



---

## Micro-architecture (UE S3)

---

### Dossier de Projet

SERPENTIN 7-SEGMENT PROGRAMMABLE (FPGA)

*Etudiants :*  
Afizullah RAHMANY  
Romain PEREIRA

*Enseignant :*  
M. AUGÉ

25/11/2018

# Table des matières

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Manuel utilisateur</b>	<b>3</b>
<b>3</b>	<b>Présentation générale</b>	<b>4</b>
3.1	Fonctionnement . . . . .	4
3.2	Format des messages . . . . .	5
<b>4</b>	<b>Description matérielle</b>	<b>6</b>
4.1	Schéma général . . . . .	6
4.2	Description des blocs . . . . .	7
4.2.1	Wrapper . . . . .	7
4.2.2	Dispatcher . . . . .	7
4.2.3	H100 . . . . .	8
4.2.4	H10 . . . . .	8
4.2.5	Moduleur . . . . .	8
4.2.6	Serpentin Programmable . . . . .	9
4.2.7	Serpentin clignotant . . . . .	9
4.2.8	Serpentin horaire . . . . .	9
4.2.9	Serpentin anti-horaire . . . . .	9
<b>5</b>	<b>Implémentation du bloc H100</b>	<b>10</b>

## Préambule

Ce projet a été réalisé dans le cadre de nos études à l'ENSIIE (Ecole National Supérieur d'informatique pour l'industrie et l'entreprise) d'Evry.

Ce projet est l'aboutissement de nos cours en Micro-architecture.

Nous programmions sur un FPGA d'Altera (gamme Cyclone), en VHDL et à l'aide du logiciel Quartus.

L'objectif a été de réaliser un serpentín programmable, et affichable sur un 7-segment.

# 1 Introduction

Vous cherchez un cadeau de Noël pour vos enfants ou vos grands parents ?

Ne cherchez plus, voici **le Serpentin 3000**

Cette modeste plaquette électronique passionnera les petits comme les grands enfants.

**Amusez vous !**

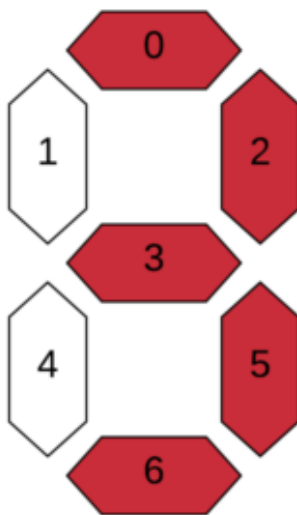
Modifier la vitesse et le type de défilement.

**Exprimez votre créativité !**

Grâce à l'interface 'Programmation 3000', vous pouvez créer votre propre serpent.

**Accessible** Pour la modique somme de 4.99€, offrez vous un

## SERPENTIN 3000



## 2 Manuel utilisateur

**Bouton de droite** 'Bouton reset' , réinitialise l'état du serpent dans son état initial

**Interrupteurs** 'IVDF2 IVDF1 IVDF0' permettent de régler la vitesse de défilement (en nombre de ticks)

IVDF2	IVDF1	IVDF0	vitesse de défilement
0	0	0	2
0	0	1	5
0	1	0	7
0	1	1	10
1	0	0	12
1	0	1	15
1	1	0	17
1	1	1	20

**Diodes d'interrupteurs** 'DVDF2 DVDF1 DVDF0' allumées ou éteintes selon que les interrupteurs IVDF2, IVDF1 ou IVDF0 sont actifs ou inactifs.

**Diode défilement** 'DVDF' cette diode clignote à la vitesse de défilement

**Sélecteur serpent** 'SEL1 SEL2' permettent de sélectionner le type de serpent

SEL1	SEL0	type de serpent
0	0	clignotement
0	1	horaire
1	0	anti-horaire
1	1	programmable

**BUS - IA** BUS-IA, prend un message pour communiquer avec le serpent [3.2](#)

### 3 Présentation générale

#### 3.1 Fonctionnement

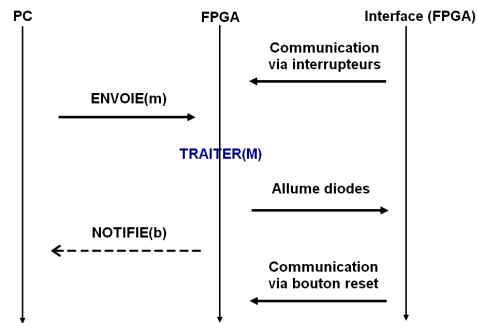
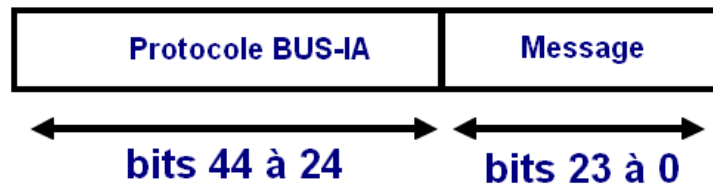


