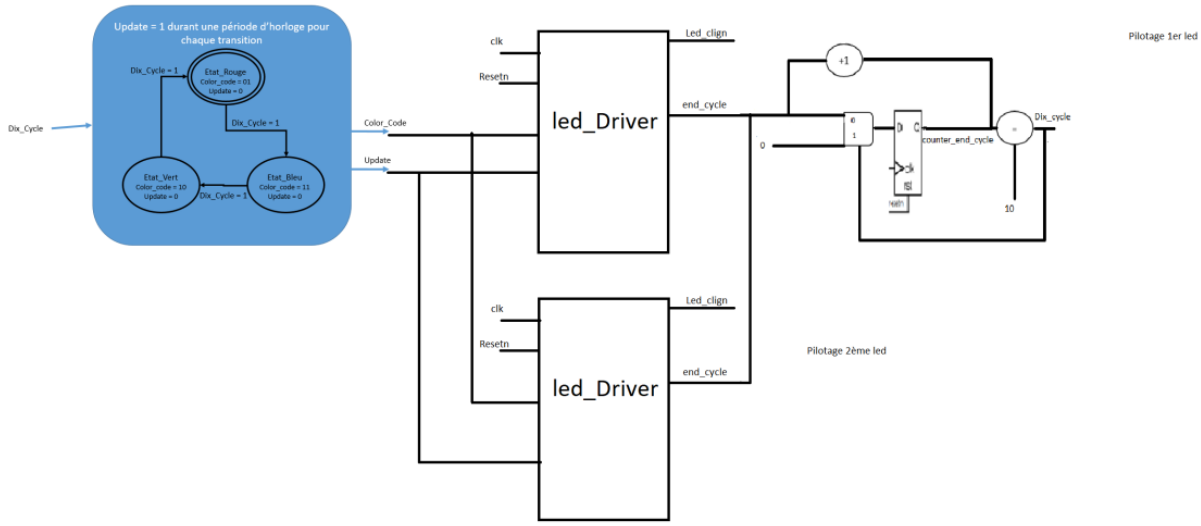
TP05 - Domaine Horloge (*Atmani Hicham & Kamal KHERCHOUCH*)

1. **A l'aide du module *LED_driver* du TP4, créez une architecture RTL permettant de piloter les deux LED RGB. Les LEDs RGB devront clignoter 10 fois en rouge puis 10 fois en bleu et 10 fois en vert avant de recommencer à partir du rouge. En entrée des modules *LED_driver* le signal *update* ne devra pas être à 1 pendant plus d'un coup d'horloge. Il doit s'agir d'une impulsion.**

