



Rapport

Projet VHDL

I4

Novembre 2004

Charles
Sylvain



SOMMAIRE

I. DESCRIPTION DU PROJET.....	3
A. LES ENTREES/SORTIES	3
B. CONTRAINTES ET CARACTERISTIQUES	3
C. DESCRIPTION DU COMPOSANT	4
II. PARTIE LOGIQUE.....	5
A. LE DETECTEUR DE FRONT	6
B. LE COMPTEUR MODULO 30	6
C. LE COMPTEUR DE VITESSE ET LE BLOC « SAUVEGARDE ET FIXATION »	7
III.MACHINE D’ETAT	8
A. PROTOCOLE DE COMMUNICATION	8
B. PRINCIPE DE FONCTIONNEMENT	8
IV.SIMULATION.....	9
A. DESCRIPTION DE LA SIMULATION	9
B. EXEMPLES DE FONCTIONNEMENT	10
V. OPTIMISATION	12
A. SYNTHESE	12
B. OPTIMISATION	13
C. CONTRAINTES DE TEMPS	13
CONCLUSION.....	15
ANNEXE 1 : CODES SOURCES	16

EL411 - VHDL

Rapport de projet

Le but de ce projet de l'unité de VHDL du début de la quatrième année est de mettre en pratique les connaissances acquises pendant un mois et demi. L'exemple concret proposé ici nous permet ainsi de réaliser un circuit, du cahier des charges jusqu'à la simulation, sans oublier l'optimisation du composant dans l'éventualité d'une future implémentation.

I. Description du projet

L'objectif du projet est de déterminer la vitesse d'un moteur par comptage d'impulsions. Un capteur monté sur le stator du moteur émet des impulsions à raison de 30 impulsions par tour. Au bout des 30 impulsions, le système envoie une valeur proportionnelle à la vitesse du moteur.

Pour réaliser ce système, nous avons décidé de créer un compteur qui, à chaque fois que le moteur fait un tour, envoie sa valeur sur la sortie. Le compteur est incrémenté à chaque front d'horloge.

A. Les Entrées/Sorties

Le composant est doté de 4 entrées sur 1 bit :

- I : entrée provenant du moteur sur laquelle on peut relever les impulsions correspondant aux tours du moteur. Impulsions comptées sur front montant.
- Clk ou H : l'horloge du système cadencée à 20 MHz.
- Raz : Remise à zéro du composant. Actif à l'état bas.
- Rq : « Request », ligne provenant d'un système externe informant le composant qu'une valeur de la vitesse veut être lue.

Le composant est doté de 1 sortie sur 1 bit et de 1 sortie sur 21 bits :

- Ack : « Acknowledge », informe le système externe qu'une valeur peut être lue.
- V : bus sur 21 bits représentant la vitesse du moteur.

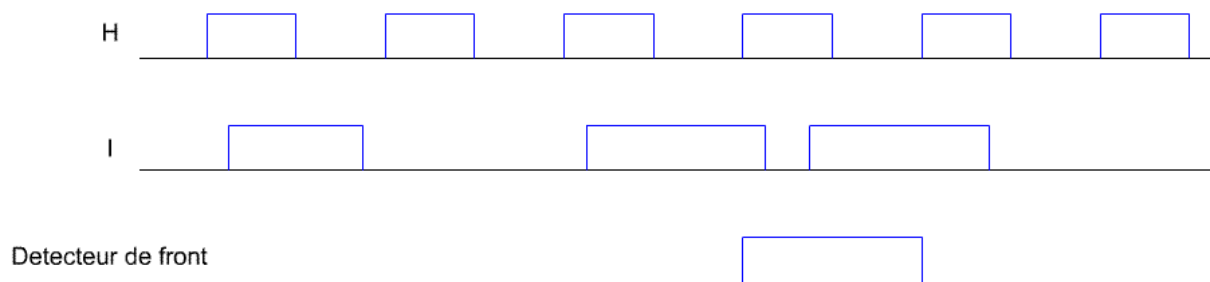
B. Contraintes et Caractéristiques

1. La forme de l'impulsion

Pour que le composant puisse calculer correctement la vitesse du moteur, il faut que les impulsions aient une largeur au moins égale à 40 ns ce qui correspond à la période de

l'horloge. En effet, le front montant de l'impulsion est détecté en se calant sur les fronts d'horloge.