FIGURE 1 – MSC des actions utilisateur

*m* est un *message*. Il s'agit d'une suite de **44 bits** avec le format :

- **bits 43 à 24** Ce sont les bits du contrôle du BUS-IA (voir Bus-IA)
- **bits 23 à 0** Ce sont les bits du message



Un *message* doit avoir le format :

- **bits 23 à 21** Ce sont les bits codant le *type* du *message*
- **bits 20 à 0** Ce sont les données du message, le format varie selon le *type*



## 3.2 Format des messages

Ci-dessous, voici les commandes supportées par le processeur, et le format des messages permettant d'exécuter ces commandes.

**noop** Ne rien faire

- **bits 23 à 21** : "000" (= (bit 23, bit 22, bit 21))

- **bits 20 à 0** : *non utilisées*

**h-init(n)** Génère un tick sur H100 tous les n coups d'horloge maître

- **bits 23 à 21** : "001"

- **bits 20 à 0** : *non utilisées*

**h-check-ON()** Demande au processeur d'envoyer au PC un message TICK1000 tous les 1000 tickets de H100.

- **bits 23 à 21** : "010"

- **bits 20 à 0** : *non utilisées*

**h-check-OFF()** Demande au processeur d'arrêter d'envoyer au PC un message TICK1000

- **bits 23 à 21** : "011"

- **bits 20 à 0** : *non utilisées*

**clr()** Indique au processeur d'afficher un serpent vide sur le 7-segment (**n** à 32 et met à 0 toutes les *valeurs* programmées)

- **bits 23 à 21** : "100"

- **bits 20 à 0** : *non utilisées*

**set-N(n)** Indique au processeur le nombre de **valeurs** à faire défiler en boucle sur le 7-segments. (**n** compris entre 1 et 32)

- **bits 23 à 21** : "101"

- **bits 20 à 6** : *non utilisées*

- **bits 5 à 0** : **n** codé sur 6 bits ( $2^6 = 64$ )

**set-val(i, v)** Indique au processeur la i-ème valeur du 7-segments est **v**

- **bits 23 à 21** : "110"

- **bits 20 à 13** : *non utilisées*

- **bits 12 à 6** : **v** les 7 bits correspondent aux 7 segments de l'afficheur (0 => éteint, 1 => allumé)

- **bits 5 à 0** : **n** codé sur 6 bits ( $2^6 = 64$ )

## 4 Description matérielle

### 4.1 Schéma général

Quelques simplifications ont eu lieu pour la lisibilité du schéma :

- En réalité, chaque bloc prend la master clock et le bouton reset en entrée
- Les 224 bits de configuration et les 6 bits codant N ont été groupé en un seul fil (de 230 bits)
- S6 S5 S4 S3 S2 S1 S0 sont combinés en un seul fil

