TP4 partie 2
Jean Baptiste NARI

L'objectif de cette partie est de réaliser un design permettant de faire clignoter une LED RGB avec une séquence de couleurs entrées à l'aide des boutons. Dans cette partie, nous utiliserons le module LED_driver de la partie 1 et nous ajouterons l'utilisation d'un composant mémoire : la FIFO.

Question 1

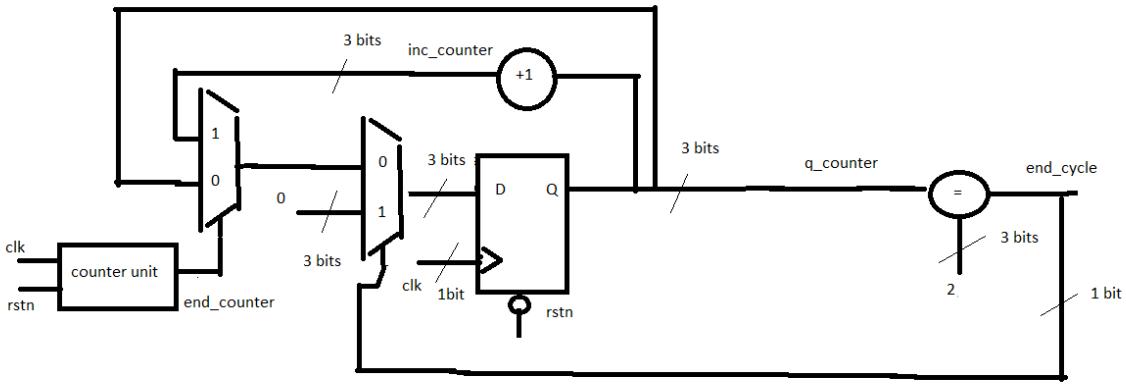
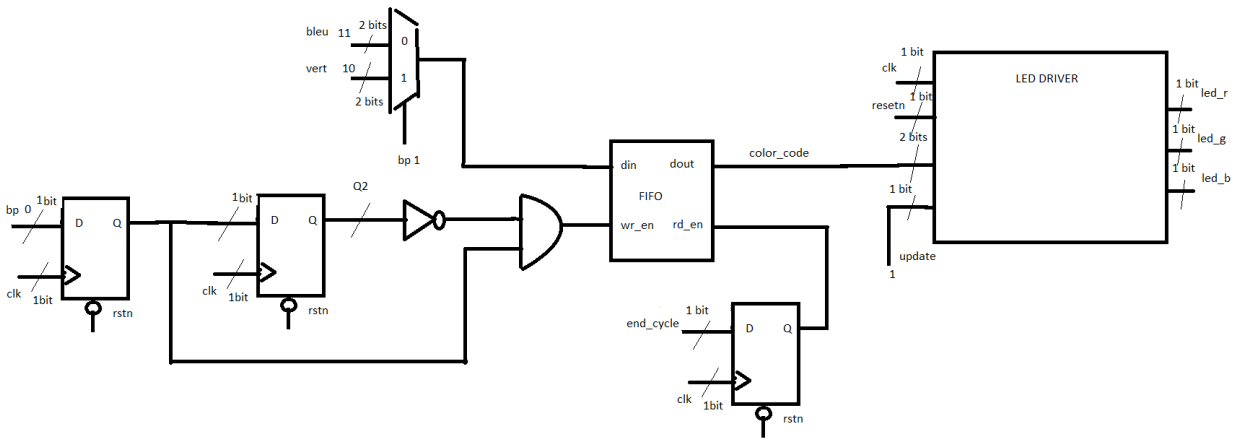


Schéma RTL permettant de créer le signal end_cycle. Ce signal est à l'état 1 à la fin d'un cycle allumé/éteint de la led rgb.

Question 2

Schéma RTL utilisant le block fifo :