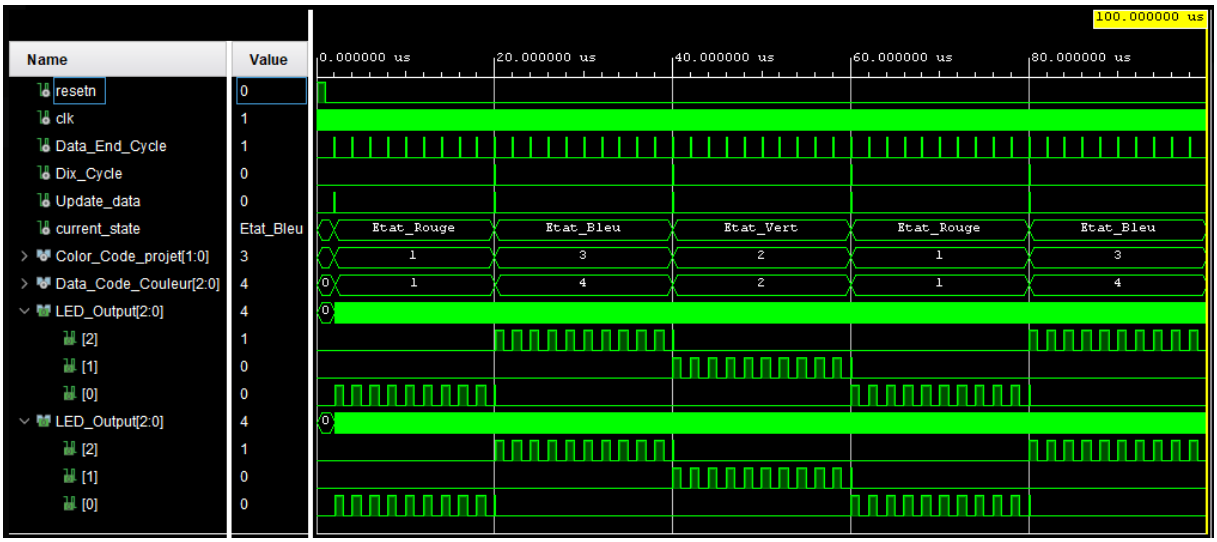


On utilise les modules Led_driver construit dans le TP précèdent. Il y a un module Led_Driver pour la gestion de la LED0 et un second pour la gestion de la LED1.
On ajoute également un compteur sur le signal End_Cycle afin de générer une impulsion (signal dix_cycle) au bout de 10 cycles d'End_Cycle.
Et pour finir, on ajoute une machine d'état qui va permettre de gérer les signaux code couleur et update.

2. **Rédigez le code VHDL correspondant à votre architecture.**

Cf. Code Domaine_Horloge.vhd

3. **Rédigez le testbench et simulez votre design. Vérifiez que les modules réceptionnent correctement le signal *update*.**



Ici, on voit bien que l'impulsion sur le signal Dix_Cycle intervient à Chaque fois qu'on compte 10 impulsions de Data_end_counter.
La machine d'état est bien validée car on fait bien la boucle entre rouge, bleu et vert.
Et on voit bien que les sorties LED_Output des LED0 et LED1 clignotent bien 10 fois avant de changer de couleur.

4. **Modifiez votre design pour gérer deux signaux d'horloge différents. La première horloge, *clkA*, est associé à la logique en dehors des modules *LED_driver* et au module *LED_driver* de la LED0. La deuxième horloge, *clkB*, est associé au module *LED_driver* de la LED1.**

Pour modifier le design, on remplace Clk par Clka et ClkB.
Clka remplacera tous les Clk utilisé dans notre code écrit précédemment, tandis que clkb ne sera utilisé que pour le deuxième module Led_Driver.

Cf. Code Domaine_Horloge.vhd

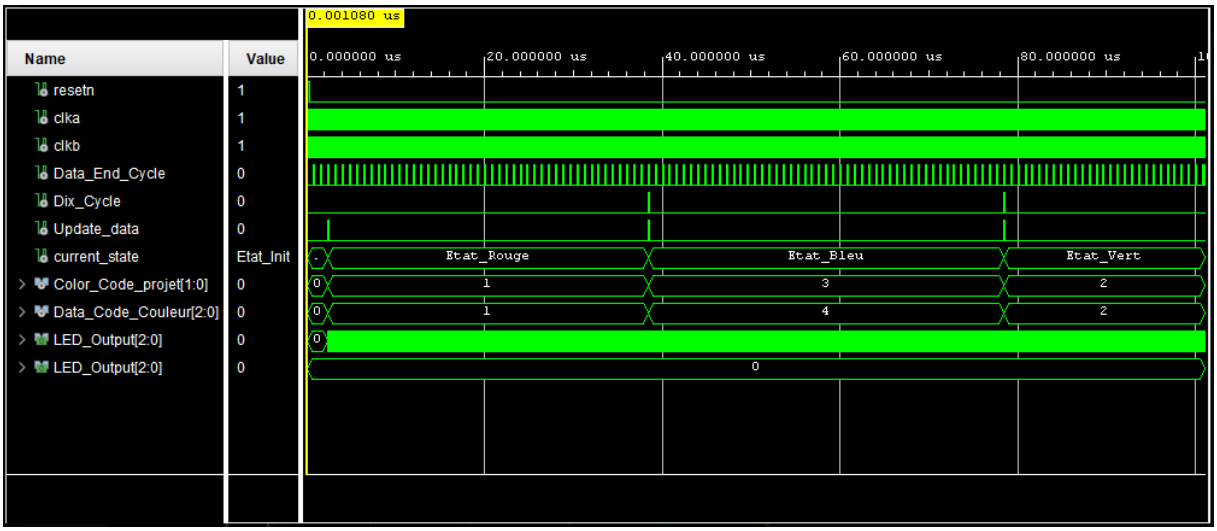
5. **Modifiez votre testbench tel que l'horloge *clkA* ait une fréquence de 50Mz et *clkB* 250MHz.**

Pour le testbench, on utilise deux process un pour clka et le second pour clkb avec des temps de période correspondant aux fréquences demandées.

```
constant hpA : time := 2 ns;      --demi periode de 2ns
constant periodA : time := 2*hpA;  --periode de 4ns, soit une frequence de 250MHz
constant hpB : time := 10 ns;      --demi periode de 10ns
constant periodB : time := 2*hpB;  --periode de 20ns, soit une frequence de 50MHz
```

Note : une erreur été présente dans l'énoncé, plus tard on a remplacé les valeurs des fréquences de clkA et clkB.

