TRAVAUX PRATIQUES

SYSTÈMES D'INFORMATION NUMÉRIQUES

2020-2021

Logique séquentielle

TP 4

Bascules D

Objectifs

- Étudier le fonctionnement des bascules synchrones de base
  - Fonctionnement général des bascules D
  - Rôle des entrées de commande asynchrones
  - Rôle de l'horloge
  - Rôle des entrées de commande synchrones
- Apprendre à exprimer un système séquentiel à partir d'un système combinatoire et de bascules
- Comprendre les principes de la simulation

Fonctions logiques étudiées

- Bascule D

Composants fournis

- D_ff : Bascule D

1. Introduction

Pour passer de la logique combinatoire, dans laquelle les sorties dépendent à tout moment des entrées, à la logique séquentielle, dans laquelle les sorties dépendent des entrées ET d'un état interne, il est nécessaire d'ajouter un élément permettant la mémorisation. Pour cela, n'importe quelle bascule permet de réaliser une mémoire. La plus simple d'entre elles et la plus utilisée est la bascule D.

2. Bascule D

2.1. Cahier des charges

Une bascule synchrone est caractérisée par le fait qu'elle dispose d'une entrée spéciale, appelée **horloge**, qui autorise la mise à jour des sorties sur son front montant. On considère deux modes de fonctionnement d'une bascule synchrone :

- le mode de fonctionnement **asynchrone** : commandé par les entrées asynchrones (clrn pour la mise à '0' et prn pour la mise à '1' de la sortie). Ce mode de fonctionnement est prioritaire sur le mode synchrone. Il permet d'initialiser ou de réinitialiser le composant.
- le mode de fonctionnement **synchrone** : évolution de la sortie par l'entrée d'horloge en fonction de l'entrée synchrone D. Il s'agit du fonctionnement « normal » du composant.

Nous nommerons **D_ff** le composant bascule D.

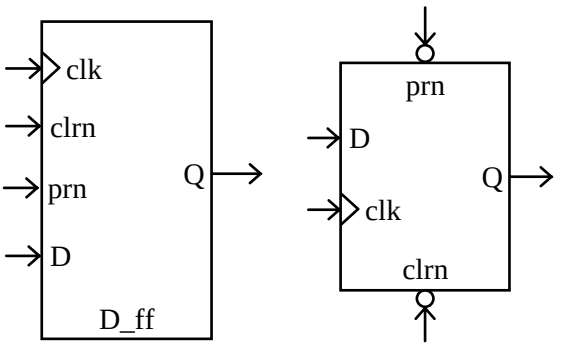
Vue composant et représentation usuelle d'une bascule

Nom du composant :

D_ff

Entrées/sorties :

clk : horloge
D : entrée synchrone
clrn : entrée de mise à zéro asynchrone
prn : entrée de mise à un asynchrone
Q : sortie de la bascule



Vue composant Représentation usuelle

2.2. Analyse – Modélisation

Complétez la table de fonctionnement de la bascule D. Q_n indique la valeur de la sortie Q à l'instant n, et peut dépendre de Q_{n-1}. Il n'y a pas de validation sur cette étape, mais vous pouvez faire vérifier celle-ci par un enseignant si vous avez un doute. Cette table de vérité doit être complétée pour la validation de la simulation.

	Entrées asynchrones		horloge	sortie
	clrn	prn	clk	Q_n
Mode asynchrone				
Mode synchrone			↑	
			↓	
			0	
			1	

2.3. Construction du composant

Le composant vous est fourni sur Moodle, il s'agit du fichier **D_ff.vhd** à placer dans le dossier **sources_composants**. Ouvrez le projet *composants*, supprimez tous les fichiers présents à l'exception de **sin_pack.vhd**, et ajoutez la bascule au projet.

Ajoutez le composant **D_ff** au package **sin_pack.vhd**, puis mettez le composant **D_ff** en top level, et procédez à la compilation.

Vous obtenez cette fois-ci deux warnings. Il s'agit d'avertissements liés au fait que l'on utilise une bascule sans contraindre l'une des entrées asynchrones, ce qui n'est pas normal dans un circuit. Exceptionnellement, n'en tenez pas compte, car étudier les entrées asynchrones est justement l'un des objectifs de cet exercice.

2.4. Simulation du composant

On rappelle le principe de création d'un fichier de simulation. Par la suite nous indiquerons uniquement « Simulation du composant », et vous devrez vous référer à cette section et/ou au guide pour retrouver le détail de la démarche.

Procédure de création d'un fichier de simulation

- Créez la feuille de simulation. On nommera toujours celle-ci du nom du composant simulé.
  - Création du fichier :
    - **File ➔ New**
    - Choisissez le type de fichier **Vector Waveform File**
  - Sauvegarde la feuille de simulation :
    - File ➔ Save
    - Placez-vous dans le dossier **simulations_composants** et saisissez pour nom de fichier **D_ff.vwf**
- Définition des paramètres de la feuille de simulation.
  - Il est nécessaire de définir le pas de la grille de simulation :
    - **Edit ➔ Grid size**
    - Modifiez le temps. La valeur à utiliser en SIN sera toujours de 100 ns.
  - On peut définir la durée de la simulation :
    - **Edit ➔ End time**
    - Modifiez la durée : il faudra à chaque fois estimer la durée nécessaire pour bien montrer tous les modes de fonctionnement. Pour cet exercice, choisissez 5 µs.
- Insertion sur la feuille de simulation des entrées-sorties du système à simuler.
  - Cliquez avec le bouton droit dans la colonne **Name** de la feuille de simulation
  - Sélectionnez **Insert ➔ Insert Node or Bus**…
  - Une nouvelle fenêtre s'ouvre :
    - Choisissez le Radix **Hexadécimal**
    - Cliquez sur **Node Finder**…
  - Une nouvelle fenêtre apparaît avec les champs suivants :