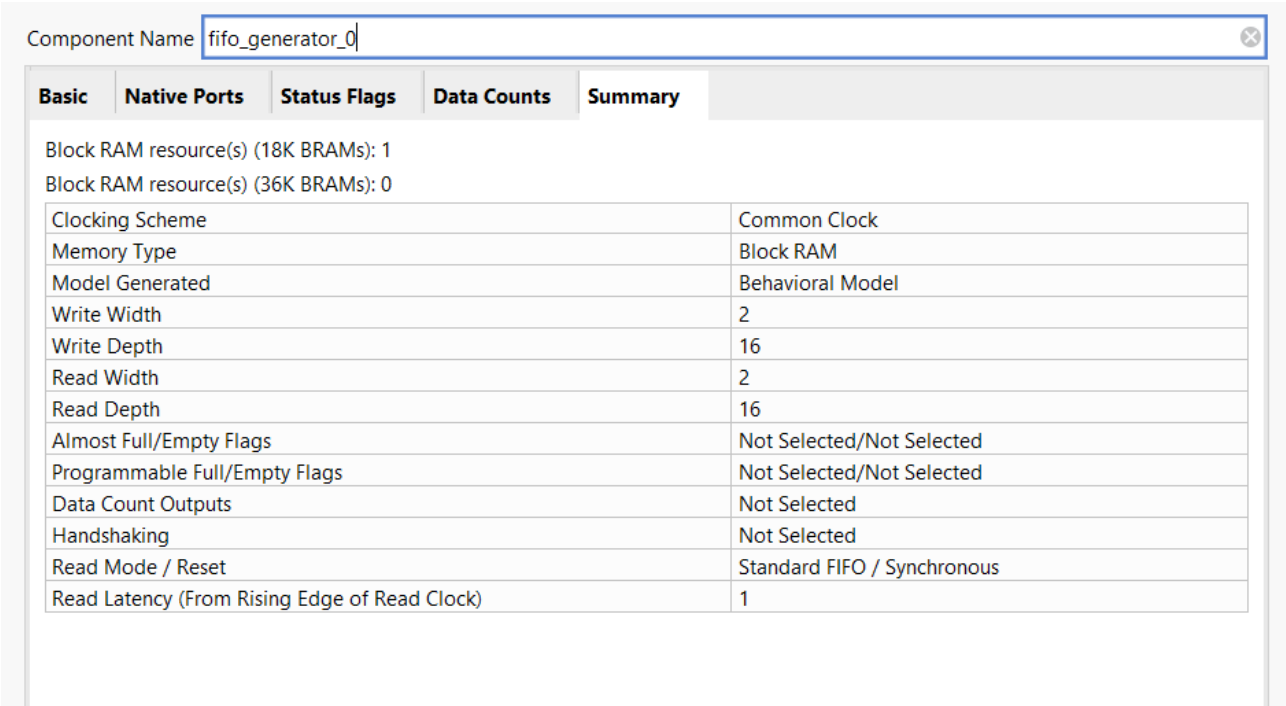
Le signal update est fixé à l'état 1 afin d'assurer une mise à jour en continu du module LED driver. Ainsi les signaux led_r, led_g, led_b, correspondront en permanance au code de couleur présent sur l'entrée color_code.

Le signal rd_en permettant la lecture de la fifo, correspond au signal end_cycle décalé d'un cycle d'horloge.

Génération d'une fifo à l'aide de fifo generator :

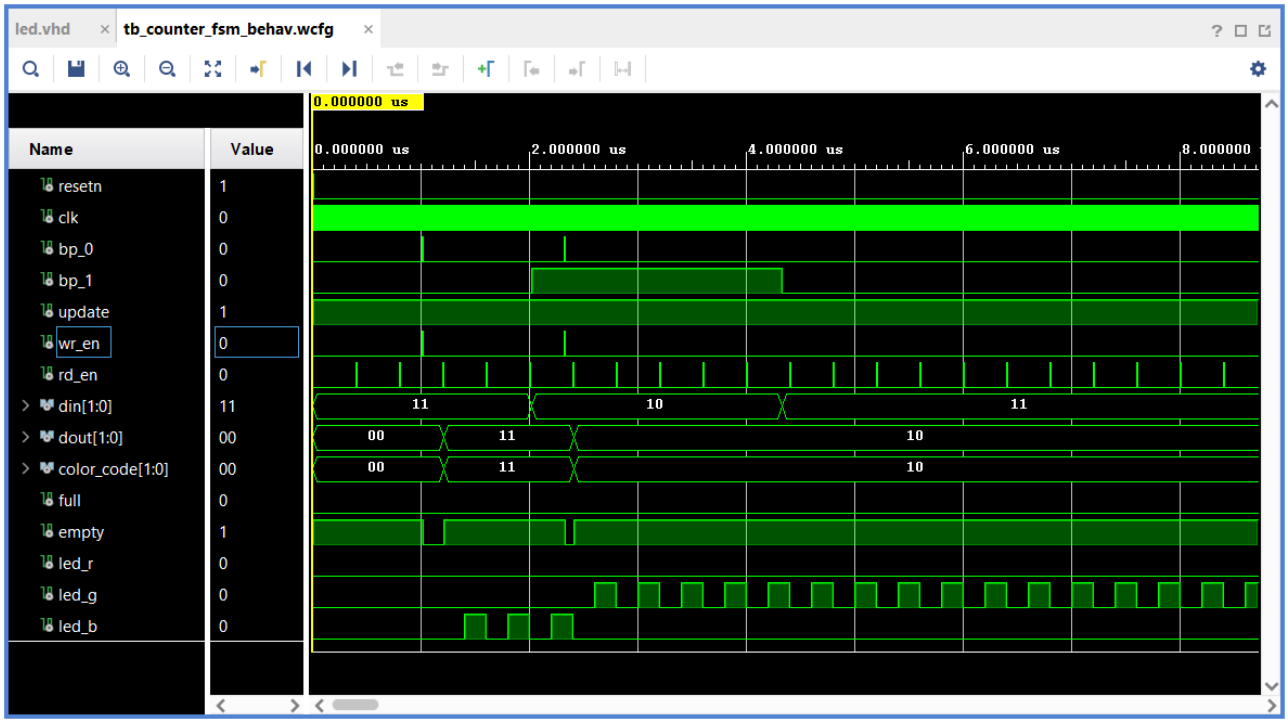
write width 2 bits
write depth 16 bits
read width 2 bits
read depth 16 bits