De plus, le temps entre chaque impulsion doit également être égal à 40 ns.



On remarque sur le chronogramme ci-dessus que si l'impulsion est plus courte que 40 ns alors elle ne sera pas détectée et que si deux impulsions se suivent de moins de 40 ns alors, la deuxième impulsion sera ignorée.

2. Vitesse minimum

La vitesse minimum souhaitée est de 20 tours/s. Avec un bus de 21 bits, il est possible de descendre à une vitesse minimum de 12 tours/s.

20 tours par secondes => 1 tour toutes les 50 millisecondes.

La vitesse de l'horloge étant de 25 MHz, en 50 ms, le compteur a atteint la valeur de 1 250 000. Il faut donc un bus d'au minimum 21 bits pour atteindre cette valeur.

Avec 21 bits, on peut même compter jusqu'à 2 097 152, ce qui représente 0,08 secondes par tour, c'est à dire 12 tours par secondes.

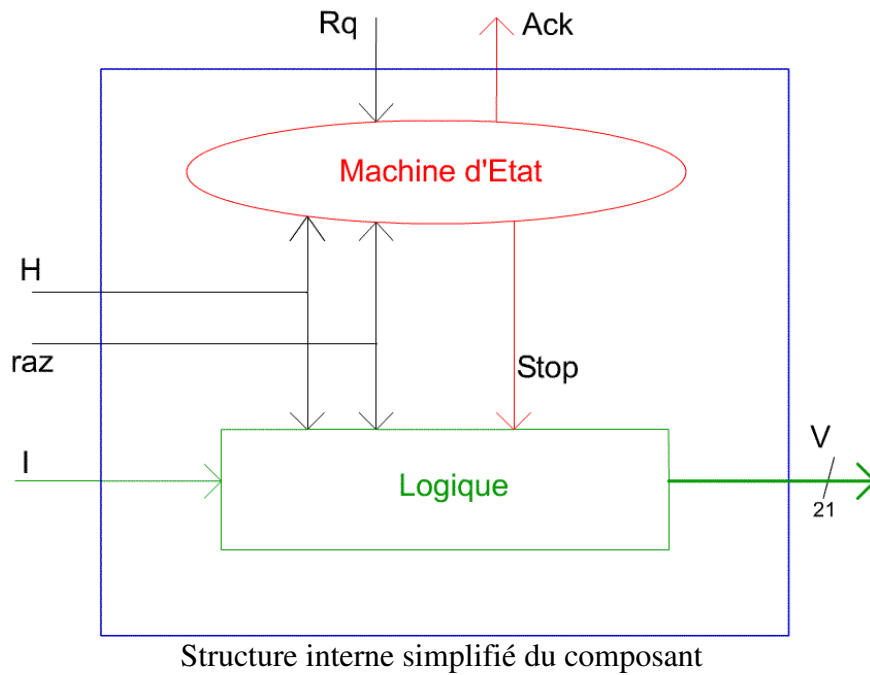
3. Vitesse maximum

La vitesse maximum du moteur pouvant être calculée est directement liée avec la forme de l'impulsion. La durée d'une impulsion étant au minimum de 40 ns et le temps entre chaque impulsion étant également de 40 ns, au maximum, on peut avoir 1 tour toutes les 2 400 ns, ce qui correspond à 416 666 tours par seconde. Physiquement, ce résultat n'a pas beaucoup d'importance tellement sa valeur est grande mais cela montre que la fréquence de l'horloge est largement suffisante pour calculer la vitesse de rotation de moteurs très rapides.

C. Description du composant

Le système se sépare en en deux parties très distinctes, une partie logique, qui génère la valeur correspondant à la vitesse du moteur et une partie machine d'état qui gère l'interface avec le système externe.

Le seul lien qu'il y ait entre la machine d'état et la partie logique est le signal Stop qui permet de bloquer la valeur de sortie V pendant sa lecture par le système externe.

