

# 1 Introduction

La complexité de la conception des systèmes embarqués modernes étant devenue trop élevée, il est indispensable de recourir à des outils afin d'automatiser ce processus fastidieux. Dans le cours A3, on voit que les étapes de cette conception consistent en la préparation d'un système ayant un CPU virtuel dit *SoftCore*, une ou plusieurs mémoire pour accompagner ce CPU et une brique FPGA qui agit comme un accélérateur matériel. Durant les séances de TP, on procède à la conception de plusieurs variante cette brique afin d'effectuer le calcul d'une racine carrée entière sur FPGA décrite en VHDL. Puis, on instancie un système embarqué grâce à l'outil Qsys d'Intel/Alter pour avoir un CPU et les périphériques requis. Enfin, on intégrera la brique conçue dans ce système afin de mesurer ses performances globales.

## 2 Conception d'opérateur racine carrée

On commence par une analyse de l'algorithme afin de comprendre sa complexité et l'utilité d'accélérer un tel calcul matériellement.

---

**Input:** (X,n) : X entier codé sur  $2 \times n$  bits  
**Output:** Z : Z entier codé sur n bits

```
1 Charger X
2  $V = 2^{2n-2}$ 
3  $Z = 0$ 
4 for  $i = n-1$  à 0 do
5    $Z = Z + V$ 
6   if  $X - Z \geq 0$  then
7      $X = X - Z$ 
8      $Z = Z + V$ 
9   else
10     $Z = Z - V$ 
11     $Z = Z/2$ 
12     $V = V/4$ 
13 retourner Z
```

---

### 2.1 Implémentation combinatoire

Dans un premier temps, on pense à simplement traduire **tout** l'algorithme en vhd dans un seul *process* afin d'avoir une séquence d'éléments combinatoires propageant le résultat pour chaque valeur de i. Ceci sera synthétisé comme un circuit volumineux très lent.

#### Résultats

#### Code

```
1 library IEEE;
2 use IEEE.std_logic_1164.all;
3 use IEEE.numeric_std.all;
4
5 entity SQRT_one_process is
6   generic(
```

```

7      N                : natural    := 32;
8      HANDSHAKE_ENABLE : boolean    := false ;
9      RST_ON           : STD_LOGIC := '0' --active low
10 );
11 port (
12     X      : in  STD_LOGIC_VECTOR(2*N-1 downto 0);
13     start  : in  STD_LOGIC;
14     r_out  : out STD_LOGIC_VECTOR(N-1 downto 0);
15     done   : out STD_LOGIC;
16     clk    : in  std_logic;
17     rst    : in  std_logic
18 );
19 end entity;
20
21 architecture rtl of SQRT_one_process is
22     constant zero : UNSIGNED(2*N-1 downto 0) := (others => '0');
23     constant V_INI : UNSIGNED(2*N-1 downto 0) := (2*N-2 => '1',others => '0');
24     signal R : UNSIGNED(2*N-1 downto 0) := (others => '0');
25 begin
26     process(clk,rst)
27         variable V      : UNSIGNED(2*N-1 downto 0);
28         variable temp_Z : UNSIGNED(2*N-1 downto 0);
29         variable temp_X : UNSIGNED(2*N-1 downto 0);
30     begin
31         if rst= RST_ON then
32             r_out <= (others => 'Z');
33             done  <= '0';
34         else
35             if rising_edge(CLK) then
36                 if start='1' then
37                     temp_X := UNSIGNED(X);
38                     temp_Z := UNSIGNED(zero);
39                     V      := UNSIGNED(V_INI);
40                     T <= R; -- force quartus to analyse
41                     for I IN N downto 0 loop
42                         temp_Z := temp_Z + v;
43                         if(SIGNED(temp_X-temp_Z)>=0) then
44                             temp_X := temp_X-temp_Z;
45                             temp_Z := temp_Z+V;
46                         else
47                             temp_Z := temp_Z-V;
48                         end if;
49                         temp_Z := SHIFT_RIGHT(temp_Z,1);
50                         V      := SHIFT_RIGHT(V,2);
51                     end loop;
52                     r_out <= STD_LOGIC_VECTOR(temp_Z(N-1 downto 0));
53                     done  <= '1';
54
55                     T <= R; -- force quartus to analyse
56

