ARC TD3

Automate
(1 séance sur machine $\simeq$ 4h voire un peu plus), 9 avril 2024

Construit à partir du poly de TD-TP des cours IF-AC et IF-AO

Dans ce chapitre, vous allez construire une implémentation matérielle d'un algorithme, sous la forme d'un circuit séquentiel complexe avec séparation de la *partie contrôle* (automate à nombre fini d'états) et de la *partie données* (chemin de données). Pour gagner du temps, la partie données du circuit vous est fournie dans le sujet ; votre travail consistera à implémenter l'automate de contrôle, et à exécuter le circuit pas à pas pour bien en comprendre le fonctionnement.

On veut construire un diviseur, c'est à dire, un composant qui calcule le quotient Q de la division entière d'un nombre positif ou nul N par un autre nombre positif P.

1 Sur papier !

1.1 Algorithme

Afin de simplifier l'exercice, on va adopter un algorithme itératif simpliste, qui procède par additions successives en calculant p, $p+p$, $p+p+p$, $p+p+p+p$, et ainsi de suite jusqu'à dépasser N. Quand la somme partielle dépasse N, alors le résultat de la division est le nombre de fois qu'on a pu ajouter p à lui-même.
L'algorithme utilise deux variables auxiliaires, x pour la somme partielle, et q pour le quotient partiel. Lorsque $x+p$ dépasse n, alors on a terminé, et q est le résultat recherché.

```
n := entrée N
p := entrée P
x := 0
q := 0
tant que x+p ⩽ n
    x := x+p
    q := q+1
fin tant que
sortie Q := q
```

QUESTION 1 ▶ executez l'algorithme pour $N = 12$ et $P = 5$

1.2 Interface

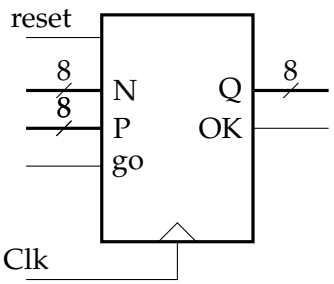


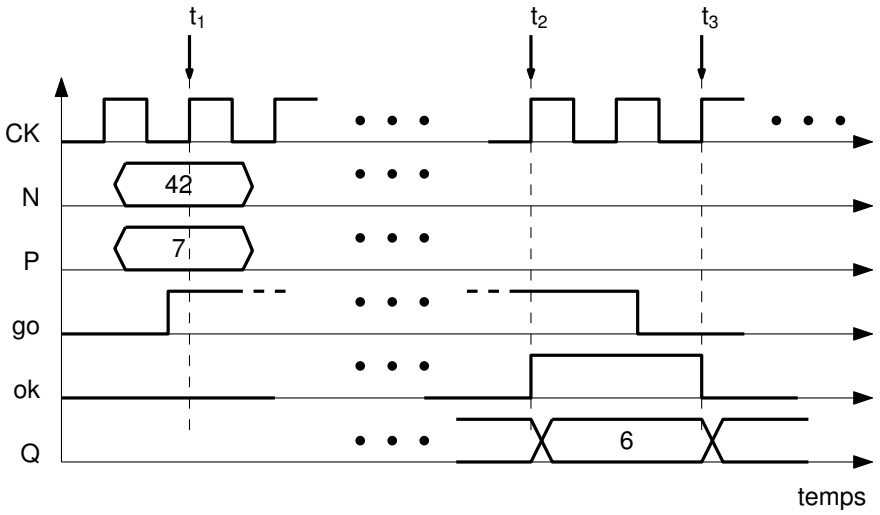
FIGURE 1 – L'interface de notre circuit avec son environnement

On suppose que les nombres N et P sont tous les deux positifs, et sont codés en binaire sur 8 bits. Le signal d'entrée *go* ordonne au composant de se mettre à travailler, et le signal de sortie *ok* permet au composant de signaler qu'il a terminé son calcul.

QUESTION 2 ► Mettez des flèches sur tous les signaux qui apparaissent sur la figure 1 pour indiquer dans quel sens ils vont.

