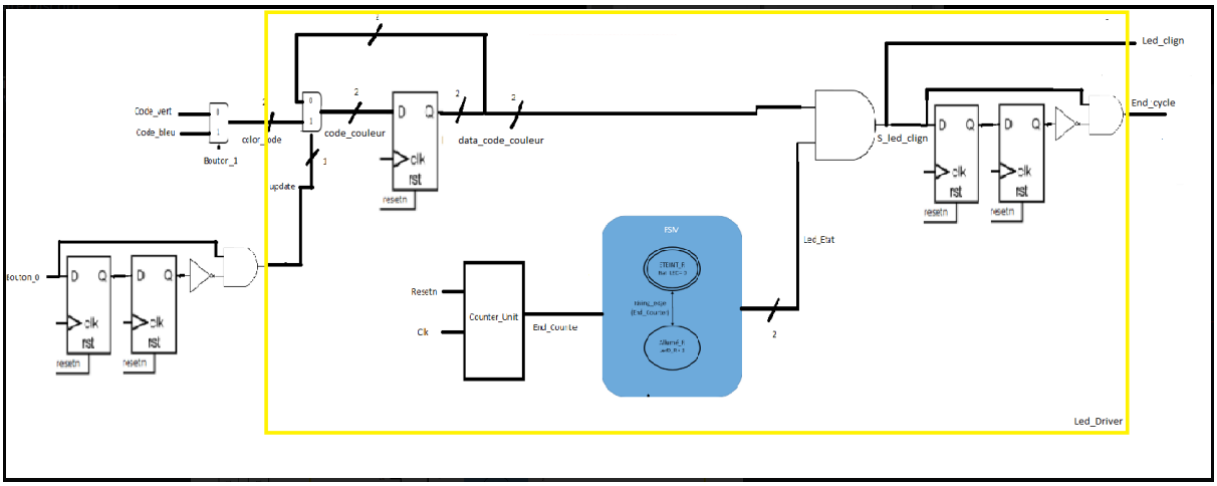
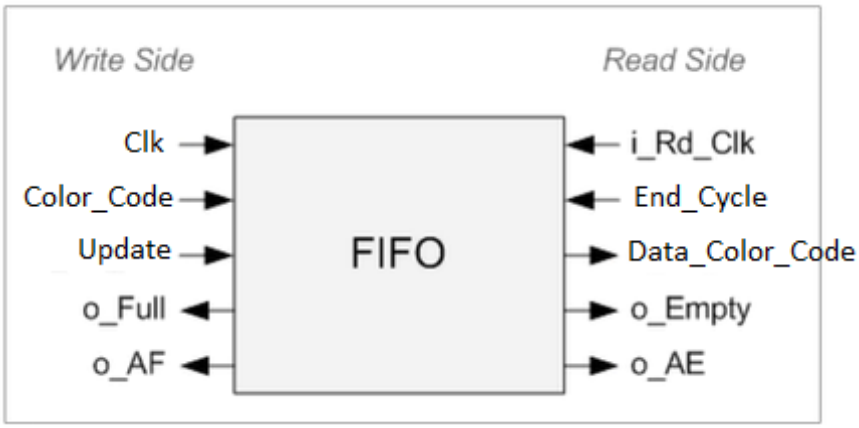
1. **Sur l'architecture RTL, modifiez le module *LED_driver* en ajoutant une sortie *end_cycle*. Cette sortie vaudra 1 à la fin d'un cycle allumé/éteint de la LED RGB.**