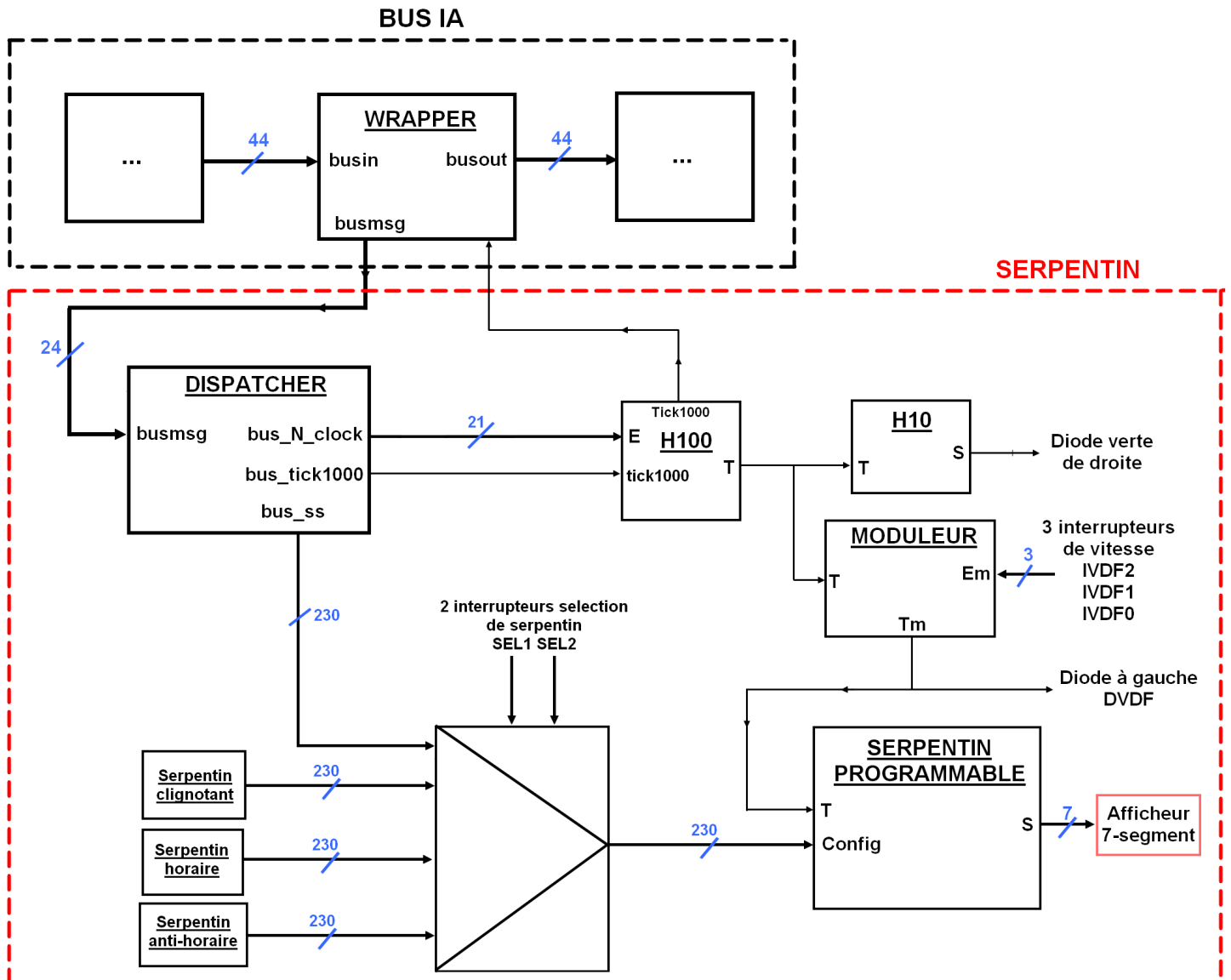


FIGURE 2 – Schéma général simplifié du matériel

## 4.2 Description des blocs

### 4.2.1 Wrapper

Il s'agit de l'interface de communication dans le BUS-IA.

Ce bloc suit le protocole 'handshake' dans le BUS-IA.

Il récupère les messages qui lui sont destinés, puis les renvoie vers le processeur du serpent.

#### Entrées

- clk, reset : la master clock et le reset
- busin : entrée dans le BUS-IA (44 bits, protocole handshake)
- busin\_valid : entrée dans le BUS-IA (1 bit, protocole handshake)
- busout\_eated : entrée dans le BUS-IA (1 bit, protocole handshake)

#### Sorties

- busout : sortie vers le BUS-IA (44 bits, protocole handshake)
- busout\_valid : sortie vers le BUS-IA (1 bit, protocole handshake)
- busin\_eated : sortie vers le BUS-IA (1 bit, protocole handshake)
- busmsg : sortie vers le processeur du serpent (24 bits, pas de protocole)

### 4.2.2 Dispatcher

Il s'agit du bloc de répartition.

Il récupère les *messages* qui lui sont destinés, les interprète, et modifie ses sorties en fonction.

#### Entrées

- clk, reset : la master clock et le reset
- busmsg : le message à interpréter (24 bits)

#### Sorties

- bus\_ss : sortie vers le 7-segment ( $32 * 7 = 224$  bits), codant les configurations possibles du 7-segments.
- bus\_n : sortie vers le 7-segment (6 bit), codant 'N' ([3.2](#))
- bus\_n\_clock : sortie vers le générateur de ticks (20 bits), codant le nombre de **kilo master clock** à attendre pour générer un tick.

La master clock est à 50MHz, pour une entrée

$$(1001110001000)_2 kHz = (5000)_{10} kHz = (5000000) Hz$$

On aura 100 ticks seront générés par secondes.