6. **Lancer une simulation. Que se passe-t-il au niveau des signaux *update* des modules *LED_driver* ? Qu'elle incidence cela a-t-il sur les LEDs ?**



Ici, on voit que le LED0 est fonctionnel tandis que la LED1 rencontre le phénomène du domaine d'horloge. Donc n'est pas fonctionnelle en l'état. Cela est dû au fait que le signal Update du second led driver (Clkb) n'est pas synchronisée avec le signal update du premier led driver (ClkA).

7. **Proposez une solution pour corriger le problème lié au changement de domaine d'horloge, vous pourrez vous aider du lien https://nandland.com/lesson-14-crossing-clock-domains/.**

On a utilisé la solution du stretch afin de synchroniser les deux signaux update des leds driver. On a étiré notre signal Update du second led driver afin de les synchroniser.

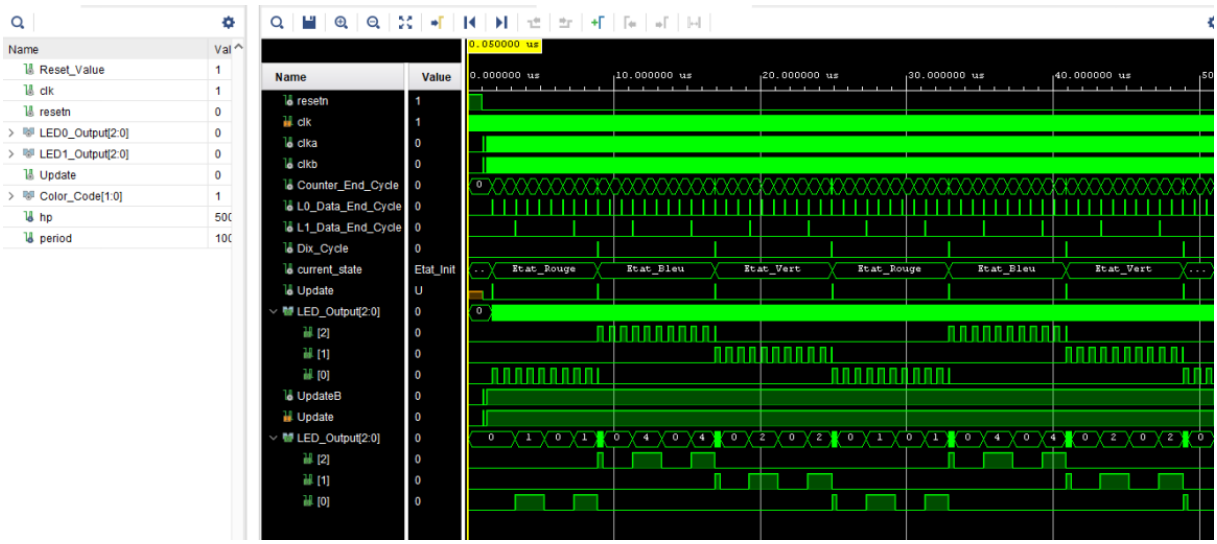
Pour ce faire on ajoute le bout de Code suivant (Compteur de stretch) et on affecte le signal UpdateB au signal update du second LedDriver.

```
----------Début process synchronisation des clock A et B----------------------
        process (clka) is
        begin
        if (rising_edge (clka)) then
            if ( Update_data ='1') then
                Count_stretch <= 4;
            elsif Count_stretch > 0 then  -- Evite l'overflow
                Count_stretch <= Count_stretch-1;
            end if;
        end if;
        end process;

----------Fin process synchronisation des clock A et B----------------------

----------Début combinatoire synchronisation des clock A et B----------------------
    UpdateB <= '1' when Count_stretch > 0
        else '0';
```

