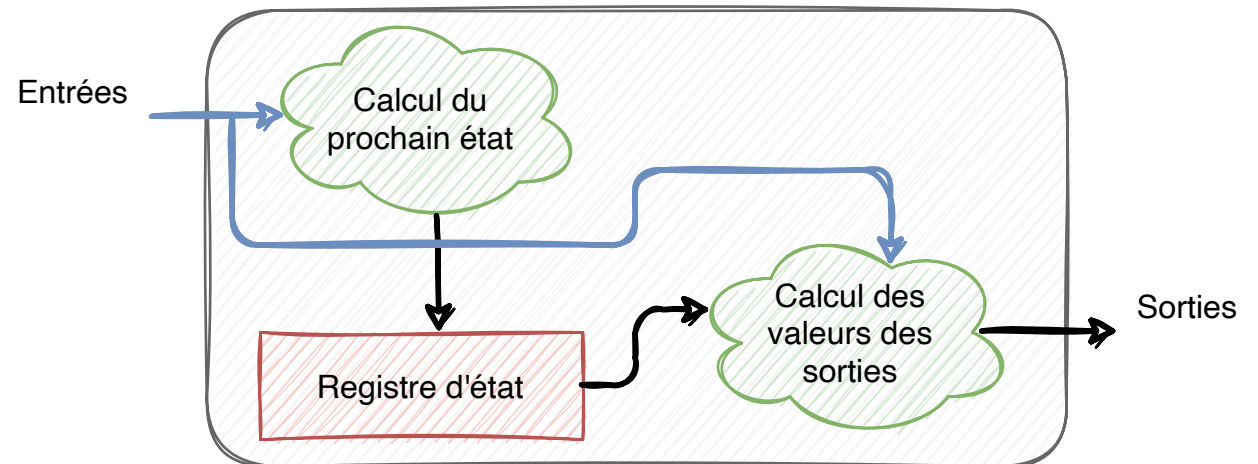
Cet ordre de dilacération n'est pas imposé tout comme les sauts de ligne... mais cela aide lors du debug !

Question n°2 - Rappels du cours concernant les FSM



Machine d'états de type Moore

Machine d'états de type Mealy



Question n°2 - Description de l'architecture

```
ARCHITECTURE Behavioral OF fsm IS
    TYPE STATE_TYPE IS (init, play_fwd, play_bwd, pause, stop);
    SIGNAL current_state : STATE_TYPE;
    SIGNAL next_state    : STATE_TYPE;
BEGIN

    -- The first process

    -- The second process

    -- The third process

END Behavioral;
```

Question n°2 - Description du processus synchrone

```
PROCESS (CLOCK)
BEGIN
  IF (CLOCK'EVENT AND CLOCK = '1') THEN
    IF RESET = '1' THEN
      current_state <= INIT;
    ELSE
      current_state <= next_state;
    END IF;
  END IF;
END PROCESS;
```

Question n°2 - Description du processus de calcul de l'état futur

```
PROCESS (current_state, B_CENTER, B_LEFT, B_RIGHT)
BEGIN
    CASE current_state IS
        WHEN INIT =>
            IF B_CENTER = '1' THEN next_state <= PLAY_FWD;
            ELSE next_state <= INIT;
            END IF;
        WHEN PLAY_FWD =>
            IF B_CENTER = '1' THEN next_state <= PAUSE;
            ELSE next_state <= PLAY_FWD;
            END IF;
        WHEN PLAY_BWD =>
            IF B_CENTER = '1' THEN next_state <= PAUSE;
            ELSE next_state <= PLAY_BWD;
            END IF;
        WHEN PAUSE =>
            IF B_LEFT = '1' THEN next_state <= PLAY_BWD;
            ELSIF B_RIGHT = '1' THEN next_state <= PLAY_FWD;
            ELSIF B_CENTER = '1' THEN next_state <= STOP;
            ELSE next_state <= PAUSE;
            END IF;
        WHEN STOP =>
            IF B_CENTER = '1' THEN next_state <= PLAY_FWD;
            ELSE next_state <= STOP;
            END IF;
        WHEN OTHERS=>
            next_state <= INIT;
    END CASE;
END PROCESS;
```