- bus\_tick1000 : sortie vers le générateur de tick, '1' ou '0' selon que le générateur envoie ou non un message au PC tous les 1000 ticks.



### 4.2.3 H100

Il s'agit du générateur de tick.

Il récupère un nombre  $N\_clock$ , puis compte  $N\_clock$  **kilo master clock** avant de générer un tick.

Lorsqu'un tick est généré, sa sortie passe à '1' pendant un master clock, sinon elle vaut '0'

#### Entrées

- clk, reset : la master clock et le reset
- N\_clock : le nombre de **kilo master clock** (20 bits)

#### Sorties

- T : '1' ou '0' si un tick est généré ou non (1 bit)

### 4.2.4 H10

Il s'agit d'un compteur de tick.

Il module le signal du tick en un signal de clignotement.

Sa sortie alterne entre '1' et '0' tous les 10 ticks.

Une diode est branché sur ce bloc, elle clignote avec une fréquence de 10 ticks.

#### Entrées

- clk, reset : la master clock et le reset
- T : un tick (1 bit)

#### Sorties

- S : le signal de sortie (1 bit)

### 4.2.5 Moduleur

Il s'agit d'un moduleur de tick programmable.

Il prend un entier  $X$  codé sur 3 bits en entrée (avec donc 7 valeurs possibles)

Il génère un tick tous les  $Y$  ticks de H100, avec la correspondance :

X	"000"	"001"	"010"	"011"	"100"	"101"	"110"	"111"
Y	2	5	7	10	12	15	17	20

Ce tick est envoyé dans l'entrée du serpentín programmable.

#### Entrées

- clk, reset : la master clock et le reset
- E2 E1 E0 : l'entier codé sur 3 bits (3 x 1 bit)
- T : un tick (1 bit)

#### Sorties

- S : le tick de sortie (1 bit)

#### 4.2.6 Serpentin Programmable

Il s'agit du serpentín programmable.

On appelle *frame* 7 bits configurant l'état des segments sur le 7-segment.

Si le i-ème bit est à '1', cela indique que le i-ème segment doit être allumé, sinon il est éteint.

Ce bloc prend en entrée un tableau de 32 *frames*.

Il sélectionne successivement (modulo N) les frames 1 à N, lorsqu'un tick est reçu.

##### Entrées

- clk, reset : la master clock et le reset
- T : un tick venant du moduleur (1 bit)
- frames : les images programmées sur le 7-segment ( $7 * 32 = 224$  bits)
- N : le nombre de frames sur lesquelles le 7-segment doit boucler (entre 1 et 32, 6 bits)

##### Sorties

- S6 S5 S4 S4 S3 S2 S1 S0 : les 7 bits codant la frame en cours.

#### 4.2.7 Serpentin clignotant

Il s'agit du serpentín pré-programmé.

Il sort la configuration du serpentín correspondant à un clignotement.

##### Sorties

- N : "000010" (2, le serpentín boucle sur 2 frames)
- frames : "000...000 1111111" (toutes les frames éteintes, et une allumée)

#### 4.2.8 Serpentin horaire

Il s'agit du serpentín pré-programmé.

Il sort la configuration du serpentín correspond à une rotation horaire.

##### Sorties

- N : "000110" (6, le serpentín boucle sur 6 frames)
- frames : la configuration correspondante

#### 4.2.9 Serpentin anti-horaire

Il s'agit du serpentín pré-programmé.

Il sort la configuration du serpentín correspond à une rotation anti-horaire.

##### Sorties

- N : "000110" (6, le serpentín boucle sur 6 frames)
- frames : la configuration correspondante