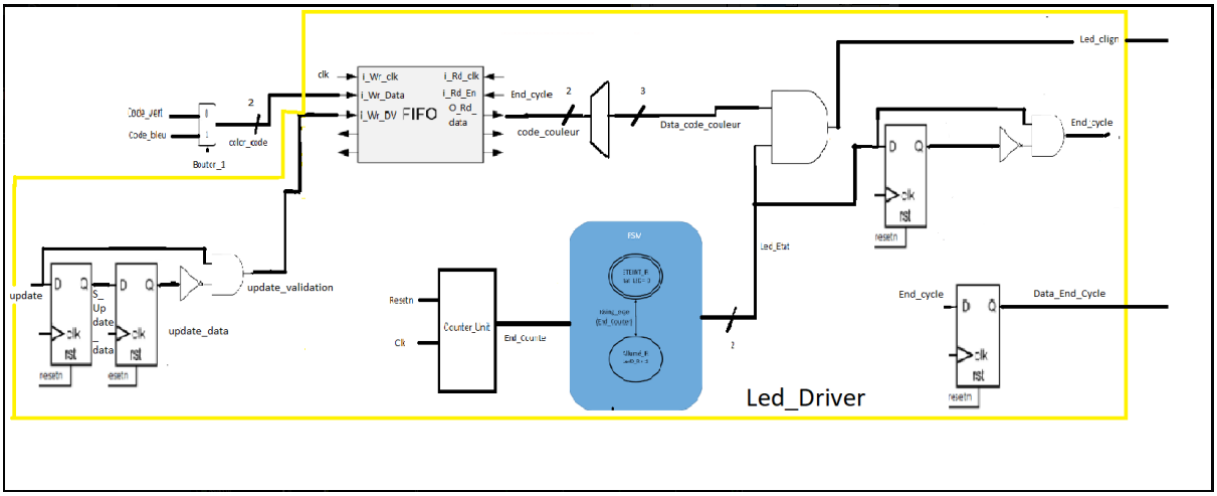

Sur l'architecture RTL de la fin du TP4 (part1), On a modifié le module Led_Driver afin d'ajouter une détection de front montant sur le signal Led_Clignotement. Ce qui nous permettra d'obtenir le signal end_cycle à chaque cycle allumé/éteint de la led RGB.

2. **Modifiez la logique en entrée du module pour ajouter une FIFO. Cette FIFO doit prendre en entrée le code couleur « vert » ou « bleu » suivant l'état du bouton_1 et est connectée en sortie à l'entrée *color_code* du module *LED_driver*. La donnée est écrite dans la FIFO lorsqu'il y a un front montant du bouton_0. La donnée de la FIFO est lue lorsque le signal *end_cycle* du module LED_driver vaut 1.**

On ajoute, à notre schéma RTL la fifo suivante :



En remplaçant la partie combinatoire d'entrée par une Fifo, on obtient le schéma RTL suivant :

