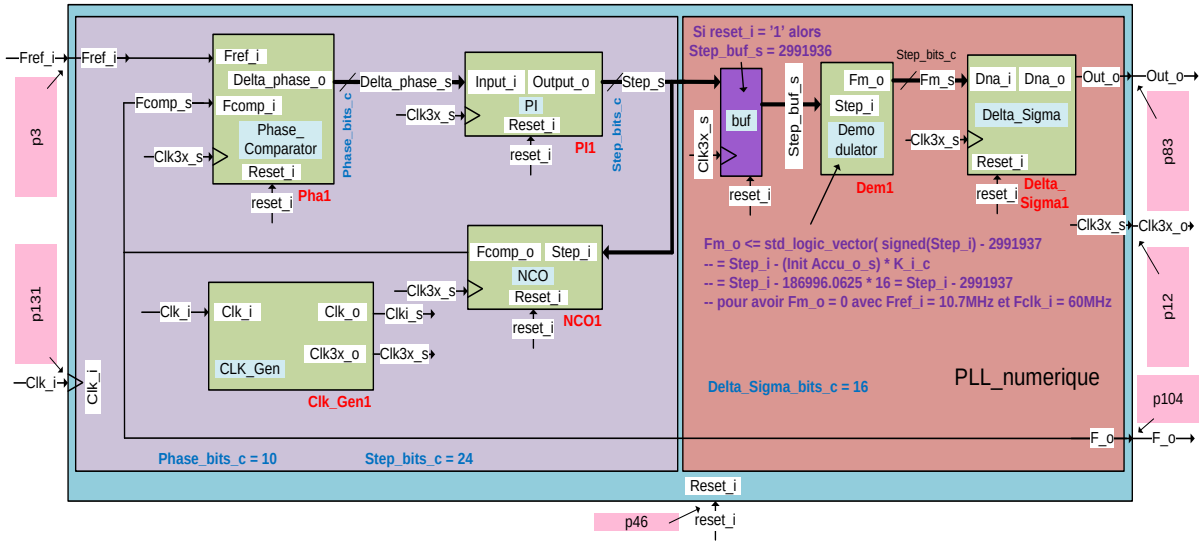
TP6-7-8: PLL numérique

1. BUTS DU TRAVAIL

- ✓ **Etude** des éléments constituant une PLL numérique suivie d'un convertisseur D/A Sigma-Delta.
- ✓ **Modélisation VHDL** du circuit complet (seulement la PLL numérique, c.à.d. la partie bleue du schéma ci-dessous)
- ✓ **Modélisation du testbench** non synthétisable selon les instructions du cours.
- ✓ **Simulation** du circuit à l'aide de Modelsim et du testbench.
- ✓ **Synthèse** du circuit sur la FPGA Spartan3A (50k gates et 144 broches ; XC3S50A TQ144) de Xilinx avec ISE
- ✓ **Vérification** acoustique (démodulation FM) du circuit téléchargé sur la carte **R-FM (Recepteur FM)** de l'EIA-FR.

2 DESCRIPTION DU CIRCUIT

Le schéma bloc du circuit à réaliser a été vu au cours. Il est représenté ci-dessous :



Dans ce travail pratique, les étudiants doivent modéliser tous les blocs, sauf le tripleur de fréquence «clk_gen». Cette partie en rouge et le clk_gen est donnée par le professeur. Le détail du contenu de chaque bloc est décrit dans le support de cours.

3 TRAVAIL À EFFECTUER

A. **Créer un dossier PLL_numérique** avec tous les sous répertoires usuels ainsi que la librairie de travail work (ce travail a déjà été fait au cours). Attention, dans le fichier **modelsim.ini**, changer la **résolution en ps**.

B. **Modéliser** en **VHDL** (fichier source avec une librairie personnelle pour la définition des constantes) le comparateur de phase « Phase Comparator ». Placer ces blocs dans le fichier PLL_numerique.vhdl fourni par le professeur. Ce fichier se trouve sous.

C. Dans ce fichier PLL_numerique.vhdl (Template téléchargeable depuis Moodle), **modéliser** en **VHDL** le régulateur **PI** et le **NCO** (Numerically-Controlled Oscillator) qui peut être ici considéré comme un diviseur de fréquence de l'horloge système.

   Pour l'horloge système à 60 MHz de cette PLL nous avons besoin d'un multiplicateur de fréquence par 3.

```
entity Clk_3x is
   port ( CLKIN_IN         : in    std_logic;
          RST_IN           : in    std_logic;
          CLKFX_OUT        : out   std_logic;
          CLKFX180_OUT     : out   std_logic;
          LOCKED_OUT       : out   std_logic);
end Clk_3x;
```

   Comme ce genre de multiplicateur est décrit dans le cours VHDL et fait intervenir des notions plus avancées, il est déjà modélisé dans le fichier PLL_numerique.vhd fourni par le professeur.

D. Modèliser en VHDL le testbench selon les indications données au cours (pages 16 et 17). Ces informations sont également décrites dans l'annexe de cette donnée.

E. Simuler votre modèle VHDL à l'aide de votre testbench. Utiliser pour cela les fichiers Phase_comparator_cours_ps.do, PLL_numerique_tb.do et PLL_numerique_tb.jpg donnés par le professeur.

   Remarque: Les fichiers fournis par le professeur se trouvent sous Moodle

F. Synthétiser votre circuit et utiliser pour cela le fichier PLL_numerique.ucf donné par le professeur.

G. Télécharger et tester votre circuit sur la carte RADIO FM du labo. Ce test audible se fera par démodulation d'un émetteur radio FM.