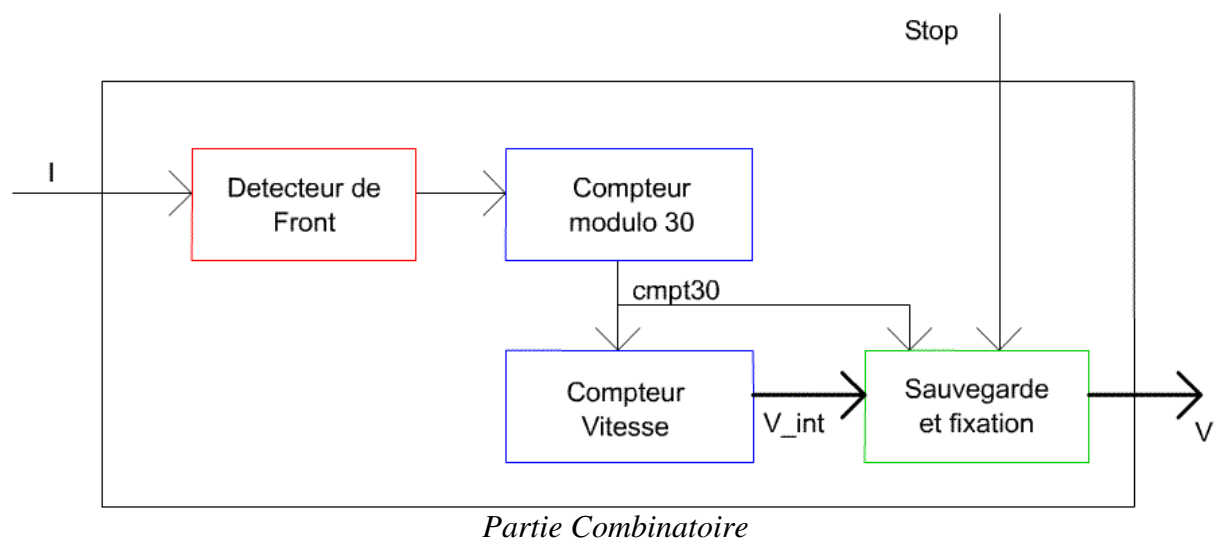


La valeur de V est mise à jour à chaque fois que le moteur fait 1 tour si Stop = 0, sinon, dès que Stop repasse à l'état bas.

II. Partie Logique

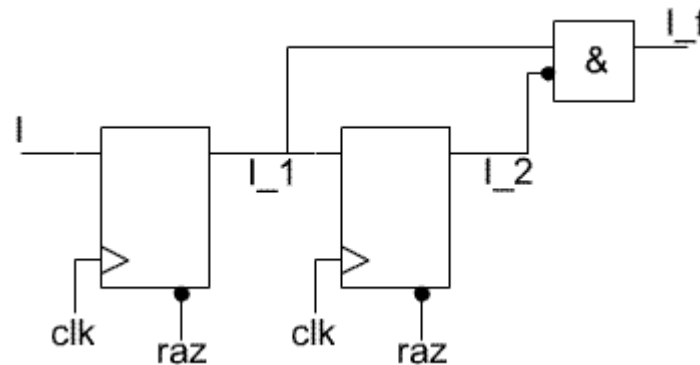
La partie combinatoire est composée de 4 blocs distincts :

- Un détecteur de front montant pour détecter les impulsions en provenance du moteur
- Un compteur modulo 30 qui compte le nombre d'impulsions du moteur
- Un compteur sur 21 bits correspondant à la vitesse du moteur et remis à zéro lorsque le moteur a fait 1 tours.
- Un bloc qui sauvegarde et fige la valeur de la vitesse sur le bus V



A. Le détecteur de front

Voici le schéma logique du détecteur de front montant.