1.3 Protocole

Le protocole de communication avec notre diviseur est illustré dans le chronogramme ci-dessous, et expliqué dans le texte qui suit.

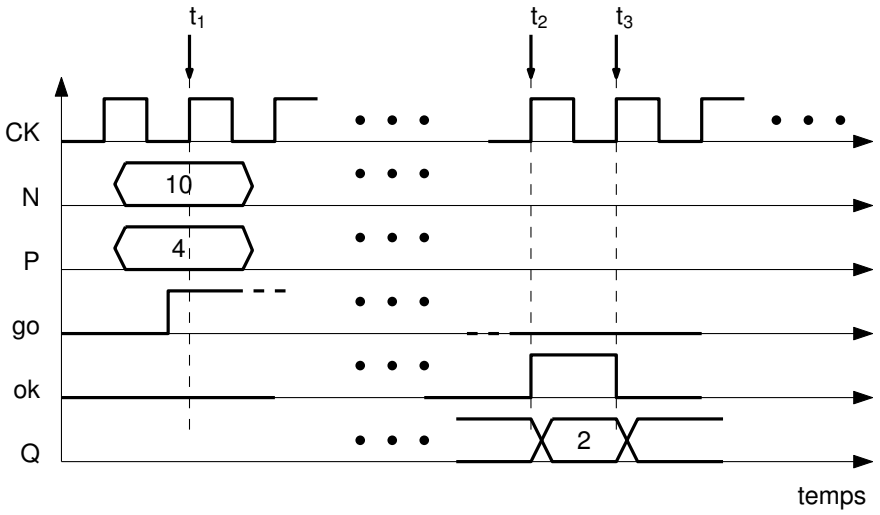


Pour demander une division, le client doit écrire N et P sur les entrées correspondantes, positionner go à 1 et attendre le prochain top d'horloge (instant t_1 sur le chronogramme). Le circuit mémorise alors ces entrées dans ses variables n et p, puis il déroule l'algorithme ci-dessus.

Lorsqu'il a terminé son calcul, le circuit positionne sa sortie ok à 1 (instant t_2) pour indiquer au client qu'il peut lire le résultat sur la sortie Q (d'autres valeurs intermédiaires ont peut-être été visibles sur Q entre-temps, mais elles ne sont pas significatives tant que ok n'est pas à 1).

Le circuit garde ok à 1 jusqu'à ce que le signal go soit retombé à zéro. La sortie Q est valide pendant ce temps-là. Lorsque go est de nouveau à zéro (instant t_3) le circuit retourne dans son état initial et attend une nouvelle commande. (Bien sûr le circuit ne réagit pas immédiatement lorsque go passe à zéro, mais seulement lors du top d'horloge suivant)

Note : Même si le client ne maintient pas go à 1 pendant tout le calcul, et que go est déjà à zéro au moment de t_2, notre circuit garde sa sortie valide pendant au moins un cycle d'horloge. Autrement dit, il y a toujours au moins un cycle d'écart entre les instants t_2 et t_3, comme illustré sur le chronogramme ci-dessous :



En résumé, on peut considérer que notre circuit implémente l'algorithme ci-dessous :

```
pour toujours faire
    attendre que go = 1
    n := entrée N
    p := entrée P
    x := 0
    q := 0
    tant que x+p ⩽ n
        x := x+p
        q := q+1
    fin tant que

    sortie ok := 1
    sortie Q := q
    attendre que go = 0
    sortie ok := 0
fin tant que
```

1.4 Chemin de données

Pour construire un tel circuit, on va essayer de séparer proprement le contrôle des données, comme sur le schéma 2 (p. 4). La partie données vous est fournie : des registres pour les différentes variables, et des circuits combinatoires pour les calculs. Attention, les registres utilisés sont ceux fournis sur le Mooddle du cours (`Reg.dig` ou `Reg8.dig`, ils sont fournis dans l'archive `.zip` fournie). Ils sont synchrones et possède un *reset* et un *enable* . Au moment du top d'horloge :

- si *reset* est vrai, le registre passe à zéro
- sinon, si *enable* (quelqufois appelé *load*) est vrai, le registre mémorise la valeur de D
- sinon, il garde sa valeur précédente.