Question n°2 - Description du processus de calcul des sorties

```
PROCESS (current_state, B_UP, B_DOWN)
BEGIN
  CASE current_state IS
    WHEN INIT =>    PLAY_PAUSE<= '0';    RESTART    <= '1';
                   VOLUME_UP  <= '0';    VOLUME_DW  <= '0';
                   FORWARD    <= '0';

    WHEN PLAY_BWD => PLAY_PAUSE <= '1';    RESTART    <= '0';
                   VOLUME_UP  <= B_UP;    VOLUME_DW  <= B_DOWN;
                   FORWARD    <= '0';

    WHEN PLAY_FWD => PLAY_PAUSE <= '1';    RESTART    <= '0';
                   VOLUME_UP  <= B_UP;    VOLUME_DW  <= B_DOWN;
                   FORWARD    <= '1';

    WHEN PAUSE =>   PLAY_PAUSE <= '0';    FORWARD    <= '0';
                   VOLUME_UP  <= '0';    VOLUME_DW  <= '0';
                   RESTART    <= '0';

    WHEN STOP =>   PLAY_PAUSE<= '0';    RESTART    <= '1';
                   VOLUME_UP  <= '0';    VOLUME_DW  <= '0';
                   FORWARD    <= '0';

  END CASE;
END PROCESS;
```

Question 3 - Utilisation d'un seul processus pour le contrôle

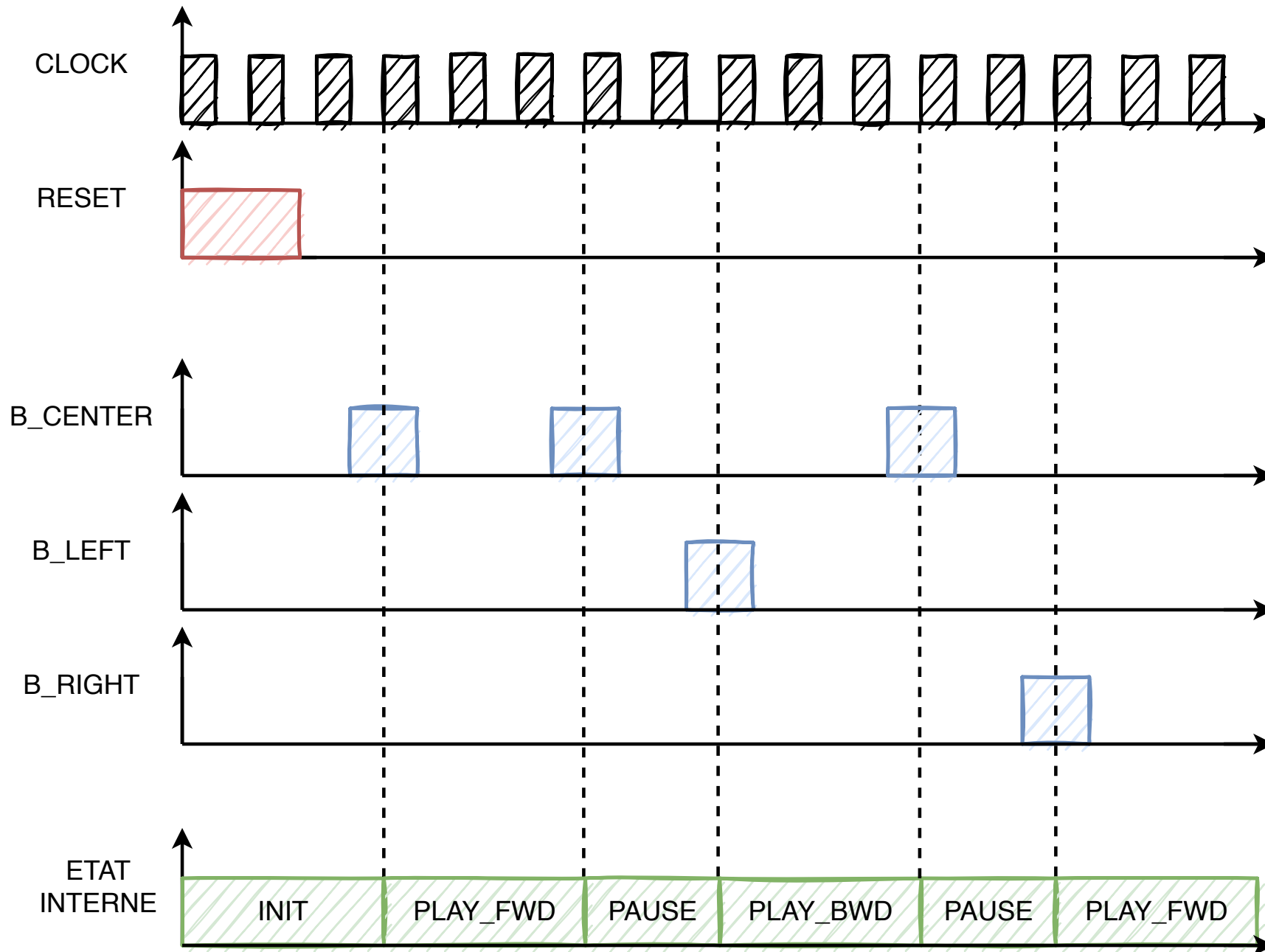
```
PROCESS (CLOCK)
BEGIN
  IF (CLOCK'EVENT AND CLOCK = '1') THEN
    IF RESET = '1' THEN
      state <= INIT;
    ELSE
      CASE state IS
        WHEN INIT =>
          IF B_CENTER = '1' THEN state <= PLAY_FWD;
          ELSE state <= INIT;
          END IF;
        WHEN PLAY_FWD =>
          IF B_CENTER = '1' THEN state <= PAUSE;
          ELSE state <= PLAY_FWD;
          END IF;
        WHEN PLAY_BWD =>
          IF B_CENTER = '1' THEN state <= PAUSE;
          ELSE state <= PLAY_BWD;
          END IF;
        WHEN PAUSE => ...
        WHEN STOP => ...
        WHEN OTHERS => state <= INIT;
      END CASE;
    END IF;
  END IF;
END PROCESS;
```