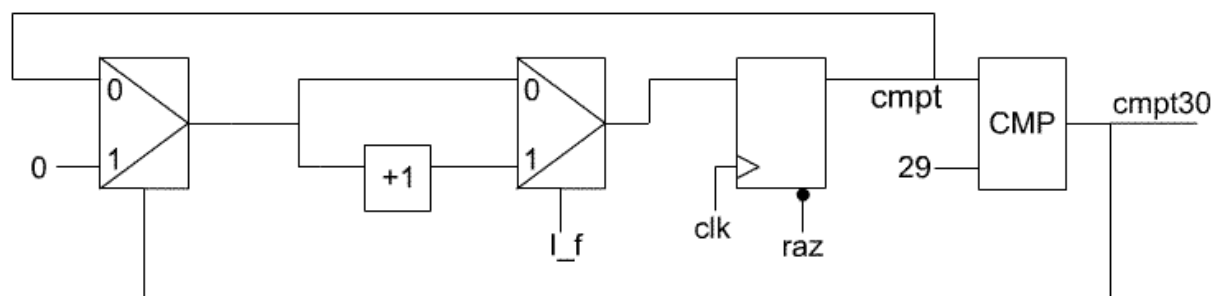
Détecteur de front montant

Le principe du détecteur de front est de comparer deux instants successifs (par rapport à l'horloge) et de comparer si le signal est passé de l'état bas à haut. Ainsi, lorsqu'un front est détecté, la sortie I_f passe à 1 pendant 1 cycle d'horloge. I_f est ensuite branché sur le compteur qui comptabilise le nombre de front détecté.

B. Le compteur modulo 30

Le compteur modulo 30 a pour rôle de compter 30 impulsions venant du moteur par l'intermédiaire de I_f . Une fois les 30 impulsions atteintes, le compteur génère un signal $cmpt30$ pendant une période de l'horloge qui indique au compteur de vitesse de sauvegarder sa valeur et de recommencer à zéro.

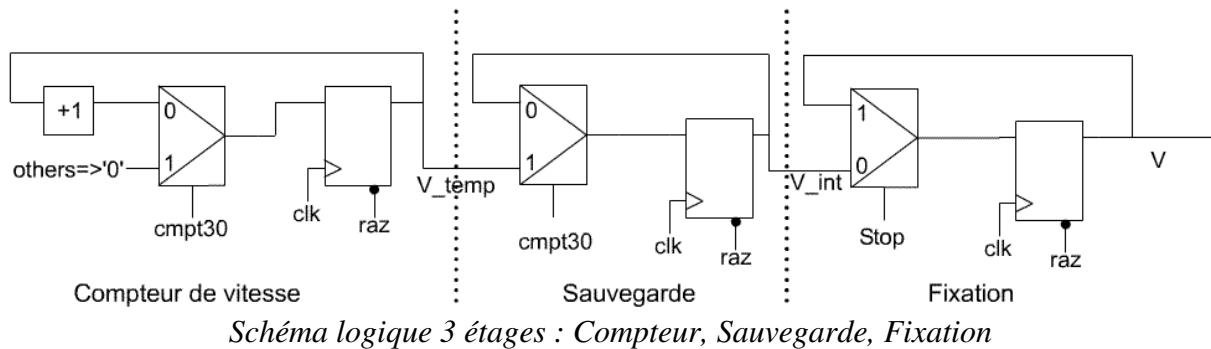
Voici le schéma du compteur modulo 30 :



Compteur modulo 30

Le bloc *CMP* est un comparateur qui teste si $cmpt$ est égal à 29. Si c'est le cas, $cmpt30$ est généré et le compteur de vitesse remis à zéro.