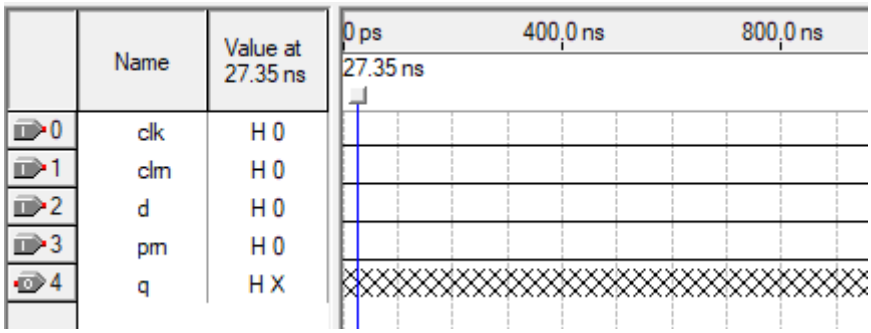
    Named * **Filter:** Pins: assigned

    look in: |D_ff|
  - Vérifiez que **Named** est bien suivi de * et **Look in** de |D_ff|
  - Modifiez **Filter :** pour le mettre à la valeur **Pins: all**
  - Cliquez sur le bouton **List**. Ceci fait apparaître tous les signaux du composant.
  - Sélectionnez les signaux à faire passer dans la fenêtre de droite : pour cela, il faut faire passer les signaux dans la partie **Selected Nodes**. Pour cela, il suffit de cliquer sur la double flèche vers la droite qui est entre les deux champs (>>) pour choisir tous les signaux.
  - Cliquez sur **Ok** pour valider la liste et revenez sur la feuille de répondant **OK** à nouveau.

Attention, la vue de la fenêtre est parfois mal réglée au début. Avant de passer à la suite, pointez la souris sur le chronogramme, maintenez la touche « ctrl » du clavier appuyée, puis descendez la molette de la souris jusqu'à ce que plus rien ne bouge, afin de vous assurer de bien visualiser l'ensemble de la simulation.

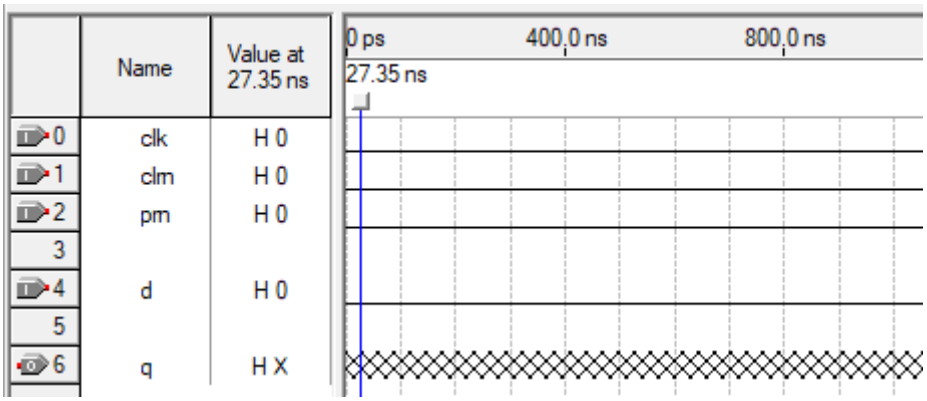
Organisation des signaux

À partir de là, vous disposez d'un chronogramme détaillant tous les signaux de la bascule D :



Il est tout d'abord nécessaire d'organiser les signaux dans un ordre logique. L'horloge et les signaux d'entrée asynchrones doivent être en premier, suivis des signaux d'entrée synchrones, puis des sorties. L'ajout d'un séparateur entre chacun de ces groupes de signaux permettra également plus de lisibilité. Pour ajouter un séparateur au-dessus d'un signal, faites un clic-droit sur le signal et sélectionnez **Insert ➔ Insert Waveform Divider**.

Dans le cas de la bascule D, on aura alors une présentation des signaux sous cette forme :



Que doit-on voir sur la simulation de la bascule D ?

La simulation de la bascule D doit faire apparaître tous les modes de la bascule. Comme il s'agit d'un composant synchrone, il faut commencer par définir le signal d'horloge. Cliquez sur le nom du signal *clk*, et choisissez l'outil de création d'horloge dans la barre d'outils (voir le guide pour la liste des outils). Saisissez une période de 100 ns et validez.

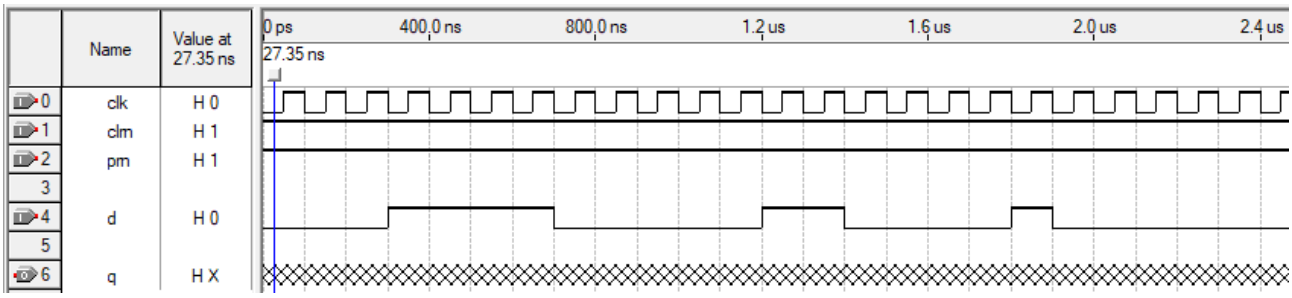
Il faut ensuite distinguer chacun des modes de fonctionnement de la bascule : le mode synchrone et le mode asynchrone.

Mode synchrone

Nous allons commencer par le mode synchrone. Pour l'activer, il faut que les deux entrées **prn** et **clrn** soient désactivées. Pour cela, cliquez sur le nom d'un des deux signaux, et choisissez l'outil « mise à 1 » afin de le désactiver sur toute la longueur de la simulation. Faites de même pour l'autre signal.

Ensuite, faites varier l'entrée **D** en sélectionnant plusieurs zones du signal que vous passerez à '1' pendant plusieurs cycles.

On obtiendra par exemple :



À ce niveau, vous pouvez visualiser le résultat en lançant la simulation :

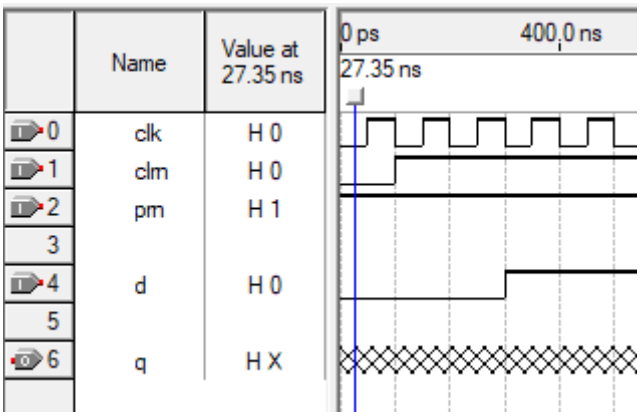
- Sauvegardez le fichier
- Faites **Processing ➔ Simulator tool** ou cliquez sur 

Une fenêtre s'ouvre :

- Vérifiez que **Simulation mode** est **Timing**
- Vérifiez que le nom du fichier de simulation est bien **D_ff.vwf** sinon il faut le changer en cliquant sur le bouton « **…** » et sélectionner le fichier précédemment créé.
- Cochez la case **Overwrite simulation input file with simulation results**
- Enfin, lancez la simulation en cliquant sur **Start**.