Les signaux `loadXXX` sont utilisés pour controler le chargement des registres stockant la variable `XXX` (on l'a appelé `enable` précédement). Ces variables permettent de contrôler le moment où l'on change la valeur de la variable : ce n'est pas forcément à chaque cycle. La présence de ces signaux augmente la complexité de l'architecture, mais on verra qu'il est très difficile de concevoir une architecture où tous les registres chargent leur valeur à chaque cycle.
Pour la partie contrôle, ce sera à vous de l'implémenter, mais la partie donnée vous est fournie sur Moodle sous forme d'un composant Digital intitulé `datapath_divider`. À défaut d'un meilleur nom, on appelle *pp* (pour «plus petit») le signal transmis par le comparateur à destination de l'automate.

QUESTION 3 ▶ Mettez des flèches sur tous les signaux qui apparaissent sur la figure 2 pour indiquer dans quel sens ils vont.

QUESTION 4 ▶ Téléchargez le fichier ZIP contenant le composant `divider_datapath`, dezippez le fichier `.zip` cela crée un répertoire `divider-student` qui contient les composants qui vous sont donnés.

QUESTION 5 ▶ Mettez digital en mode simulation et Simuler une division en modifiant les valuersd des pates d'entrée du composants. Par exemple, divisé N=12 par P)=5 : le signal `PP` doit repasser a zéro en même temps que le registre `Q` contient la valeur `2`. Attention, cette question peut être relativement longue, il faut bien comprendre à quel moment activer toutes les entrées à 1 ou 0, c'est ce que devra faire votre contrôleur, il faut donc bien comprendre exactement ce qui doit se passer au cours du temps.