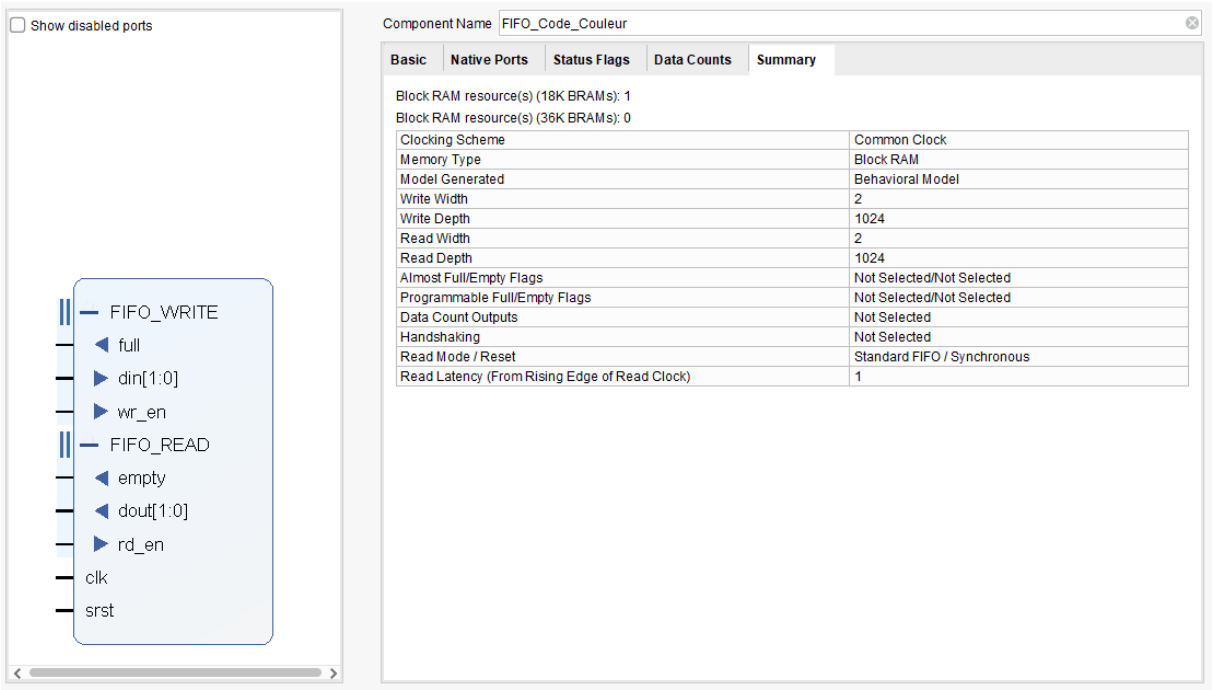



La Fifo créée un décalage entre l'entrée End_Cycle et la sortie Code_Couleur. Pour régler ce problème, il faut ajouter un registre sur le signal End_Cycle afin d'avoir les signaux Led_Clignotement et Data_End_Cycle synchronisé.

**3. Modifiez vos codes de la partie 1 pour y ajouter les nouveaux éléments de votre architecture.**

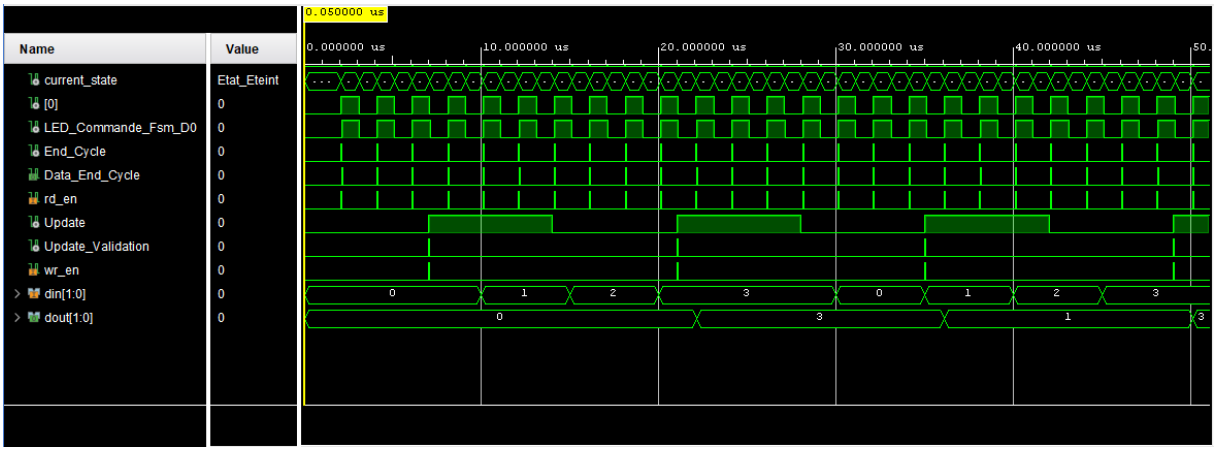
1ère étape : on modifie notre module Led_Driver en y ajoutant la Fifo.