À la fin de la simulation, pour rafraîchir l'affichage, il faut revenir sur l'onglet présentant le chronogramme et choisir « Oui » dans la boîte de dialogue qui s'ouvre.

Vous devez alors remarquer que la sortie **Q** du composant reste à la valeur 'X' (valeur indéterminée). En effet, comme aucune entrée asynchrone n'est contrainte statiquement (voir remarque précédente sur les warnings), il est nécessaire d'initialiser la bascule avant de l'utiliser en appliquant au moins un niveau actif sur l'une des deux entrées asynchrones. Sélectionnez l'entrée *clrn* pendant une durée d'un cycle au début de la simulation, et appliquez-lui un niveau actif :



Sauvegardez, et relancez la simulation. Il n'est normalement pas nécessaire de configurer à nouveau le **Simulator tool**, cliquez directement sur **Start**.

Étudiez la simulation et vérifiez que vous avez bien compris le fonctionnement d'une bascule D.

Mode asynchrone

Votre simulation doit faire apparaitre tous les comportements. Pour une bonne visibilité des comportements asynchrones, il faut que la sortie de la bascule soit préalablement à '1' lorsque l'on fait une mise à '0' et vice-versa.

Choisir une section de la simulation dans laquelle le signal **D** est à '1' pendant plusieurs périodes d'horloge puis, au milieu de cette section, placez le signal **clrn** à sa valeur active pendant une période d'horloge. Enfin, dans une partie du chronogramme dans laquelle l'entrée **D** est à '0', placez le signal **prn** à sa valeur active pendant une période d'horloge.

Sauvegardez, et simulez à nouveau votre chronogramme en utilisant le **Simulator tool**. Votre simulation doit maintenant présenter un changement sur la sortie **Q** de la bascule.

Étudiez le comportement de la bascule pour vérifier que le comportement correspond bien à la table de vérité.

Faites valider la table de vérité et la simulation par un enseignant, auquel vous devrez expliquer les différents comportements de la bascule.

Dans le cas général, que doit faire apparaître la simulation d'un composant séquentiel ?

Vos simulations devront faire apparaître *tous* les comportements possibles :

- Initialisation : une simulation devra toujours commencer par l'activation du signal d'initialisation (en général, ce sera le signal *arazb*)
- Comportements asynchrones : utilisation à nouveau du signal d'initialisation à un moment pertinent, c'est-à-dire lorsqu'il y a effectivement une valeur à remettre à zéro.
- Comportements synchrones : remise à zéro synchrone (même remarque que ci-dessus), signal enable activé ou non, etc.
- Conflits entre les signaux : si un signal est prioritaire sur un autre (table de vérité avec priorité), montrez que c'est bien le cas pour votre composant en activant les deux signaux simultanément.

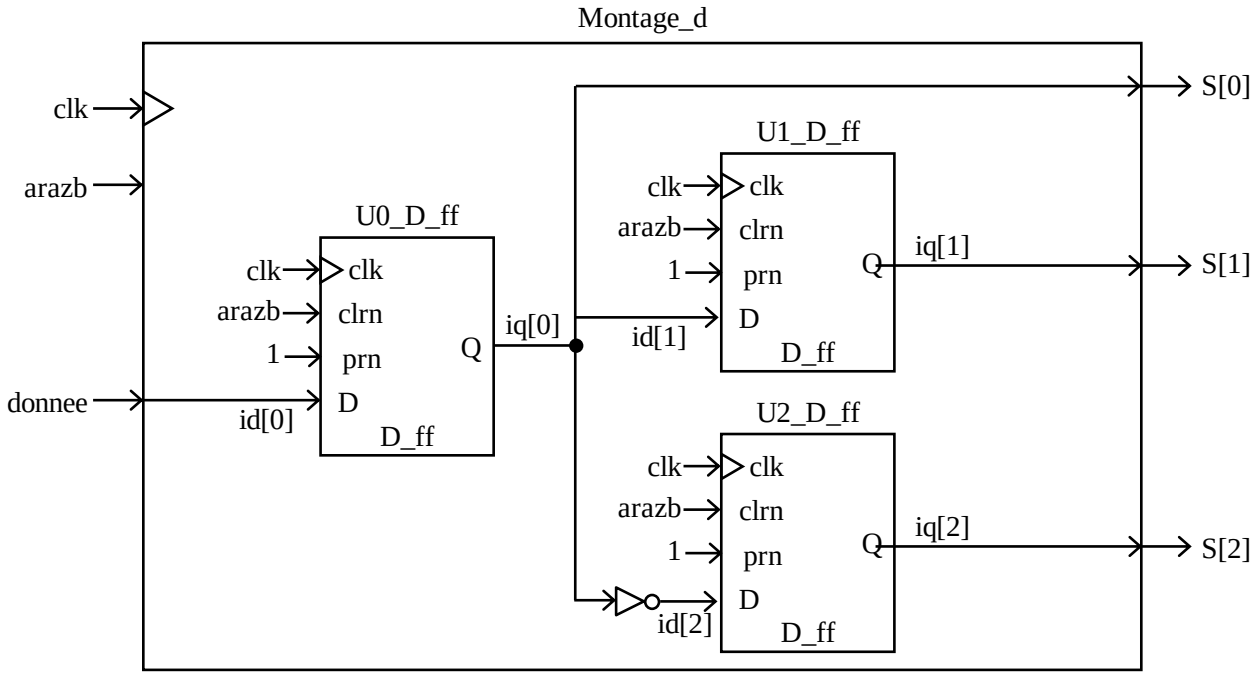
Il est important de séparer les différents comportements présentés en laissant plusieurs cycles d'horloge au cours desquels il ne se passe rien, permettant de prouver que votre composant n'évolue pas lorsqu'on ne le lui demande pas.

Lors de la validation, la présentation de la simulation à l'encadrant devra toujours être accompagnée d'une explication de chaque étape. Assurez-vous pour chaque étape que vous savez pourquoi la sortie réagit comme elle le fait lorsque l'entrée est stimulée de telle ou telle façon.