C. Le compteur de vitesse et le bloc « sauvegarde et fixation »

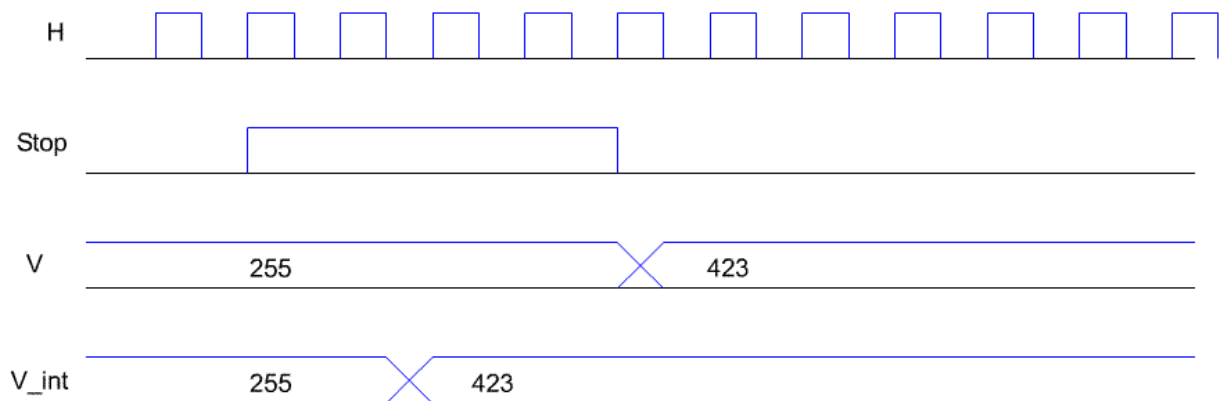


Le compteur de vitesse est un simple compteur qui s'incrémente à chaque top d'horloge et qui est remis à zéro, soit par le *raz* global du circuit, soit par *cmpt30* provenant du compteur modulo 30.

Ensuite, il est nécessaire de faire un étage de sauvegarde de la valeur de la vitesse à chaque fois que le moteur fait 1 tour. C'est un simple multiplexeur avec une bascule qui permet de faire cet étage pour chaque bit du registre.

Enfin, l'étage de fixation permet de fixer la valeur de sortie pendant que le système externe la lit. Cette fixation se fait grâce au signal *Stop* qui est activé par la machine d'état. Ainsi, lorsque le système externe demande une lecture avec la ligne *Rq* (Request), la machine d'état bloque la sortie avec le signal *Stop* puis autorise la lecture avec la ligne *Ack* (Acknowledge).

Le système a également été réalisé de manière à ce que, si la valeur de *V_int* change alors que *Stop* est activé, *V* ne change que lorsque *Stop* repasse à 0.



Chronogramme schématique

Sur le chronogramme ci-dessus, on peut voir que le *V* n'est mis à jour que lorsque *Stop* repasse à zéro.

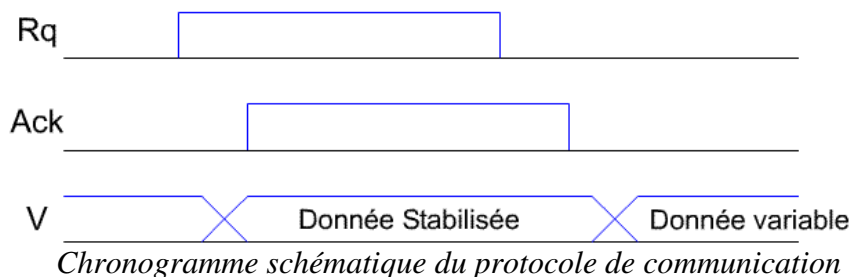
Cependant, on aurait pu faire abstraction de l'étage de sauvegarde. On pourrait ainsi économiser, sachant que *V* est un bus de 21 bits, 21 bascules D et 21 multiplexeurs 2 vers 1. Le problème de cette solution est que si *V_temp* change pendant que la fixation est bloquée par *Stop*, alors la valeur est perdue et il faut donc attendre 1 tour supplémentaire pour avoir la nouvelle valeur. Le choix de la solution dépend donc de la politique adoptée, c'est à dire si l'on veut un système qui prend très peu de place ou si l'on veut un système qui réagit très vite. La solution pourrait dépendre également de la vitesse du moteur. Si le moteur est très lent, la solution à 3 étages paraît la plus adaptée. Si le moteur est très rapide, on peut considérer qu'attendre 1 tour est négligeable et donc privilégier la solution à 2 étages.

III. Machine d'Etat

A. Protocole de communication

Pour communiquer notre vitesse V au système externe qui n'utilise pas la même horloge, nous utilisons un protocole de communication asynchrone nommé handshaking. Le fonctionnement de ce système est le suivant, lorsque le système externe veut lire la valeur de V , il envoie une requête en mettant la ligne Rq (Request) du composant à l'état haut. Notre composant détecte cette requête et fixe la valeur de V pour s'assurer que cette valeur ne sera pas modifier avant la fin de la lecture, puis envoie l'autorisation de lecture via la ligne Ack (Acknowledge). Quand le système externe a fini de lire il remet la ligne Request à 0. Enfin lorsque notre composant a détecté la fin de la lecture, il remet Acknowledge à 0 pour indiquer qu'il est prêt à accepter une nouvelle requête.