Dans l'IP catalogue on choisit le composant FIFO Generator :

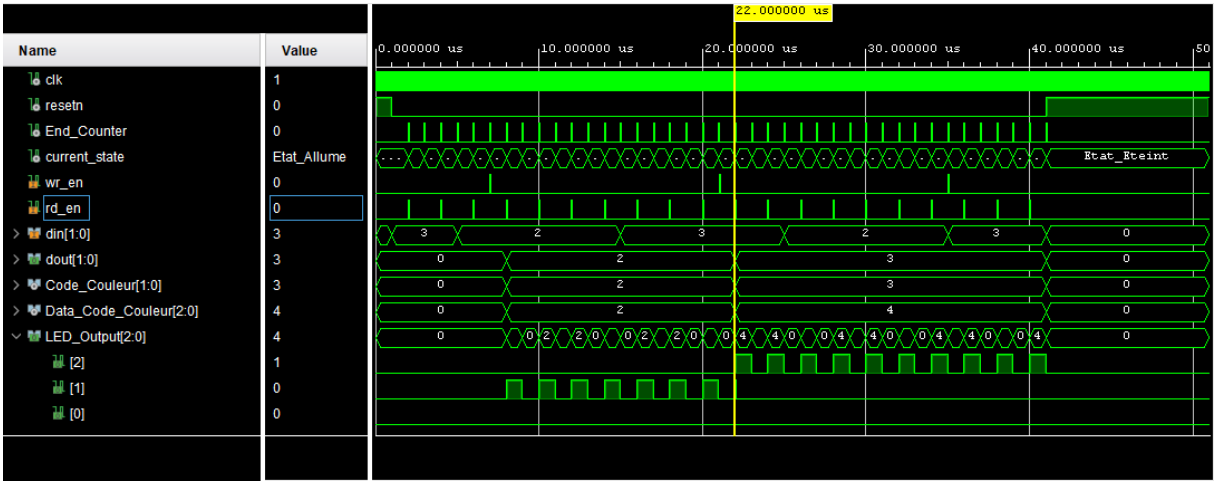



2ème étape : on modifie notre code en y ajoutant le module Led_Driver avec FIFO.

4. **Mettez à jour le testbench et réalisez une simulation pour vérifier votre design.**

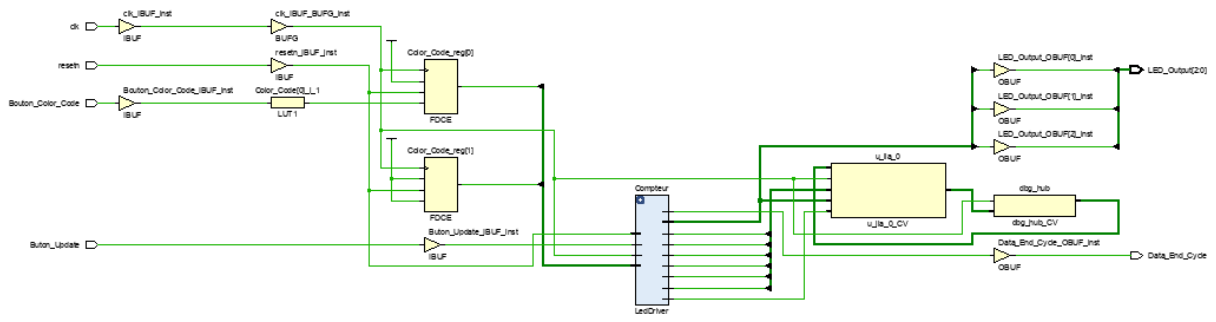



Cette simulation nous permet de valider notre composant Led_Driver avec utilisation d'une FIFO. Lorsque le signal wr_en de la FIFO détecte un front montant elle enregistre la valeur présente sur l'entre Din de cette dernière. Et lorsque le signal re_en passe à 1 cette valeur est envoyée à la sortie Dout de la FIFO.




La dernière simulation montre que notre système fonctionne bien avec la mise en place de la Fifo. En fonction du code couleur et le la sortie de le Fifo on fait clignoter la LED désiré.