3. Mise en œuvre d'un système comportant plusieurs bascules

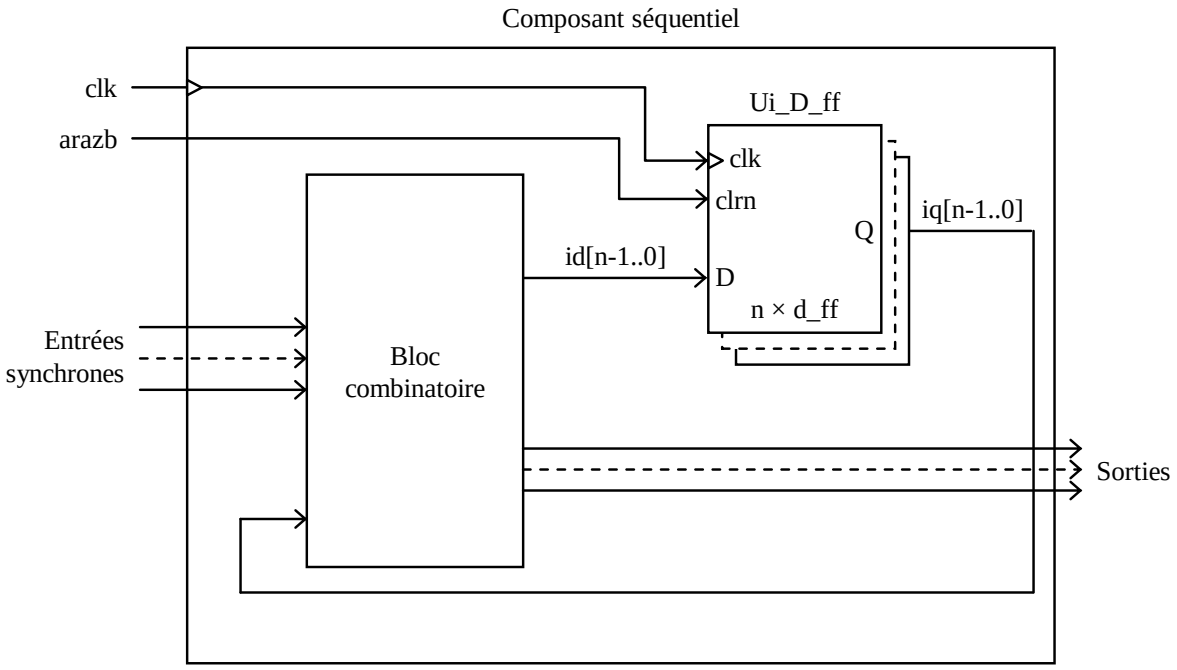
3.1. Cahier des charges

On souhaite maintenant étudier ce qu'il se passe lorsque plusieurs bascules sont placées en série.
Pour cela, on considère le composant **Montage_D** suivant :



3.2. Analyse – Modélisation

Il s'agit d'un circuit de logique séquentielle. Pour étudier les circuits séquentiels, nous utiliserons une approche permettant de généraliser la mise en œuvre. Celle-ci sera basée sur la décomposition du schéma en deux blocs comme suit :



Le bloc combinatoire n'est pas un composant mais un ensemble d'équations combinatoires, pouvant être par exemple représenté par une ou plusieurs tables de vérités. Il n'a donc pas de nom d'instanciation.

Les bascules étant empilées, on leur donne un nom d'instanciation générique : Ui_D_ff, le i étant remplacé pour chaque bascule par son numéro.

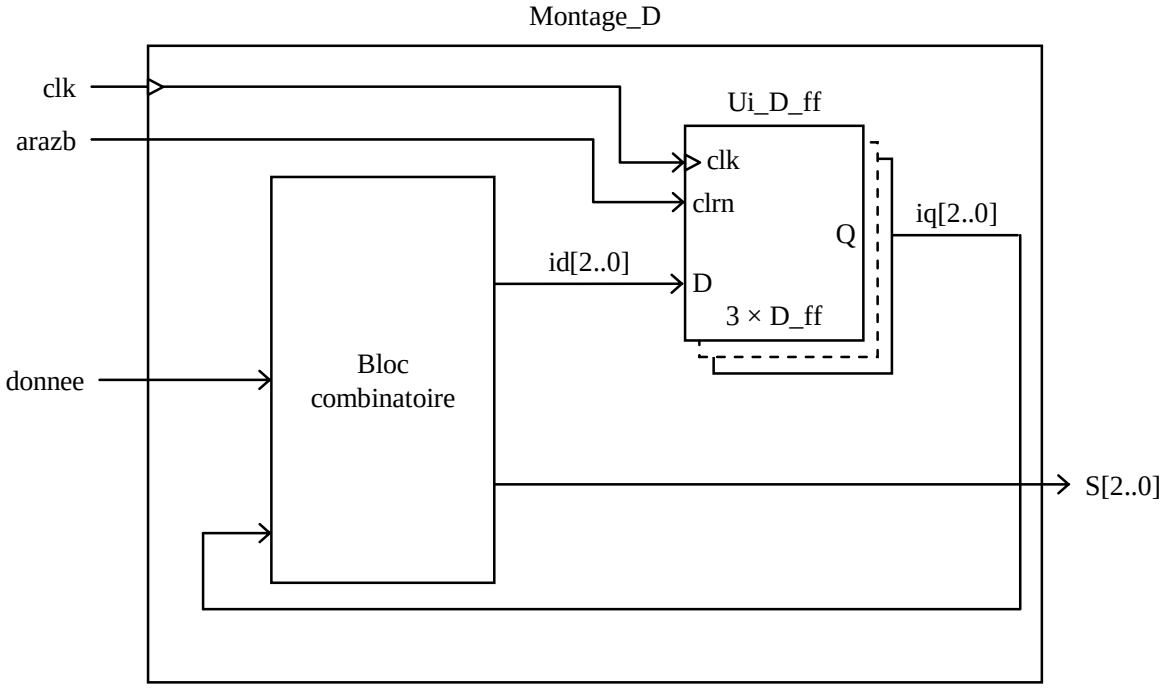
Les constantes de ce schéma, quel que soit le composant séquentiel, seront donc les suivantes :

- Les entrées *arazb* et *clk*
- L'entrée *prn* des bascules sera forcée à '1' (elle ne sera pas représentée pour simplifier le schéma),
- Les deux blocs, combinatoire et bascules,
- Les signaux internes *id* et *iq*.