Question 3 - Calcul des sortie via des processus implicites

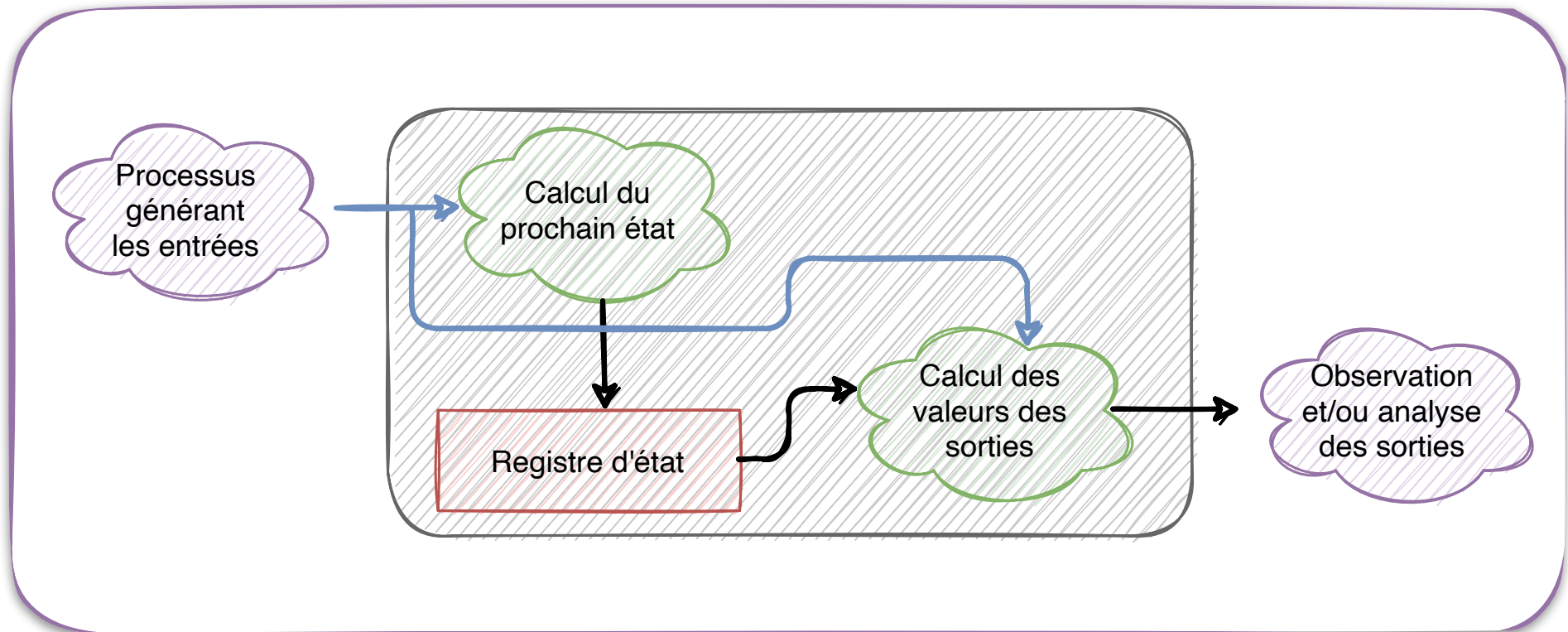
--Output signal computation

```
RESTART  <= '1'    WHEN (state = INIT)    OR (state = STOP)    ELSE  
          '0';  
  
PLAY_PAUSE <= '1'    WHEN (state = PLAY_BWD) OR (state = PLAY_FWD) ELSE  
          '0';  
  
VOLUME_UP  <= B_UP   WHEN (state = PLAY_BWD) OR (state = PLAY_FWD) ELSE  
          '0';  
  
VOLUME_DW  <= B_DOWN WHEN (state = PLAY_BWD) OR (state = PLAY_FWD) ELSE  
          '0';  
  
FORWARD    <= '1'    WHEN (state = PLAY_FWD)                                ELSE  
          '0';
```

