COMPTE-RENDU SYSTÈMES SUR PUCES

« GCD » - HLS SYNTHESIS

Réalisé par Sean Marotta & Pierre-Élie Morrot

Enseignant : Julien Dubois & Barthélémy Heyrman
Année : 2020 – 2021

➢ INTRODUCTION :

Dans ce TP, nous allons apprendre à créer notre propre IP depuis Vivado HLS à partir d'un code C (avec les Test Bench associé bien sûr), et une fois cette IP créée, nous allons l'intégrer dans un Block Design sur Vivado normal. Cet IP que nous allons créer permettra de calculer le GCD entre deux entiers.

Pour commencer, dans Vivado HLS (2019.2), il faut créer trois fichiers :

- *gcd.c* : le code correspondant calcule tout simplement le GCD.

```
int gcd(int x, int y)
{
    while(x != y){
        if(x > y){
            x = x - y;
        }else{
            y = y - x;
        }
    }
    return x;
}
```

- *gcd_tb.c* : ici nous testons la fonction ci-dessus avec les valeurs 9 et 6.

```
#include<stdio.h>
#include "gcd.h"

int main(){
    printf("GCD is %d",gcd(9,6));
    return 0;
}
```

- *gcd.h* : dans ce fichier, nous mettons les fonctions du *gcd.c*.

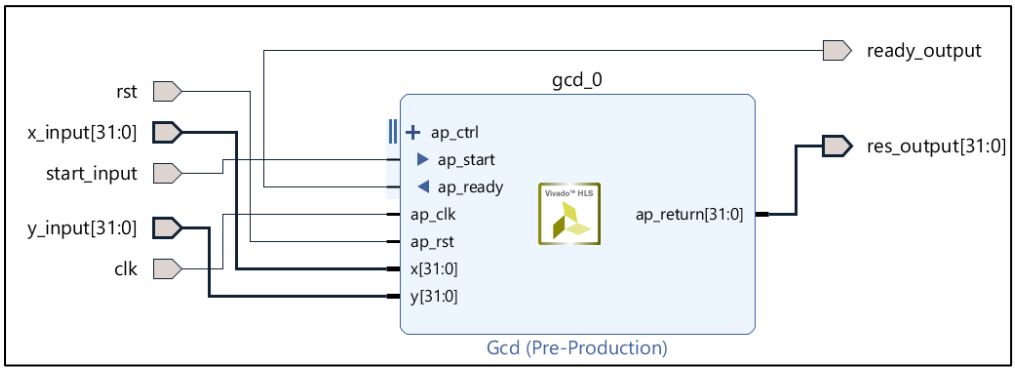
gcd.h

```
int gcd(int a, int b);
```

Une fois terminé, il faut **Run C Simulation**, **Run C/RTL Cosimulation** et **Export RTL**. Avant de passer sur Vivado 2019.2, on peut **Open Wave Viewer** pour voir le bon fonctionnement sous forme de signaux.

Dans Vivado 2019.2, on ajoute l'IP généré précédemment dans un Block Design (Create Block Design → Add IP → ajouter notre composant Gcd) mais tout d'abord il ne faut pas oublier de l'ajouter dans l'IP Catalog (Click droit → Add Repository → Retrouver notre composant).

Nous obtenons donc ce schéma dans le Block Design après avoir ajouté tous les ports nécessaires :

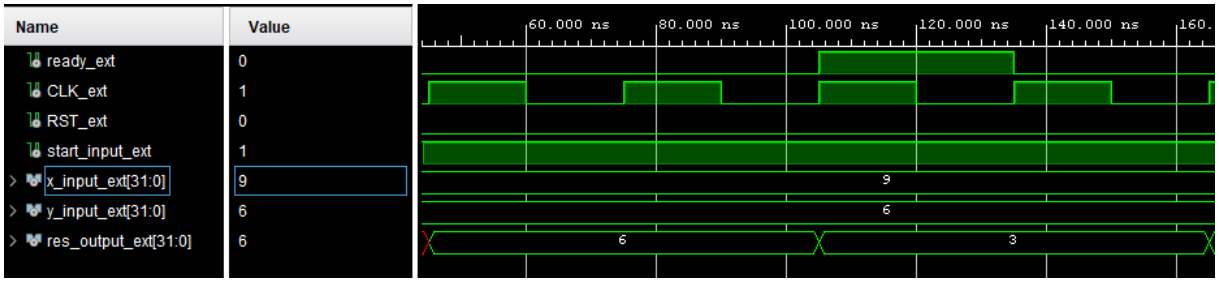


Désormais, nous devons Create HDL Wrapper et passer au Test Bench. Voici le code correspondant au Test Bench :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity gcd_tb is
--  Port ( );
end gcd_tb;

architecture Behavioral of gcd_tb is
component design_1_wrapper
    PORT(
         x_input, y_input : in STD_LOGIC_VECTOR(31 downto 0);
         res_output : out STD_LOGIC_VECTOR(31 downto 0);
         clk, rst, start_input : in STD_LOGIC;
         ready_output : out STD_LOGIC
     );
end component;

SIGNAL ready_ext, CLK_ext, RST_ext,start_input_ext : STD_LOGIC;
SIGNAL x_input_ext, y_input_ext, res_output_ext : STD_LOGIC_VECTOR(31 downto 0
);

begin

UUT : design_1_wrapper PORT MAP (ready_output => ready_ext, start_input => sta
rt_input_ext,
x_input => x_input_ext,y_input => y_input_ext, res_output => res_output_ext,
clk => CLK_ext, rst => RST_ext);

PROCESS BEGIN
    CLK_ext <= '0';
    wait for 15 ns;
    CLK_ext <= '1';
    wait for 15 ns;
END PROCESS;
```

```
PROCESS BEGIN
    x_input_ext <= x"00000009";
    y_input_ext <= x"00000006";
    RST_ext <= '1';
    start_input_ext <= '0';
    wait for 20 ns;
    RST_ext <= '0';
    start_input_ext <= '1';
    wait for 20 ns;
    wait;

END PROCESS;
```

Ensuite, nous pouvons regarder la simulation en vérifiant que *res_output_ext* sort le bon résultat :



Nous obtenons bien 3 en ayant mis 9 et 6 sur les entrées x et y. Notre IP fonctionne bien.