Résumé de création du block fifo :



Question 3,4

Résultat de simulation :



Lorsque wr_en est à l'état 1 la valeur présente sur din est memorisée dans la fifo.
Lorsque rd_en est à l'état 1 la valeur est lue dans la fifo.
Nous pouvons constater dès que wr_en est à l'état 1, le signal empty passe à l'état 0 ce qui confirme l'ecriture dans la fifo.
Dans ce chronograme nous venons lire la fifo alors qu'elle est vide, dans ce cas la fifo nous renvoie la valeur présente lors de sa dernière lecture.

Les boutons bp_0 et bp_1permettent d'engeristrer le color_code désiré dans la fifo. Et sur la sortie dout est présent le color_code qui déterminera le clignotement de la led verte ou bleu. Ce fonctionnement est conforme à nos attentes.

Question 5

Rapport de synthèse :

Report Cell Usage:

	Cell	Count
1	fifo_generator	1
2	BUFG	1
3	CARRY4	7
4	LUT1	1
5	LUT2	32
6	LUT3	3
7	LUT4	7
8	LUT6	3
9	FDCE	37
10	IBUF	4
11	OBUF	3

Il y a 37 FDCE (Registre à reset asynchrone) 28 pour le compteur, 2 pour le detecteur de fronts montants, 1 pour le signal end_cycle, 3 registres pour les trois bits de comptage de cycles allumé/éteint, 1 pour la machine à état, 2 pour la gestion du signal update de color_code.

Il y a 4 IBUF pour les entrées : clk, resetn, bp_0, bp_1.
Resetn est relié sur une digital IO car il n'y a que deux boutons poussoir présents sur la carte.
Il y a 3 OBUF pour les sorties : led_r, led_g, led_b

Question 6

Rapport de timing :

WNS(ns)	TNS(ns)	TNS Failing Endpoints	TNS Total Endpoints
5.000	0.000	0	86

WHS(ns)	THS(ns)	THS Failing Endpoints	THS Total Endpoints
0.139	0.000	0	86

TNS et THS sont à 0 il n'y a pas de violation de setup ou de hold.

Question 7

Génération du bitstream, programmation de la carte.

Après un premier appui sur bp0 la led clignote en bleu. Si l'on appui sur bp1 puis bp0 la led clignote en vert. Si l'on réappui de nouveau sur bp0 la led clignote en bleu.
Si l'on réalise plusieurs appuis sur les boutons bp1, bp0 (choix différents de couleurs) avant que la led ne clignote 1 fois cette sequence est mémorisé puis affiché dans les clignotements suivants.

C'est conforme au fonctionnement attendu.

Comparaison des deux parties du TP4

Dans les deux parties de ce tp4 nous avons réalisé le clignotement de la led rgb en utilisant deux implémentations différentes.

Première partie :

Le choix du color_code qui pilote le module led driver est réalisé par un multiplexeur.

Deuxième partie :

Le choix du color_code qui pilote le module led driver est réalisé par une fifo.

L'implementation de la première partie est la plus simpliste mais réalise tout de même la fonction attendu.
L'implementation de la deuxième partie est plus sophistiquée et rajoute une memorisation (dans la limite de la profondeur de la fifo) des color_code demandés.