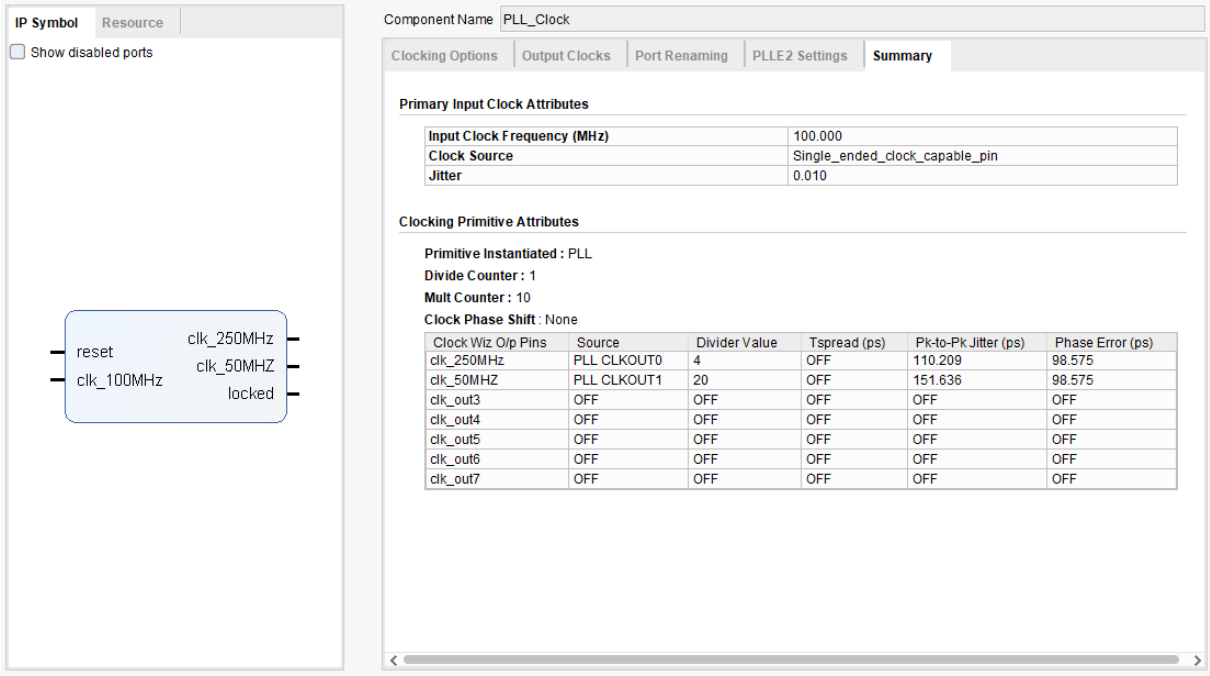
8. **Mettez en place votre solution et testez-la en simulation. Si votre résultat de simulation n'est toujours pas valide, proposez une autre solution.**



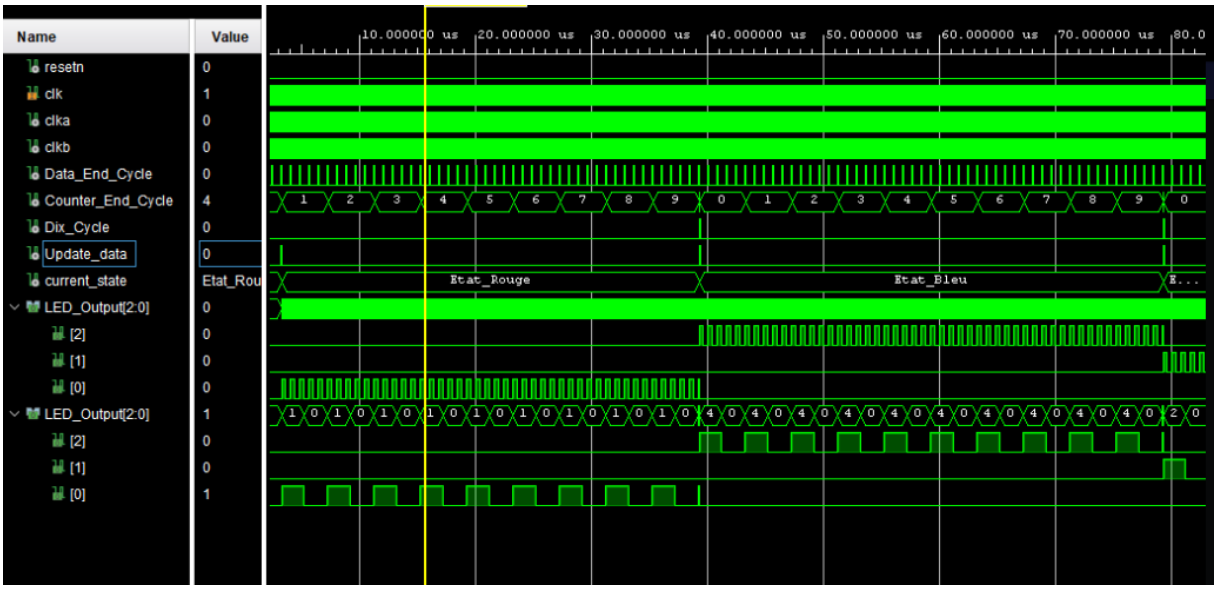
Ici, on voit bien que la LED0 dépend de la fréquence de clka et LED1 de celle de clkb. De plus, nos signaux de sortie des LEDs changent de couleurs en fonction des signaux updates respectives.