Question 4 - Chronogramme des E/S du module

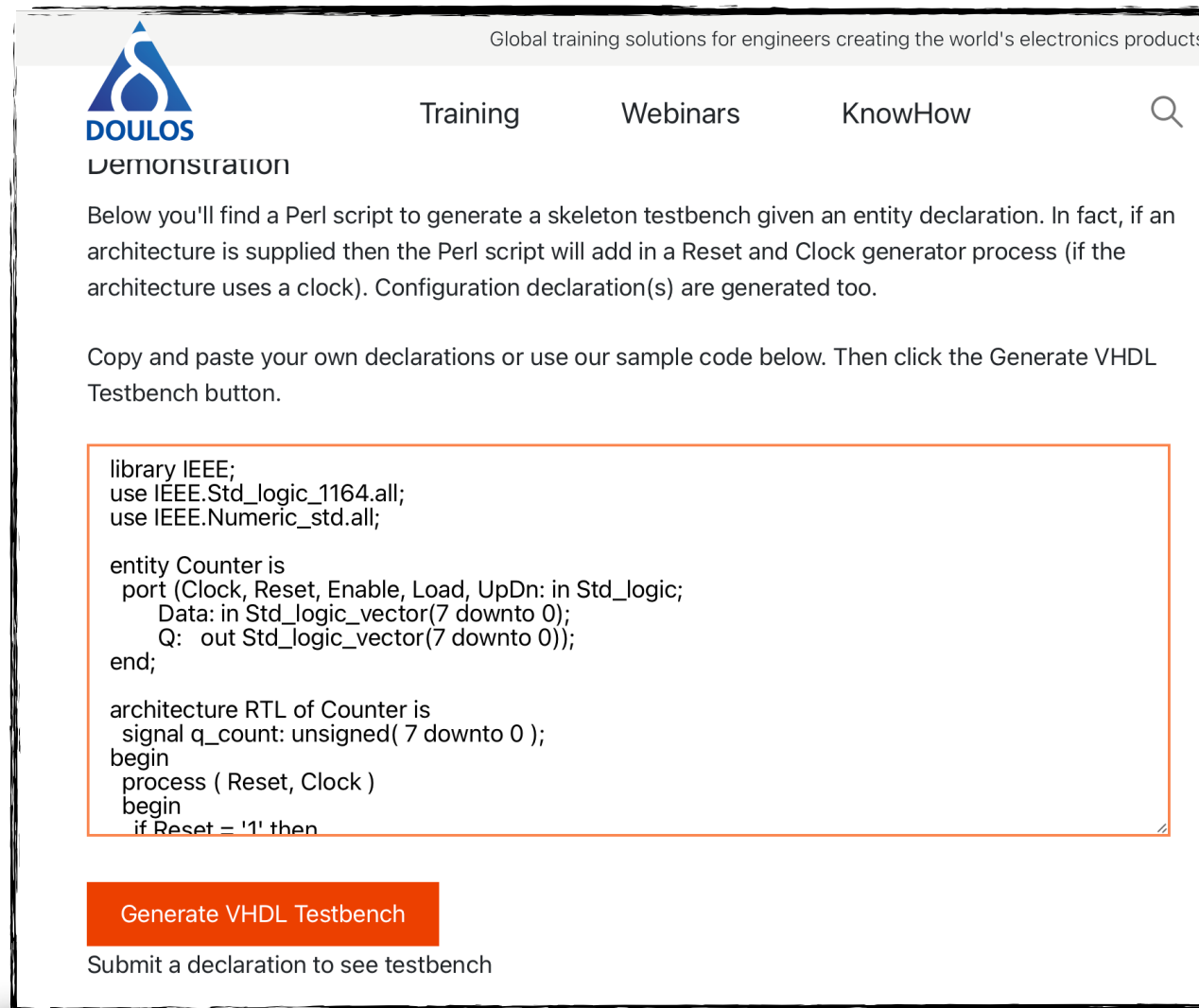


Question 5 - Génération du banc de test du module



Question 5 - Génération du banc de test du module

~~<https://www.doulos.com/httpswww.douloscomknowhow/perl/vhdl-testbench-creation-using-perl/>~~