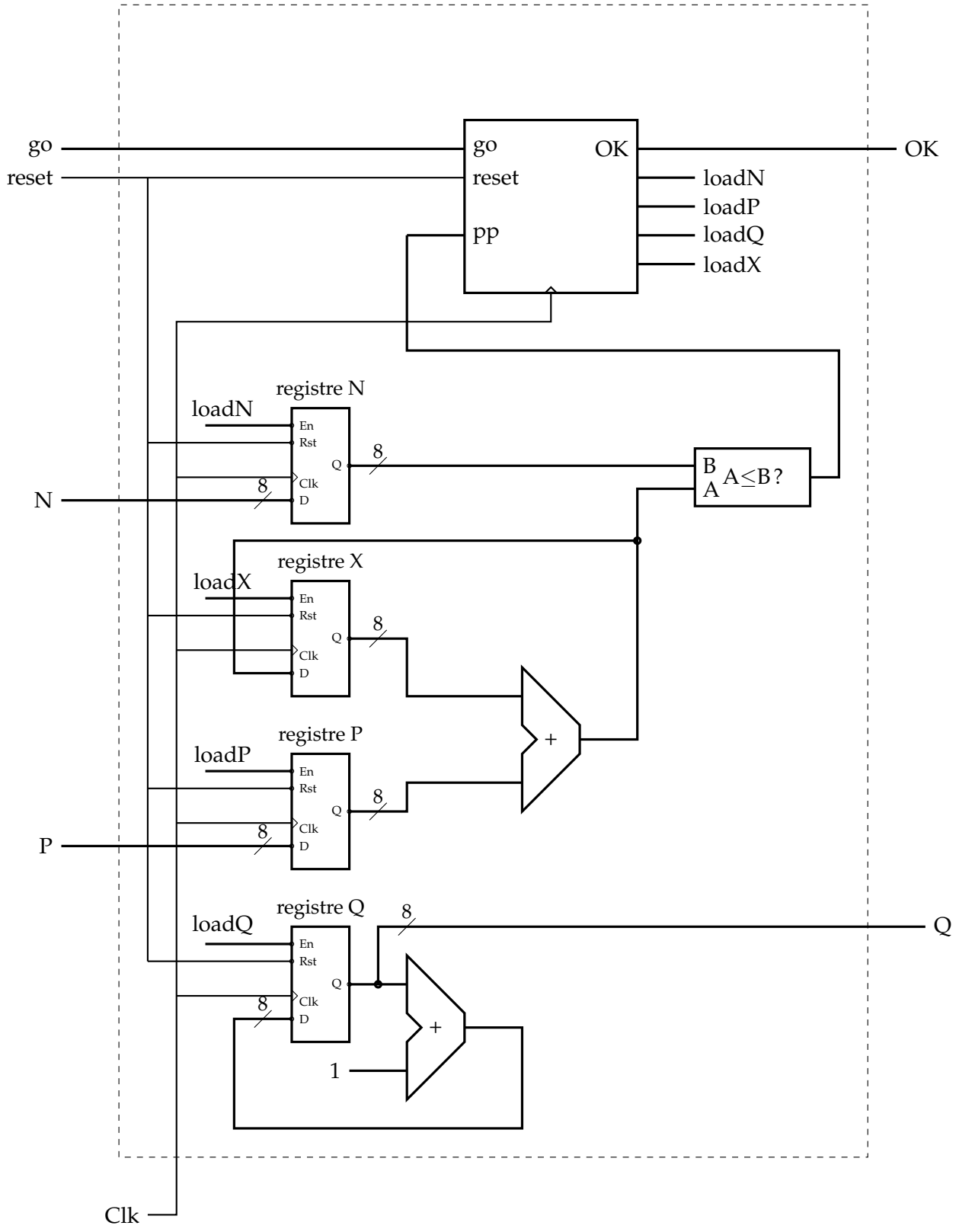


FIGURE 2 – Le circuit du diviseur

1.5 Automate de contrôle

Nous allons maintenant réaliser l'automate qui pilote ce chemin de donnée. Il y a plusieurs automates possibles, en général on utilise un automate de Moore pour la conception de circuits, car les automates de Mealy (i.e dont les sorties changent de manière combinatoire par rapport aux entrées) résultent dans des comportement beaucoup plus compliqués à comprendre. Nous allons donc concevoir un automate sur le modèle de la figure 3.
Même en se limitant à un automate de Moore, il existe plein d'automate qui feront marcher correctement le chemin de données. Le passage de l'algorithme de la division au couple "chemin de donnée/automate" peut être automatisé mais reste complexe, nous allons vous guider pour que tout le monde ait le même automate.

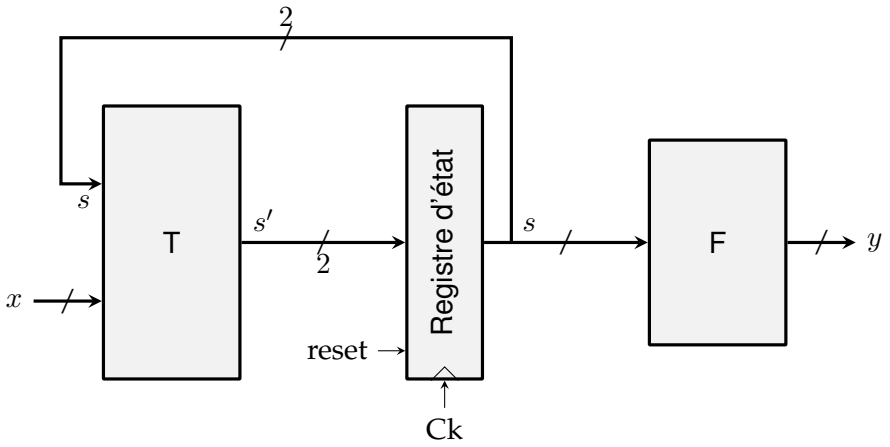


FIGURE 3 – Forme générique d'un automate de Moore : x sont les signaux d'entrée, s est l'état (ici à 2 bits, donc 4 valeurs possibles pour l'état), y sont les signaux de sorties. Le bloc T indique la fonction de transision de l'automate, le bloc F indique la fonction de sortie