Ce qui variera d'un composant à l'autre :

- Les entrées synchrones et les sorties,
- Le nombre de bascules (et donc la taille de *id* et *iq*),
- Le contenu du bloc combinatoire.

En adaptant le schéma précédent au composant décrit précédemment, on obtient donc ce schéma :



Il faut donc déterminer le contenu du bloc combinatoire. Celui-ci comporte deux sorties, chacune d'entre-elles étant représentée par une table de vérité ou une équation. Ici, nous disposons directement des équations de chaque signal en analysant le schéma initial du composant présenté en page précédente.

En suivant les exemples indiqués, définissez l'équation de chaque signal sortant du bloc combinatoire. Celles-ci doivent uniquement dépendre des entrées du bloc combinatoire. Il n'y a pas de validation sur cette étape, mais vous pouvez faire vérifier celle-ci par un enseignant si vous avez un doute. Cette table de vérité doit être complétée pour la validation de la simulation.

Signal	Équation
id[2]	
id[1]	iq[0]
id[0]	
S[2]	
S[1]	iq[1]
S[0]	

3.3. Construction du composant

Dans le fichier **Montage_D.vhd** :

- Remplissez l'entité,
- Déclarez les signaux internes,
- Instanciez les trois bascules,
- Définissez les équations des six bits générés par le bloc combinatoire.

3.4. Simulation du composant

Appliquez la démarche de simulation pour vérifier le bon fonctionnement de votre composant.

Si vous ne savez pas comment commencer, choisissez des stimuli proches de ceux des chronogrammes de la page précédente, que vous complèterez pour bien détailler tous les comportements.

Vérifiez que les résultats de la simulation correspondent bien à ce qui serait attendu du schéma initial. Comparez les signaux S[1] et S[0] : que constatez-vous ? Même question entre le signaux S[2] et S[1]. Faites valider la table de vérité ainsi que votre simulation .

TP 5

Registres

Objectifs

- Appliquer la démarche vue au TP précédent à partir d'un cahier des charges
- Étudier la notion de registre

Fonctions logiques étudiées

- Registre tampon
- Registre à décalage

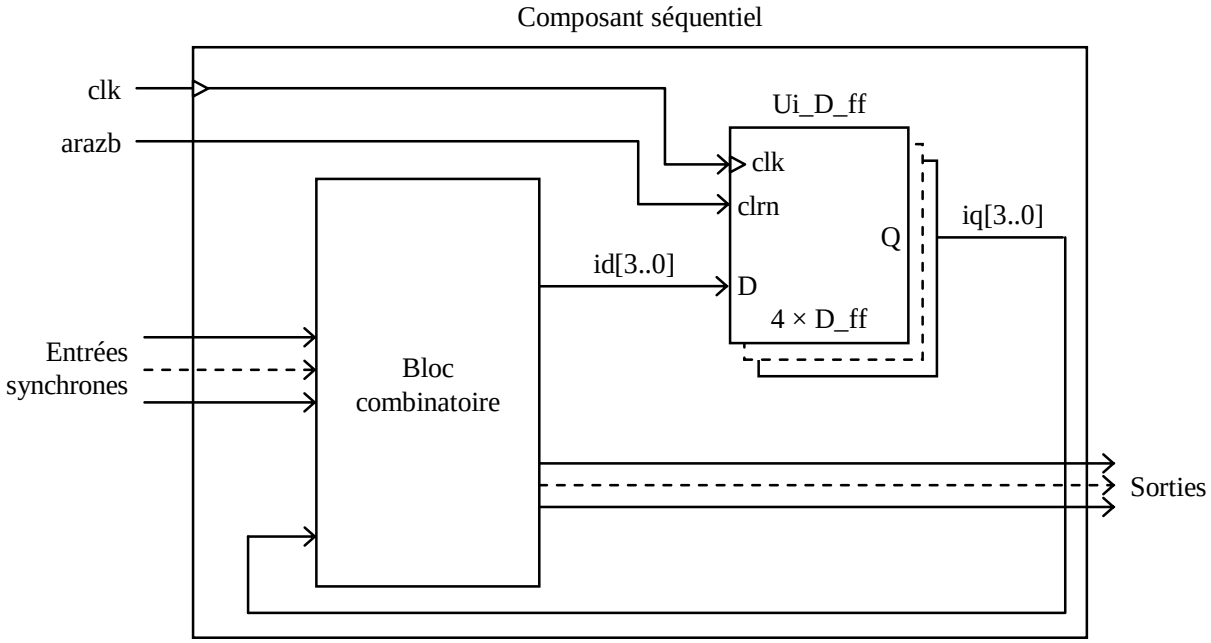
Composants fournis

- Modèle séquentiel : Trame descriptive pour les composants séquentiels
- D_ff : Bascule D
- Deco7seg : Décodeur pour afficheur 7 segments

1. Approche générale des systèmes séquentiels synchrones

Comme vu au TP précédent, les systèmes séquentiels synchrones ont une structure constituée :
- de bascules D (composant D_ff)
- d'un bloc combinatoire

Par exemple, pour un composant à quatre bascules, on aura la structure suivante :



On rappelle que si l'entrée *prn* n'apparaît pas sur ce schéma synthétique, ce n'est pas qu'elle n'est pas connectée, mais qu'elle est forcée à sa valeur inactive (donc '1', car l'entrée est active à l'état bas).

2. Registre tampon

2.1. Cahier des charges

On veut réaliser un registre tampon simple de 4 bits, **Reg_tampon**, avec mise à zéro asynchrone (*arazb*), mise à zéro synchrone (*sraz*) et autorisation de chargement synchrone (*sload*). *sraz* sera prioritaire sur *sload*.

Le fonctionnement du registre sera le suivant : lorsque l'autorisation de chargement est active, le composant enregistre la valeur de l'entrée *E[3..0]* dans sa mémoire interne (les bascules). Lorsque l'autorisation est inactive, la mémoire conserve sa valeur précédente. Une remise à zéro synchrone remet la valeur de chacun des bits de la mémoire à 0. La sortie *S[3..0]* du composant reflètera simplement l'état de la mémoire.