## 5 Implémentation du bloc H100

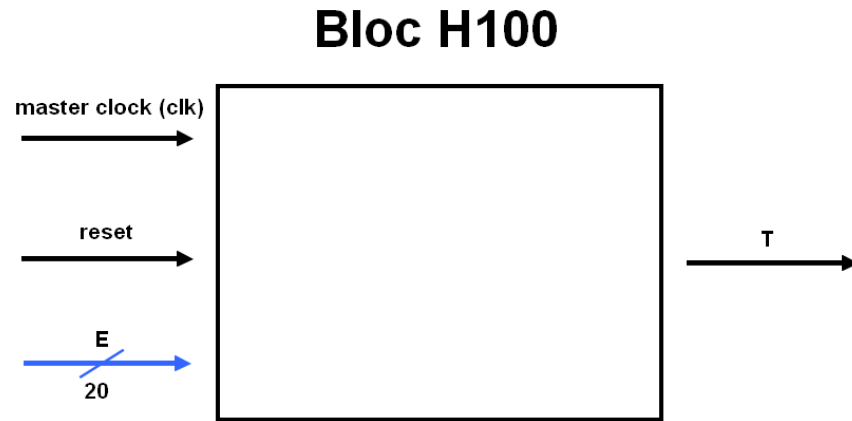


FIGURE 3 –

Etat	numéro de lignes	sortie T
LOAD	5 à 6	0
DECR	7 à 10	0
TICK	11	1

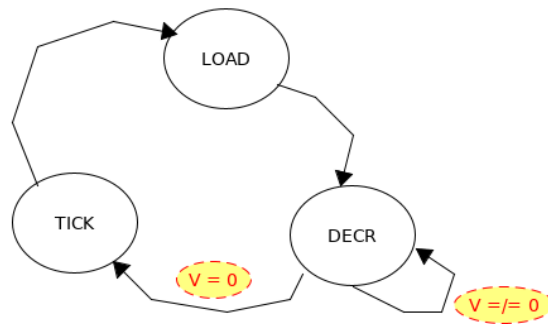


FIGURE 4 –

---

Algorithm 1 – Algorithme qui tourne en boucle sur le bloc

---

```
1: Entrée  $E$  entier sur 20 bits
2: Sortie  $T$  un tick codé sur 1 bit ('0' ou '1')
3: Registre  $R_c$  compteur sur 23 bits
4: function UPDATE
5:    $T = 0$ 
6:    $R_c := E$ 
7:   Tant que  $R_c \neq 0$ 
8:      $R_c := R_c - 1$ 
9:     Attendre 1 master clock
10:  Fin tant que
11:   $T = 1$ 
12:  Attendre 1 master clock
13: end function
```

---

Implémentation VHDL

```
-----
-- Entrée:
--           clk, reset la clock et le reset
--           Entier E codé sur 21 bits (nombre de kilo master cycle)
--
-- Sortie:
--           T , le tick : '1' tous les 'E' master cycle, sinon '0'
-----
```

```
library IEEE;
use IEEE.std_logic_1164.all;
use ieee.numeric_std.all;

entity h100 is
  port(
    clk : in STD_LOGIC;
    reset : in STD_LOGIC;
    E : in STD_LOGIC_VECTOR(20 downto 0);
    T : out STD_LOGIC
  );
end h100;

architecture montage of h100 is
```

```
-----
-- Partie Opérative
-----
  type T_CMD is (LOAD, DECR, NOOP);
-- la commande courante
  signal CMD : T_CMD;
-- le registre de stockage, compteur des clocks
  signal R : unsigned (23 downto 0);
  -- boolean vrai si R est 0
```

```

    signal R_IS_NULL: STD_LOGIC;

-----
-- Partie Contr le
-----

    type STATE_TYPE is (
        ST_LOAD, ST_DECR, ST_TICK
    );
    signal state : STATE_TYPE;

begin

-----
-- Partie Op rative
-----

    process (clk)
    begin if clk'event and clk = '1' then
        IF CMD = LOAD THEN
            -- charges 'E' dans 'R' en kilo master cycle
            R(23 downto 3) <= unsigned(E);
            R(2 downto 0) <= to_unsigned(0, 3);
        ELSIF CMD = DECR THEN
            R <= R - 1;
        END IF;
    end if; end process;

    R_IS_NULL <= '1' WHEN R = 0 ELSE '0' ;

-----
-- Partie Contr le
-----

-- Inputs: R_IS_NULL, state
-- Outputs: T, CMD, state
-----

-- fonction de transition
    process (reset, clk)
    begin
        if reset = '1' then
            state <= ST_LOAD;
        elsif clk'event and clk = '1' then
            case state is
                when ST_LOAD =>
                    state <= ST_DECR ;
                when ST_DECR =>
                    IF R_IS_NULL = '1' THEN
                        state <= ST_TICK;
                    END IF ;
                when ST_TICK =>
                    state <= ST_LOAD ;
            end case;
        end if;
    end process;

```

```

                                end case;
                        end if;
    end process;

-- fonction de sortie
    with state select T <=
        '1'    when    ST_TICK,
        '0'    when    others;

    with state select CMD <=
        LOAD    when    ST_LOAD,
        DECR    when    ST_DECR,
        NOOP    when    ST_TICK;

end montage;

```