QUESTION 6 ▶ Identifiez pour notre automate x et y. On remarque sur la figure 2 qu'il y a une entrée reset sur la figure 2 qui n'était pas sur la figure 1. Ce reset sera synchrone : il nous rammène le registre d'état dans l'état initial.
On va proposer un automate à 4 états : l'état `wait` (état initial, avant qu'on ait eu le "go"), l'état `test` qui fera la comparaison entre `X` et `N`, l'état `increment` qui incrémentera Q si le resultat de la comparaison est X<=N, et l'etat `done` qui est atteint quand la comparaison donne X>N

QUESTION 7 ▶ Dessinez l'automate complet avec :

- sur chaque transition, le symbole d'entrée attendu qui déclenche la transition,
- sur chaque état, la valeur des symboles de sortie émis.

QUESTION 8 ▶ Dessinez le circuit correspondant à cet automate sous la forme représentée en figure 3, sur combien de bits (au minimum) pouvez-vous coder votre état s ?

QUESTION 9 ▶ Donnez les tables de vérité de la fonction de transition et de la fonction de sortie, Pour la table de transition, on notera s l'état courant de l'automate, et s' l'état suivant. On utilisera l'étoile (*) pour indiquer que la valeur d'une entrée n'importe pas. Par exemple, dès que reset vaut 1, la valeur de toutes les autres entrée n'importe pas, l'état suivant est wait (codé 0).

1.6 "Exécution"

QUESTION 10 ▶ Complétez le chronogramme 6, jusqu'à la fin de la division.

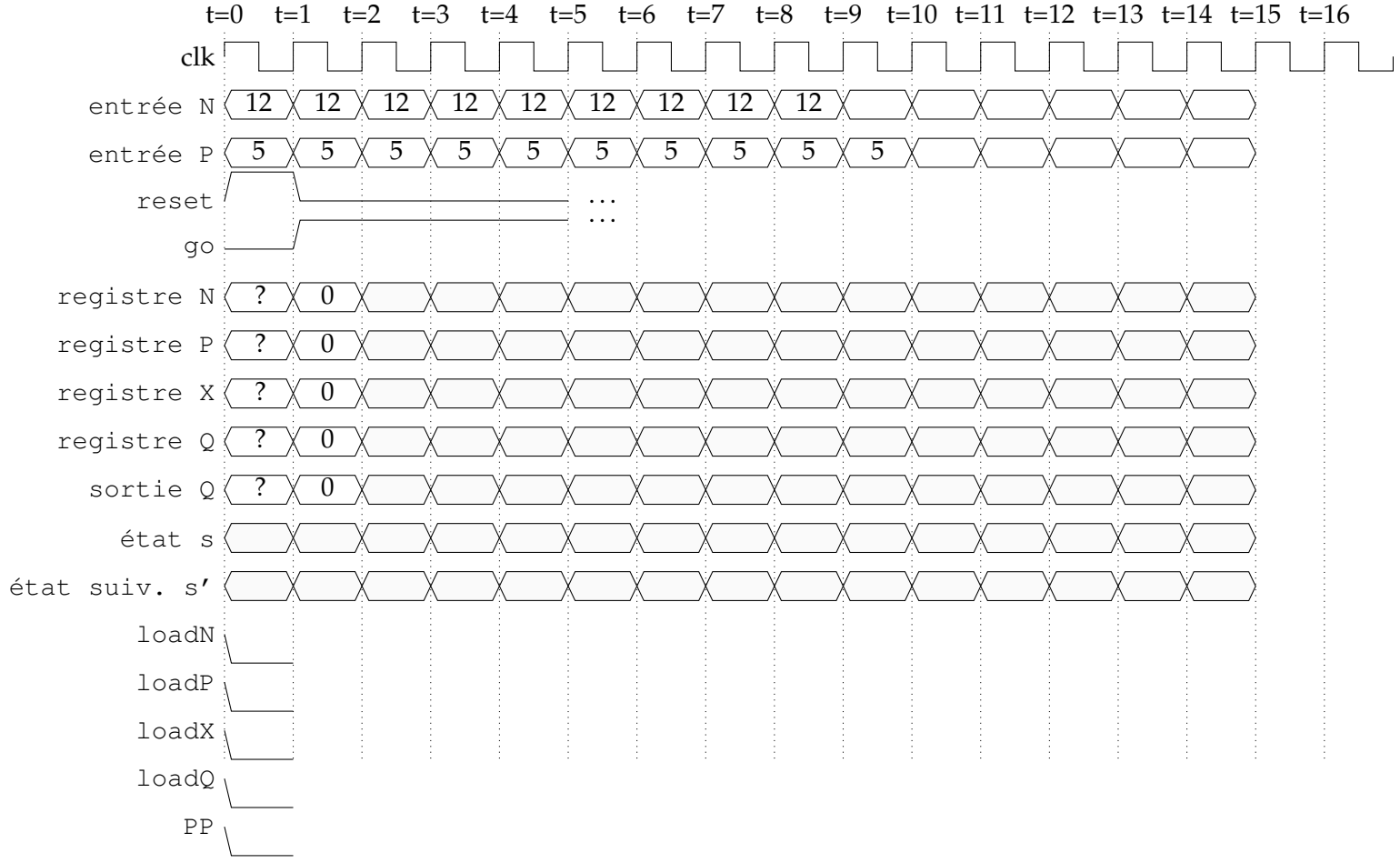
Pour cette question, on va se synchronser pour que tout le monde utilise le mêm codage de l'état : l'état `wait` sera codé 0b00, l'état `test` codé en 01, `increment` codé en 10, et `done` codé 11.