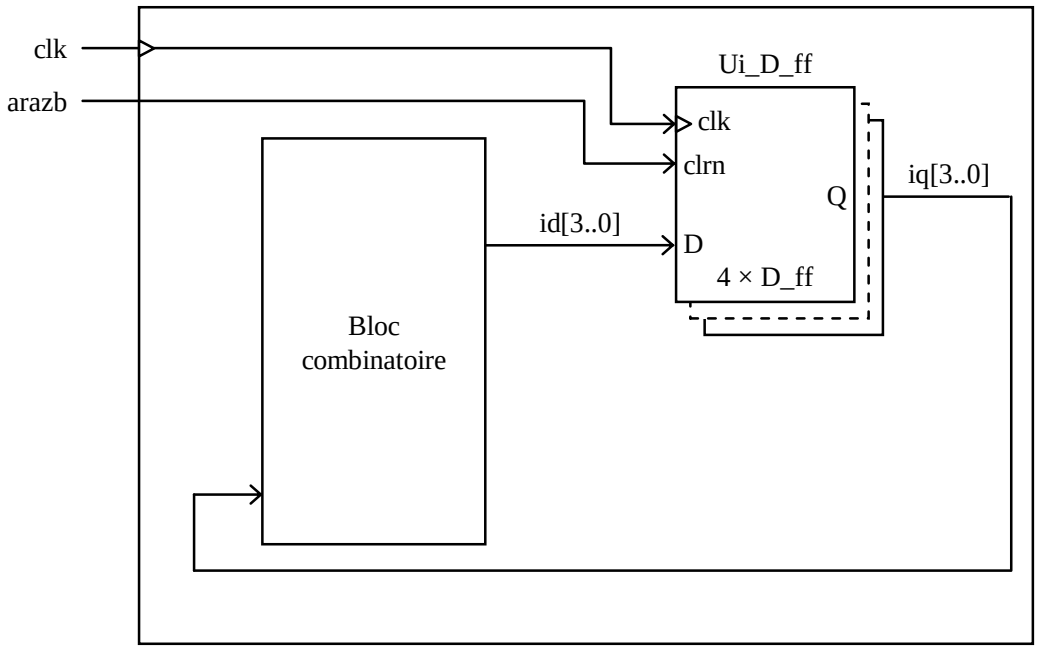
2.2. Analyse – Modélisation

À partir du cahier des charges, complétez les informations sur le composant ci-dessous : nom et liste des entrées/sorties à gauche et vue composant à droite. La liste des entrées doit être exhaustive : outre les entrées synchrones, on n'oubliera pas les entrées asynchrones et l'horloge.

Vue composant du registre tampon	
Nom du composant : Entrées / sorties :	

Complétez la vue interne ci-dessous en ajoutant les entrées et les sorties manquantes et en les reliant à l'élément interne correct. N'oubliez pas le nom du composant.

Vue interne du registre tampon



Complétez la table de vérité ci-dessous qui décrit le comportement de la sortie id[3..0] *du bloc combinatoire.*

sraz	sload	id[3..0]

La sortie *S[3..0]* du composant devant simplement recopier l'état des bascules, on obtient un table de vérité extrêmement simple, que l'on représentera de cette manière :

*	S[3..0]
*	iq[3..0]

Celle-ci pourra se résumer en VHDL à une recopie de signal :

```
S(3 downto 0) <= iq(3 downto 0);
```

2.3. Construction du composant

Vous trouverez sur Moodle un fichier modèle dans lequel les bascules sont déjà instanciées. Téléchargez le fichier **Modèle séquentiel.txt** disponible sur Moodle et placez-le dans le dossier **sources_composants**.

Dans Quartus, créez un fichier **Reg_tampon.vhd**, et copiez à l'intérieur le contenu du fichier **Modèle séquentiel.txt**. Modifiez ensuite le code du fichier VHDL :

- Saisissez le nom du composant dans l'entité et l'architecture,
- Ajoutez les entrées et les sorties dans l'entité,
- Comme on utilise le composant **D_ff**, ajoutez l'appel au package **sin_pack** en haut du fichier,
- Complétez l'architecture de manière à intégrer les tables de vérité ci-dessus,
- Placez le fichier en top level,
- Compilez.

2.4. Simulation du composant

Vérifiez le fonctionnement du composant par simulation. Référez-vous au guide Cyclone ou au TP précédent pour les détails de chaque étape.

Sur la simulation, on fera apparaitre les comportements suivants :

- Tout d'abord, la simulation doit commencer par une initialisation du composant : effectuez une remise à zéro asynchrone au premier cycle,
- Ensuite, testez le fonctionnement synchrone :
  - Activez et désactivez plusieurs fois l'entrée *sload* avec différentes valeurs sur *E[3..0]*,
  - Remettez à zéro de manière synchrone lorsque le composant a une valeur non nulle sur *S[3..0]*,
  - Vérifiez la priorité entre *sload* et *sraz* en activant les deux simultanément.
- Enfin, effectuez une remise à zéro asynchrone lorsque le composant a une valeur non nulle sur *S[3..0]*.

Simulez et vérifiez que le comportement est correct. Si ce n'est pas le cas, modifiez le composant, compilez à nouveau et re-simulez.

Lorsque le comportement est correct, faites noter le schéma, les tables de vérité et la simulation par un enseignant.

3. Registre à décalage

3.1. Cahier des charges

On veut réaliser un registre à décalage avec entrée parallèle *E[3..0]* et sortie série *S*.

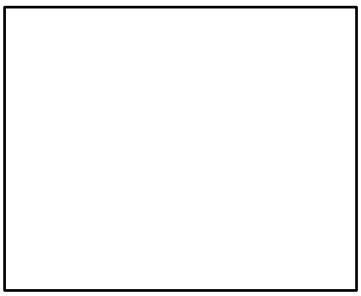
Ce registre comportera 4 bits et sera doté d'une mise à 0 asynchrone (*arazb*, active à 0), d'un chargement synchrone (*sload*, actif à 1) et d'une autorisation de décalage (*shift*, active à 1). Lorsque l'entrée *shift* vaudra 0 on ne fera pas de décalage. On décalera le contenu de la mémoire d'un bit vers la **gauche** lorsqu'elle vaudra 1. Le bit 0 sera remplacé par la valeur 0 à chaque décalage.

Le bit 3 du registre sera recopié sur la sortie série. Par ailleurs, afin de mieux comprendre le fonctionnement du composant, on rajoutera une sortie *visu[3..0]* qui recopiera la valeur de l'ensemble de la mémoire.

sload sera prioritaire sur *shift*.