The screenshot shows the DOULOS website interface. At the top, there is a navigation bar with the DOULOS logo, the tagline "Global training solutions for engineers creating the world's electronics products", and links for "Training", "Webinars", and "KnowHow". A search icon is also present. Below the navigation bar, the page is titled "Demonstration". The main content area contains a paragraph explaining that a Perl script is provided to generate a skeleton testbench from an entity declaration. It notes that if an architecture is supplied, the script will also generate a Reset and Clock generator process. Below this, it instructs users to copy and paste their own declarations or use the sample code provided. The sample code is a VHDL entity declaration for a counter, including port definitions and a partial architecture. At the bottom of the code block, there is an orange button labeled "Generate VHDL Testbench". Below the button, there is a text input field and a "Submit a declaration to see testbench" button.

Global training solutions for engineers creating the world's electronics products

DOULOS

Training Webinars KnowHow

Demonstration

Below you'll find a Perl script to generate a skeleton testbench given an entity declaration. In fact, if an architecture is supplied then the Perl script will add in a Reset and Clock generator process (if the architecture uses a clock). Configuration declaration(s) are generated too.

Copy and paste your own declarations or use our sample code below. Then click the Generate VHDL Testbench button.

```
library IEEE;
use IEEE.Std_logic_1164.all;
use IEEE.Numeric_std.all;

entity Counter is
  port (Clock, Reset, Enable, Load, UpDn: in Std_logic;
        Data: in Std_logic_vector(7 downto 0);
        Q: out Std_logic_vector(7 downto 0));
end;

architecture RTL of Counter is
  signal q_count: unsigned( 7 downto 0 );
begin
  process ( Reset, Clock )
  begin
    if Reset = '1' then
```

Generate VHDL Testbench

Submit a declaration to see testbench

Question 5 - Conception du testbench (Part I)

```
entity fsm_tb is
end;

architecture bench of fsm_tb is

    component fsm_v2
    PORT (
        RESET      : IN  STD_LOGIC;
        CLOCK      : IN  STD_LOGIC;
        ... ..
        B_UP       : IN  STD_LOGIC;
        VOLUME_DW  : OUT STD_LOGIC
    );
    end component;

    signal RESET: STD_LOGIC;
    signal CLOCK: STD_LOGIC;
    ... ..
    signal B_DOWN: STD_LOGIC;
    signal VOLUME_DW: STD_LOGIC ;

    constant clock_period: time := 10 ns;
    signal stop_the_clock: boolean;

begin
```

Question 5 - Conception du testbench (Part 2)

```
begin

    uut: fsm port map ( RESET      => RESET,
                        CLOCK      => CLOCK,
                        B_UP       => B_UP,
                        B_DOWN     => B_DOWN,
                        B_CENTER   => B_CENTER,
                        B_LEFT     => B_LEFT,
                        B_RIGHT    => B_RIGHT,
                        PLAY_PAUSE => PLAY_PAUSE,
                        RESTART    => RESTART,
                        FORWARD    => FORWARD,
                        VOLUME_UP  => VOLUME_UP,
                        VOLUME_DW  => VOLUME_DW );

    clocking: process
    begin
        while not stop_the_clock loop
            CLOCK <= '0', '1' after clock_period / 2;
            wait for clock_period;
        end loop;
        wait;
    end process;
```

Question 5 - Conception du testbench (Part 3)

```
stimulus: process
begin

    -- Put initialisation code here
    RESET <= '1'; wait for CLOCK_period * 2;
    RESET <= '0'; wait for CLOCK_period * 2;

    -- ON PASSE DANS L'ETAT PLAY FWD
    B_CENTER <= '1'; wait for CLOCK_period;
    B_CENTER <= '0'; wait for CLOCK_period * 9;

    -- ON PASSE DANS L'ETAT PAUSE
    B_CENTER <= '1'; wait for CLOCK_period;
    B_CENTER <= '0'; wait for CLOCK_period * 9;

    -- ON PASSE DANS L'ETAT PLAY FWD
    B_RIGHT <= '1'; wait for CLOCK_period;
    B_RIGHT <= '0'; wait for CLOCK_period * 9;

    -- ON PASSE DANS L'ETAT PAUSE
    B_CENTER <= '1'; wait for CLOCK_period;
    B_CENTER <= '0'; wait for CLOCK_period * 9;

    stop_the_clock <= true;
    wait;
end process;
```