```

```

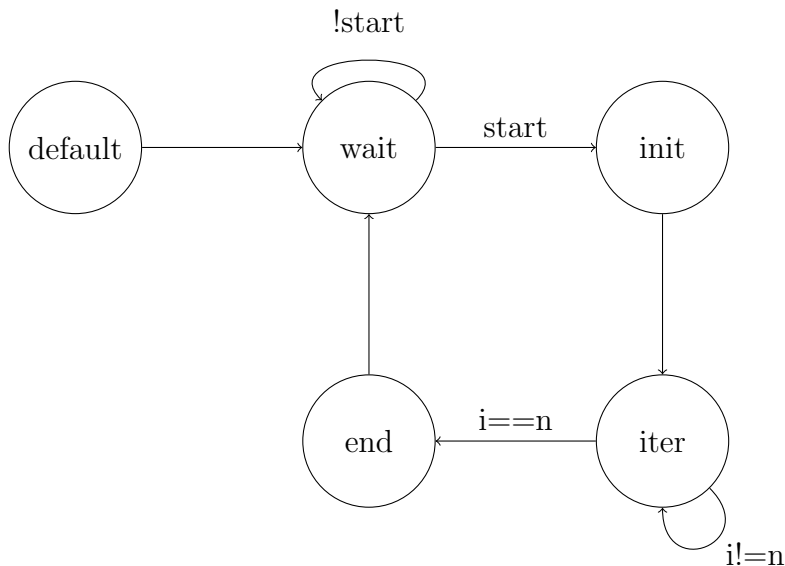
57         else
58             done <= '0';
59             T <= R; -- force quartus to analyse
60         end if;
61     end if;
62 end if;
63 end process;
64 end architecture;

```

## 2.2 Implémentation multi-cycles

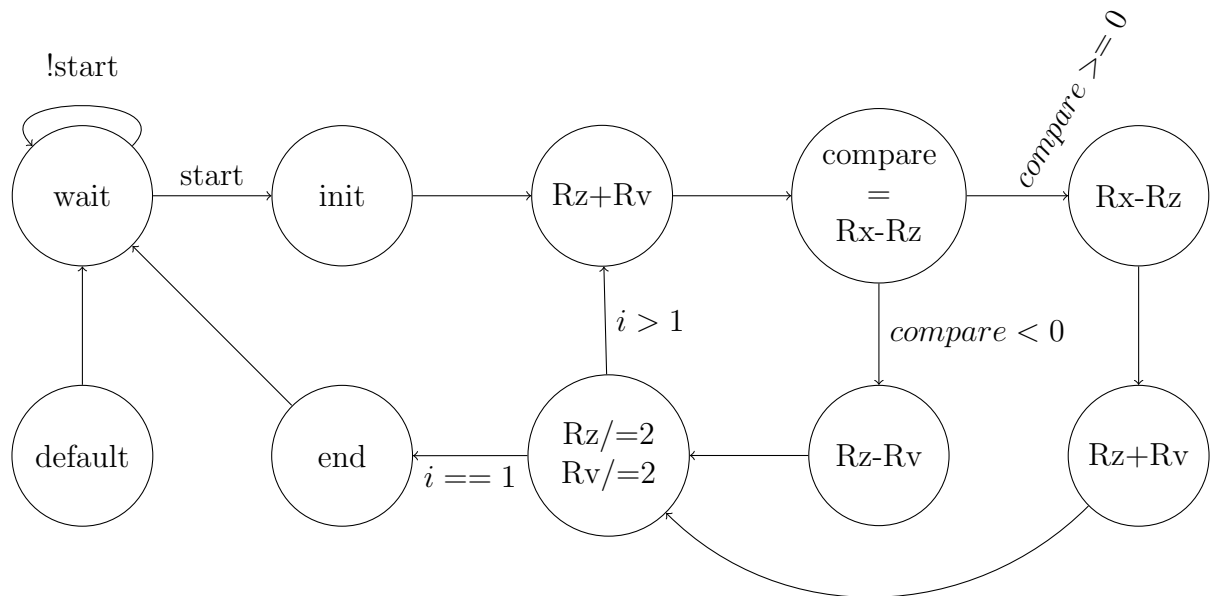
### 2.2.1 4 cycles

On choisit alors d'implémenter une machine synchrone où on décrit le module de racine carrée par une machine d'états finis. il y aura un cycle où on itère  $n$  fois . Le diagramme ci-dessous représente cette automate :



### 2.2.2 9 cycles

Cette implémentation décompose relativement bien les états de l'automate. Mais, on pense pouvoir obtenir un gain de performances en ayant un cycle par opération à effectuer. L'idée est que vu que l'état sera combinatoirement simple, il sera plus rapide à exécuter et donc la fréquence augmente. Par contre, on pense aussi que ceci consommera plus de ressources du FPGA.



### 2.2.3 Variante avare

## 2.3 Implémentation avec opérateur unique

### 2.3.1 Opérandes préemptées

### 2.3.2 Ségrégation des machines

## 2.4 Implémentation Pipeline

# 3 Comparaison et résultat

# 4 Conception du système embarqué

# 5 Programmation de la partie Nios

# 6 Intégration etc