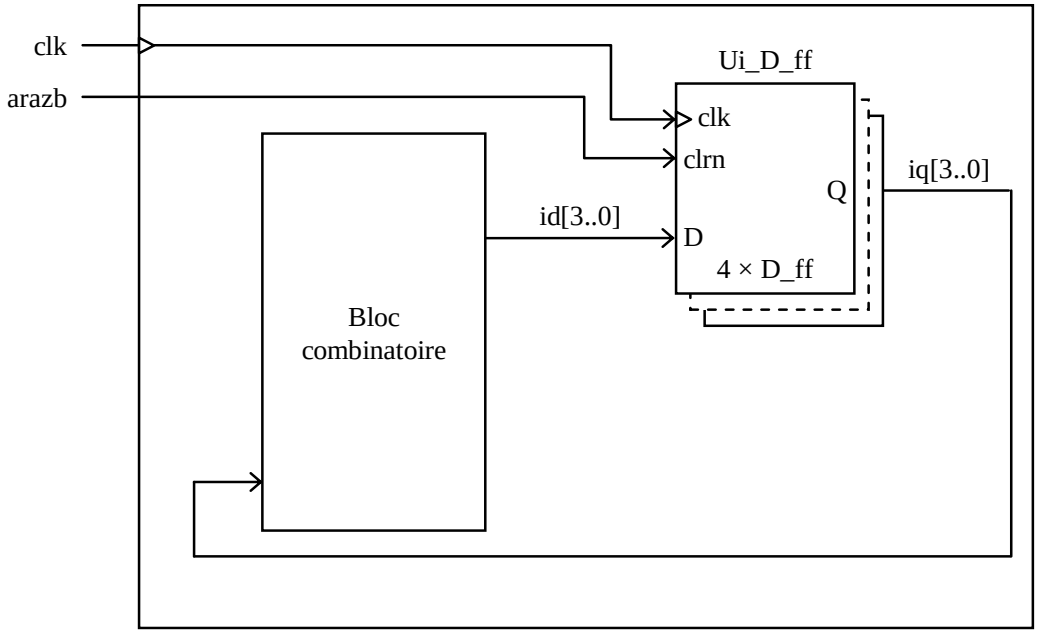
3.2. Analyse – Modélisation

Complétez les informations sur le composant à gauche et la vue composant à droite ci-dessous

Vue composant du registre à décalage	
Nom du composant : **Reg_decalage** Entrées / sorties :	

Complétez la vue interne ci-dessous.

Vue interne du registre à décalage



Complétez les tables de vérité ci-dessous qui décrivent le comportement du bloc combinatoire.

sload	shift	id[3..0]

*	S
*	

*	visu
*	

Si vous avez un doute, vous pouvez faire vérifier les schémas et les tables de vérité par un enseignant.

3.3. Construction du composant

Créez le fichier **Reg_decalage.vhd**. Vous pouvez répéter l'opération consistant à créer le fichier dans Quartus et y copier le contenu du fichier **Modèle séquentiel.txt** ou directement copier le fichier **Reg_tampon.vhd** sous le nom **Reg_decalage.vhd**. Traduisez en VHDL le comportement du composant.

3.4. Simulation du composant

Simulez le composant. Comme d'habitude, il faudra veiller à faire apparaitre tous les comportements sur la simulation. Pour cela, on propose de présenter la simulation de cette manière :

- Chargez une valeur alternant des 0 et des 1 dans le composant (par exemple, la valeur 0xA, soit "1010" en binaire) en plaçant cette valeur sur l'entrée *E[3..0]* et en activant l'autorisation de chargement *sload* pendant un seul cycle,
- Activez l'autorisation de décalage pendant au moins 4 cycles d'horloge pour visualiser un décalage complet de tous les bits,
- Chargez une nouvelle valeur dans le composant,
- Décalez la valeur une ou deux fois puis interrompez le décalage pendant quelques cycles,
- Activez à nouveau de décalage.

N'oubliez pas de commencer la simulation par une remise à zéro asynchrone, et de tester le comportement de la remise à zéro asynchrone ensuite au cours de la simulation, à un moment où la valeur de la sortie *visu[3..0]* est non nulle.

Une fois que vous avez vérifié le que tous les comportements du composant sont bien corrects, faites valider les schémas, les tables de vérité et la simulation par un enseignant auquel vous expliquerez le fonctionnement du composant grâce à votre simulation.

TP 6

Compteurs

Objectifs

- Étudier le fonctionnement des compteurs
- Savoir manipuler des tables de vérité avec une structure plus évoluée

Fonctions logiques étudiées

- Compteur binaire
- Compteur/décompteur généralisé

Composants fournis

- Modèle séquentiel
- D_ff : Bascule D
- Deco7seg : Décodeur pour afficheur 7 segments

1. Compteur binaire synchrone simple

1.1. Cahier des charges

On veut réaliser un compteur synchrone simple de 4bits avec mise à 0 asynchrone (*arazb*), mise à zéro synchrone (*sraz*) et autorisation synchrone de comptage (*en*). On nommera le composant **Cpt_simple**.

sraz sera prioritaire sur *en*.

La sortie *S[3..0]* visualisera l'état interne du compteur.

Une sortie supplémentaire *fin_cpt* permettra de détecter de façon combinatoire si l'état interne du compteur vaut "1111".

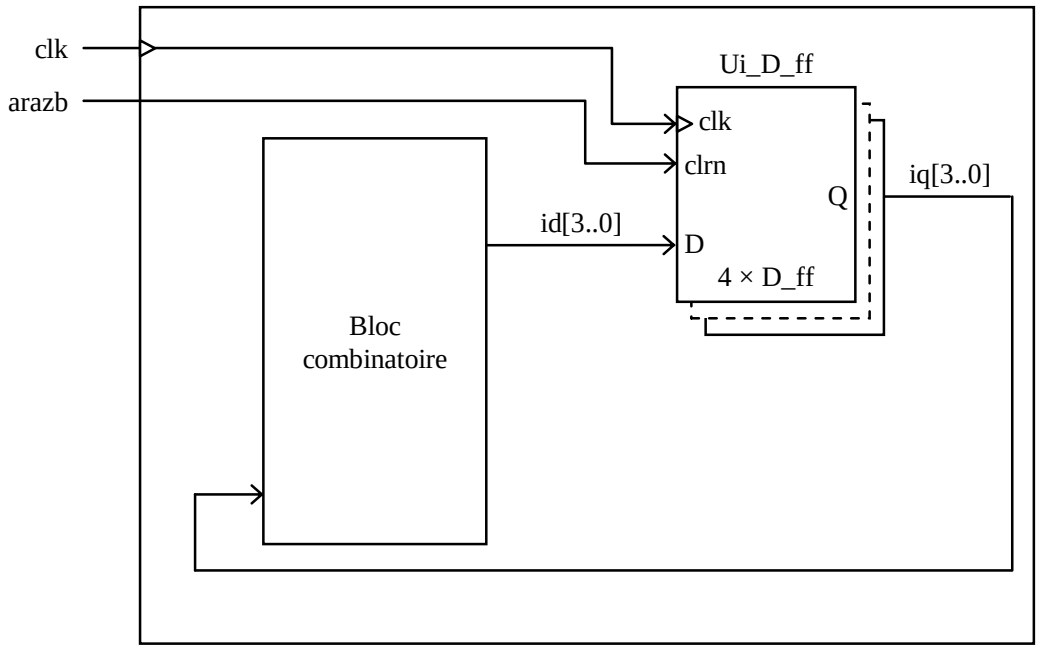
1.2. Analyse – Modélisation

Complétez les informations sur le composant à gauche et la vue composant à droite ci-dessous.