9. **Pour générer plusieurs horloges vous aurez besoin d'une PLL. Trouvez quel est le nom de l'IP PLL de l'IP Catalog de Vivado puis ajoutez là à votre architecture.**

On utilise le module clocking wizard présent dans le catalogue de Vivado.



En relançant une simulation après la mise en place de la PLL, on peut voir que cette dernière est fonctionnelle.



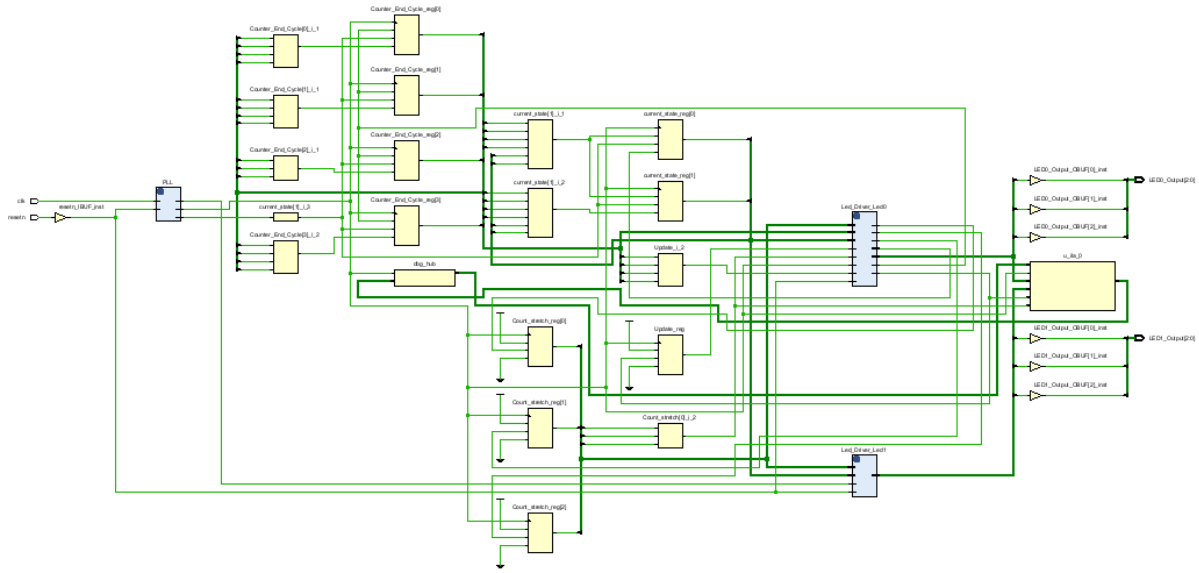
10. **Effectuez la synthèse et placer des sondes (ILA) sur les signaux pertinents pour vérifier que le changement de domaine d'horloge s'est correctement passé.**

On place des sondes ILAs sur :

- LED_Output (LED0)
- LED_Output (LED1)
- Update_data (LedDriver 0)
- UpdateB (LedDriver 1)

11. **Etudiez le rapport de synthèse.**

On obtient le schéma RTL suivant :