5. **Réalisez une synthèse et étudiez le rapport de synthèse, les ressources utilisées doivent correspondre à votre schéma RTL.**



```
---------------------------------------------------------------------------------
Start RTL Component Statistics
---------------------------------------------------------------------------------
Detailed RTL Component Info :
+---Registers :
                   2 Bit    Registers := 1
                   1 Bit    Registers := 6
+---Muxes :
       2 Input    3 Bit        Muxes := 1
       5 Input    3 Bit        Muxes := 1
       2 Input    2 Bit        Muxes := 1
---------------------------------------------------------------------------------
Finished RTL Component Statistics
---------------------------------------------------------------------------------
```

Report Cell Usage:

| | Cell | Count |
|---|---|---|
| 1 | FIFO_Code_Couleur | 1 |
| 2 | BUFG | 1 |
| 3 | CARRY4 | 7 |
| 4 | LUT1 | 3 |
| 5 | LUT2 | 2 |
| 6 | LUT3 | 3 |
| 7 | LUT4 | 1 |
| 8 | LUT5 | 1 |
| 9 | LUT6 | 35 |
| 10 | FDCE | 35 |
| 11 | IBUF | 4 |
| 12 | OBUF | 4 |

On voit bien que l'ensemble, semble conforme à notre design et notre schéma RTL. Avec le bon nombre de registre, de multiplexeurs, d'adders et d'entrées / sortie de notre système.

### 6. Effectuez le placement routage et étudiez les rapports.

| Setup | | Hold | | Pulse Width | |
|---|---|---|---|---|---|
| Worst Negative Slack (WNS): | 2,058 ns | Worst Hold Slack (WHS): | 0,083 ns | Worst Pulse Width Slack (WPWS): | 2,750 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: | 3825 | Total Number of Endpoints: | 3809 | Total Number of Endpoints: | 2125 |

**All user specified timing constraints are met.**

Notre système ne devrait pas rencontrer de métastabilité car on ne retrouve pas de stack (TNS et THS = 0ns)

```
Max Delay Paths
--------------------------------------------------------------------------------
Slack (MET) :             26.613ns  (required time - arrival time)
  Source:                 dbg_hub/inst/BSCANID.u_xsdbm_id/SWITCH_N_EXT_BSCAN.bscan_switch/state_reg[0]/C
                            (rising edge-triggered cell FDRE clocked by dbg_hub/inst/BSCANID.u_xsdbm_id/SWITCH_N_EXT_BSCAN.bscan_inst
  Destination:            dbg_hub/inst/BSCANID.u_xsdbm_id/SWITCH_N_EXT_BSCAN.bscan_switch/portno_temp_reg[3]/D
                            (rising edge-triggered cell FDRE clocked by dbg_hub/inst/BSCANID.u_xsdbm_id/SWITCH_N_EXT_BSCAN.bscan_inst
  Path Group:             dbg_hub/inst/BSCANID.u_xsdbm_id/SWITCH_N_EXT_BSCAN.bscan_inst/SERIES7_BSCAN.bscan_inst/TCK
  Path Type:              Setup (Max at Slow Process Corner)
  Requirement:            33.000ns  (dbg_hub/inst/BSCANID.u_xsdbm_id/SWITCH_N_EXT_BSCAN.bscan_inst/SERIES7_BSCAN.bscan_inst/TCK rise@
  Data Path Delay:        6.404ns  (logic 1.816ns (28.357%)  route 4.588ns (71.643%))
  Logic Levels:           6  (CARRY4=2 LUT3=1 LUT4=1 LUT5=1 LUT6=1)
  Clock Path Skew:        -0.024ns (DCD - SCD + CPR)
```

On observe que chemin le plus long est celui présenté ici.

**7. Générez le bitstream et vérifiez que vous avez le comportement attendu sur carte.**

Cf Vidéo