Vue composant du compteur binaire synchrone simple	
Nom du composant : Entrées / sorties :	

Complétez la vue interne ci-dessous.

Vue interne du compteur binaire synchrone simple



Complétez les tables de vérité ci-dessous qui décrivent le comportement du bloc combinatoire.

sraz	en	id[3..0]

*	S[3..0]
*	

*	fin_cpt	
*		si …
*		sinon

Si vous avez un doute, vous pouvez faire vérifier les schémas et les tables de vérité par un enseignant.

1.3. Construction du composant

Dans Quartus, créez un fichier **Cpt_simple.vhd**, et copiez à l'intérieur le contenu du fichier **Modèle séquentiel.txt**. Décrivez en VHDL le comportement du composant.

1.4. Simulation du composant

Simulez le composant. On veillera à faire apparaitre sur la simulation un cycle complet (de 0x0 à 0xF), ainsi que le retour à zéro à la fin. Toutes les combinaisons possibles des entrées *sraz* et *en* doivent apparaitre de manière groupée. On n'oubliera pas de faire apparaitre un *arazb* en début de simulation et un en cours de simulation, lorsque *S[3..0]* a une valeur non nulle.

Faites noter les schémas, les tables de vérité et la simulation par un enseignant.

2. Compteur-décompteur généralisé

2.1. Cahier des charges

On veut réaliser un compteur-décompteur généralisé **Cpt_dcptg** de 4 bits. Comme son nom l'indique, ce composant dispose de deux modes : compteur ou décompteur.

Le composant comportera les commandes suivantes, par ordre de priorité :
- mise à zéro asynchrone (*arazb*, active à 0),
- mise à zéro synchrone (*sraz*, active à 1),
- chargement synchrone (*sload*, active à 1),
- autorisation synchrone de comptage (*en*, active à 1),
- mode comptage ou décomptage (*ud* : 0 = comptage ; 1 = décomptage),
- entrées de chargement synchrone (*E[3..0]*).

La sortie du composant sera *S[3..0]*

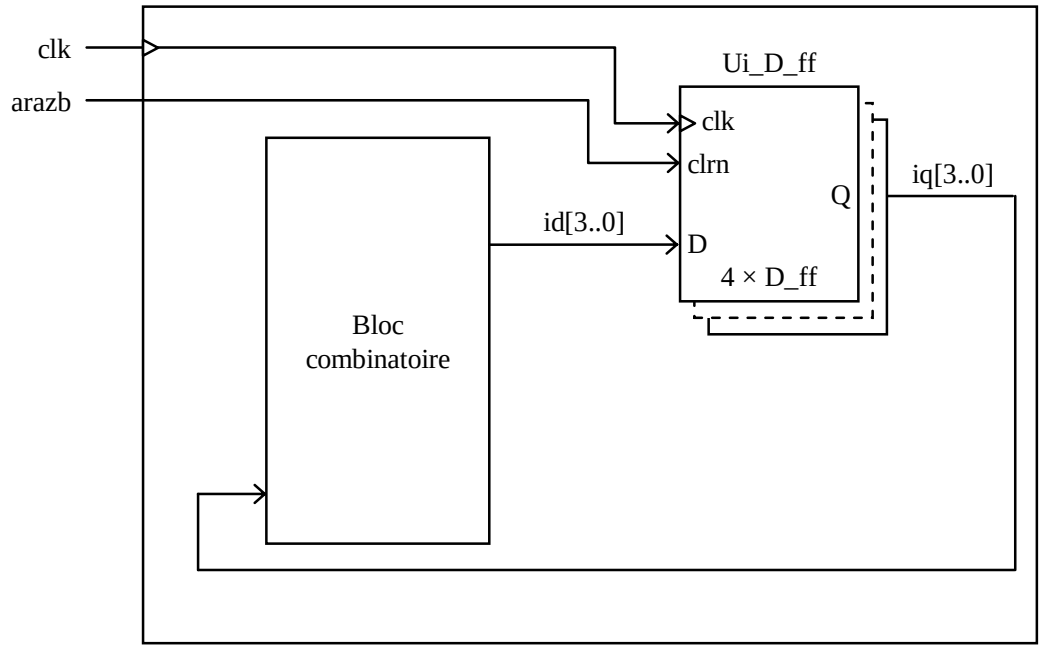
2.2. Analyse – Modélisation

Complétez les informations sur le composant à gauche et la vue composant à droite ci-dessous.

Vue composant du compteur-décompteur généralisé	
Nom du composant : Entrées / sorties :	

Complétez la vue interne ci-dessous.

Vue interne du compteur-décompteur généralisé



Complétez les tables de vérité ci-dessous qui décrivent le comportement du bloc combinatoire.

sraz	sload	en	ud	id[3…0]

*	S[3..0]
*	

Si vous avez un doute, vous pouvez faire vérifier les schémas et les tables de vérité par un enseignant.

2.3. Construction du composant

Décrivez **Cpt_dcptg.vhd,** soit à partir de **Modèle séquentiel.txt**, soit en copiant le fichier **Cpt_simple.vhd** et en modifiant la traduction en VHDL de la table de vérité.

2.4. Simulation du composant

Sur la simulation, vous devrez faire apparaitre les deux modes composant en réalisant un cycle complet de comptage et de décomptage, et en laissant le compteur aller au moins une valeur au-delà du maximum ou du minimum.

On fera également apparaître les priorités entre les différentes entrées en activant plusieurs d'entre elles simultanément. N'oubliez pas de faire apparaître des remises à zéro synchrones et asynchrones, à des moments bien choisis. Faites noter les schémas, les tables de vérité et la simulation à un enseignant.