On y retrouve bien nos deux LedDriver, notre PLL, notre FSM, nos deux entrées (Resetn et Clk) et nos sorties (LED0 et LED1).

```
---------------------------------------------------------------------------------
Start RTL Component Statistics
---------------------------------------------------------------------------------
Detailed RTL Component Info :
+---Adders :
       2 Input    4 Bit       Adders := 1
       2 Input    3 Bit       Adders := 1
+---Registers :
                  4 Bit    Registers := 1
                  3 Bit    Registers := 3
                  2 Bit    Registers := 1
                  1 Bit    Registers := 7
+---Muxes :
       2 Input    4 Bit        Muxes := 1
       5 Input    3 Bit        Muxes := 2
       4 Input    2 Bit        Muxes := 1
       2 Input    1 Bit        Muxes := 4
       4 Input    1 Bit        Muxes := 2
---------------------------------------------------------------------------------
Finished RTL Component Statistics
---------------------------------------------------------------------------------
```

En comparaison avec notre schéma RTL de la question 1, on y retrouve bien le même nombre de adders, de registre et de Mux.

```
Report Cell Usage:
+------+----------+------+
|      |Cell      |Count |
+------+----------+------+
|1     |PLL_Clock |     1|
|2     |CARRY4    |    14|
|3     |LUT1      |     1|
|4     |LUT2      |     7|
|5     |LUT3      |     2|
|6     |LUT4      |    13|
|7     |LUT5      |     1|
|8     |LUT6      |    78|
|9     |FDCE      |    70|
|10    |FDRE      |     4|
|11    |IBUF      |     1|
|12    |OBUF      |     6|
+------+----------+------+
```

On compte bien le bon nombre de registres, d'entrée et de sortie (1 entrée pour la clock et 6 sorties pour les deux LEDs RGB).

Avec toutes ces analyses, on respecte donc bien notre schéma de base (solution de la question 1).

12. Effectuez le placement routage et étudiez les rapports générés.

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	0,272 ns	Worst Hold Slack (WHS):	0,036 ns	Worst Pulse Width Slack (WPWS):	1,250 ns
Total Negative Slack (TNS):	0,000 ns	Total Hold Slack (THS):	0,000 ns	Total Pulse Width Negative Slack (TPWS):	0,000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	3821	Total Number of Endpoints:	3805	Total Number of Endpoints:	2164

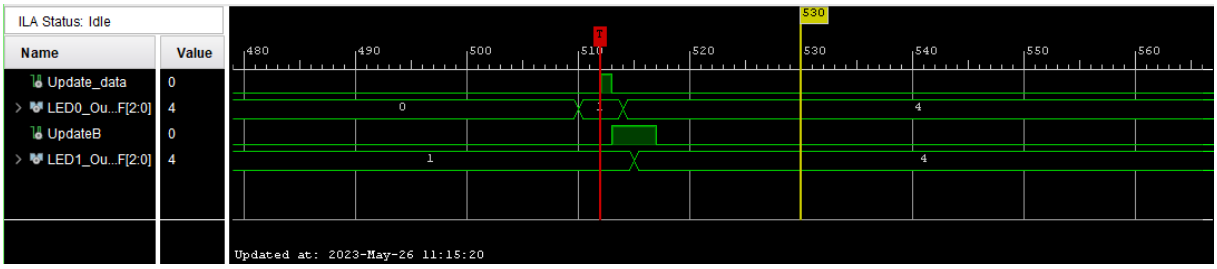
All user specified timing constraints are met.

Notre système ne devrait pas rencontrer de métastabilité, ni de Cross Clock domain (CCD) car on ne retrouve pas de stack (TNS et THS = 0ns).
Donc notre Stretch corrige bien notre problème de CCD.

```
Max Delay Paths
--------------------------------------------------------------------------------------
Slack (MET) :             0.272ns  (required time - arrival time)
  Source:                 <hidden>
                            (rising edge-triggered cell FDRE clocked by clk_250MHz_PLL_Clock  {rise@0.000ns fall@2.500ns period=5.000ns})
  Destination:            <hidden>
                            (rising edge-triggered cell FDRE clocked by clk_250MHz_PLL_Clock  {rise@0.000ns fall@2.500ns period=5.000ns})
```

On observe que chemin le plus long est celui utilisé pour la PLL et plus précisément l'horloge à 250MHz.

13. Générez le bitstream, programmez la carte et vérifiez les signaux du chipscope (ILA).



On peut voir à l'aide des sondes l'ILA que nos deux LEDs changent d'états en fonction des signaux Updates. De plus, on peut voir que le stretch à bien était réalisé sur le signal Update du second LedDriver.

Cf. Démo vidéo.