H. Résumez vos résultats (code VHDL, simulations et mesures) dans un rapport de laboratoire.

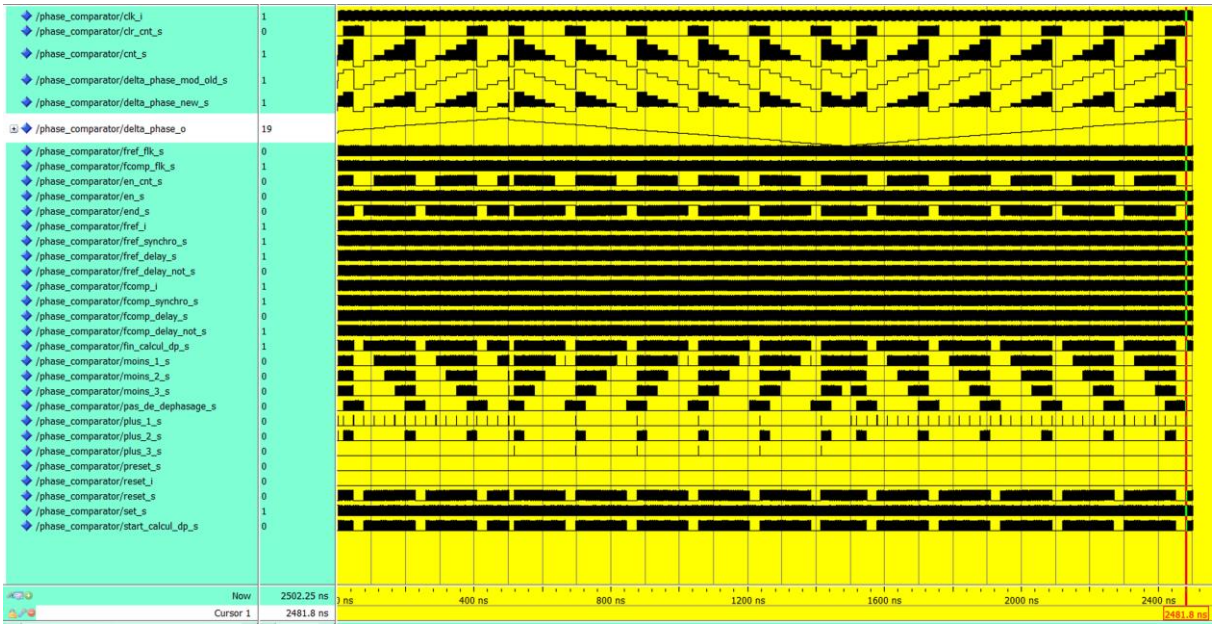
Annexe : Modèle VHDL du test bench

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

entity PLL_Numerique_tb is end PLL_Numerique_tb;

architecture comp of PLL_Numerique_tb is

COMPONENT PLL_Numerique IS
PORT(clk_i,reset_i, fref_i : IN std_logic; F_o, out_o, clk3x_o : OUT
std_logic);
END COMPONENT;

signal clk_sti, clk_3x_sti, fref_sti : std_logic;
signal reset_sti : std_logic;
signal F_obs, out_obs : std_logic;
signal out_dec_obs : integer;
signal omega_s : real;

-- to be customized by user
constant f_clk_sti_c : real := 20000000.0; -- Fréquence de l'horloge
système d'entrée 20 MHz (60MHz/3)
constant fe_porteuse_c : real := 10700000.0;        -- Fréquence d'entrée
centrale
constant deltaf_c : real := 150000.0; -- Modulation de fréquence plus/moins
150 kHz
constant fe_audio_c : real := 12500.0; -- Fréquence audio 12.5 kHz
constant ampl_sinus : real :=0.97;      -- Amplitude du sinus Audio
(réduction par rapport à deltaf_c
-- enf of customization by user

constant clockperiod_c : time := (1.0/f_clk_sti_c) * 1 sec;

shared variable init_temps_v : time;
shared variable t_v : real;

signal inputperiod_s : time := (1.0/fe_porteuse_c) * 1 sec;
signal fe_input_sti : real ;

for all: PLL_Numerique use entity work.PLL_Numerique;


begin

PLL_Numerique1 : PLL_Numerique
port map (clk_i => clk_sti, reset_i => reset_sti, fref_i => fref_sti, F_o
=> F_obs,
out_o => out_obs, clk3x_o => clk_3x_sti);

process
   begin
   reset_sti<='1';
   init_temps_v := now;
   wait for 100 ps;
   reset_sti <= '0';
   wait;
```

```
end process;

process (clk_sti, reset_sti)
begin
if reset_sti='1' then
    fe_input_sti <= fe_porteuse_c;
    omega_s <= 0.0;
elsif rising_edge(clk_sti) then
    t_v := real((now - init_temps_v) / 1 ps) / 1000000000000.0;
    omega_s <= (2.0*math_pi*fe_audio_c*t_v);
    fe_input_sti <= fe_porteuse_c +
(ampl_sinus*real(deltaf_c)*sin(omega_s));
end if;
end process;

inputperiod_s <= (1.0/fe_input_sti) * 1 sec;
fref_sti <= '0' when reset_sti='1' else not fref_sti after inputperiod_s/2;

clk_sti <= '0' when reset_sti='1' else not clk_sti after clockperiod_c/2;

process (clk_3x_sti, reset_sti)
begin
if reset_sti='1' then
    out_dec_obs <= 0;
elsif rising_edge(clk_3x_sti) then
  if out_obs = '1' then out_dec_obs <= out_dec_obs + 1;
  else out_dec_obs <= out_dec_obs - 1;
  end if;
end if;
end process;

end comp;
```

Annexes: simulation

Phase comparator:



PLL numérique :