Les premières lignes représentent les données aux entrées et aux sorties du circuit. La ligne verticale pointillée repère le top d'horloge où l'algorithme démarre. Après cet instant, les entrées N et P ne sont plus utiles, leur valeur n'est donc pas importante. Pour le signal go, vous pouvez supposer que le client garde go à 1 jusqu'au bout de l'opération, ou qu'il le remet à zéro plus tôt.

Les lignes suivantes représentent les valeurs des registres du chemin de données. Leur valeur initiale est inconnue (notée « ? » sur le chronogramme).

Les deux lignes suivantes représentent l'état courant de l'automate de contrôle, noté s et l'état suivant calculé par la fonction de transition, noté s'.

Les dernières lignes représentent les signaux de contrôle de de compte-rendu échangés entre la partie contrôle et le chemin de données.



2 En Digital

Dans cette partie, vous allez réaliser votre diviseur dans Digital, comme un assemblage de portes élémentaires pour concevoir l'automate, puis d'un assemblage de composants plus gros que vous avez déjà conçu précédemment, puis le diviseur dans son ensemble.

2.1 Construction de l'automate

On va commencer par construire le circuit implémentant le comportement de l'automate construit à la section 1.5 en s'attaquant d'abord à la fonction de transition puis à sa fonction de sortie. Pensez bien à sauvegarder vos design digital dans le même répetoire ou vous avez sauvergardé le composant `divider_datapath.dig`. Mettez à jour les composant *custom* dès que vous avez terminé un composant, il devient alors disponible pour vos autres designs.

2.1.1 Fonction de transition

La fonction de transition prend pour entrée l'état courant s et les signaux go et pp. Elle produit l'état suivant s'. Si vous avez construit un automate avec 4 états, s' est codé sur 2 bits. La fonction de transition a donc deux bits en sorties, il vous faudra donc construire 2 équations.

QUESTION 11 ► A partir des tables de vérités construites à la section 1.5, donner les équations booléennes pour s' en fonction de s, go et pp.

QUESTION 12 ► A partir des équations booléennes, construisez le circuit correspondant.

2.1.2 Fonction de sortie

QUESTION 13 ► Procédez de la même façon (équations booléennes puis circuits) pour les sorties de votre automates, à savoir : $loadN$, $loadP$, $loadX$, $loadQ$, et ok.

2.2 Construction du diviseur

Il n'y a plus qu'à assembler tout ce beau monde pour reproduire à l'identique la figure 2. C'est tellement gratifiant de voir son diviseur fonctionner :)