Ce principe de fonctionnement peut être résumé par le chronogramme suivant :



La solution retenue pour réaliser ce protocole de communication est d'utiliser une machine d'état qui pourra ainsi gérer les lignes Request, Acknowledge et le signal Stop pour figer la valeur de V .

B. Principe de fonctionnement

La machine d'état gère 3 signaux :

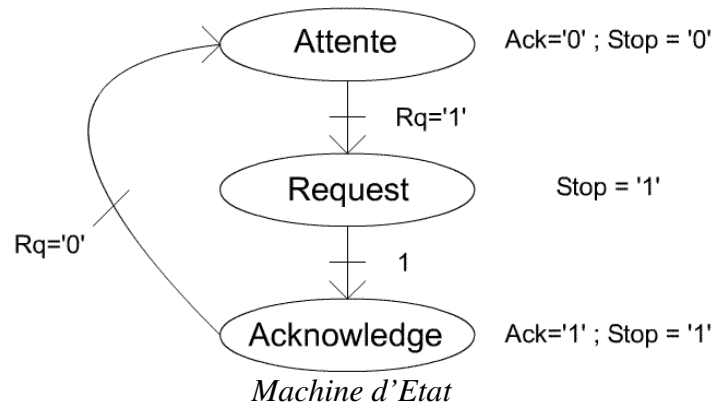
- Request : Entrée indiquant au composant qu'une valeur veut être lue.
- Acknowledge : Sortie indiquant au système externe qu'une valeur peut être lue
- Stop : Signal interne permettant d'assurer la stabilité de la sortie V lors de sa lecture (actif haut).

Pour effectuer la gestion du protocole de communication, nous avons besoin de 3 états :

- Attente : C'est l'état initial, c'est à dire qu'il n'y a aucune requête en cours, donc le signal Acknowledge est à 0. De plus lorsque l'on est dans cet état, la valeur de V peut changer, donc le signal Stop est à 0. Lorsqu'une requête est détectée, la machine d'état passe dans l'état suivant Request.
- Request : C'est l'état qui met en place les conditions nécessaires à la lecture de V par le système externe, c'est à dire que l'on garantit la stabilité de V en mettant Stop à 1. La ligne Acknowledge reste 0 pour laisser le temps à V de se stabiliser. La machine d'état passe ensuite systématiquement à l'état suivant, Acknowledge
- Acknowledge : La machine d'état continue à assurer la stabilité de V en gardant Stop à 1 et la ligne Acknowledge passe à 1 pour indiquer que la lecture est possible. Le

passage à l'état suivant se fait lors de la détection du passage de la ligne Request à 0 ce qui indique la fin de la lecture. La machine d'état retourne donc dans l'état Attente.

Voici la représentation de la machine d'état :



Attention :

- Le protocole est asynchrone mais cela ne veut pas dire que notre machine d'état est asynchrone, donc tous les changements d'états se font sur front montant d'horloge.
- La machine d'état comporte un reset asynchrone qui la met dans l'état Attente.

IV. Simulation

A. Description de la simulation

Afin de tester la description du projet, nous avons simulé le fonctionnement du système externe grâce à un fichier VHDL de simulation Sim.vhd. Ce fichier nous permet de simuler le comportement de Mvitesse (l'entité de notre projet) en réponse aux signaux extérieurs Request, Impulsion, Horloge, et RAZ. Ces signaux sont décrits manuellement pour simuler un fonctionnement réel, par exemple l'horloge est décrite comme un signal carré de demi-période 40ns (fréquence de 25Mhz).

```
clk <= not clk after 20 ns;
```

De même nous avons considéré que les impulsions générées par le détecteur du moteur pouvaient être décrites sous la forme d'un signal carré. La période de ce signal pouvant être changée dans les limites spécifiées par le cahier des charges. Ici nous avons pris 63 ns pour une demi-période.

```
I <= not I after 63 ns;
```

Nous simulons le fonctionnement du système externe en mettant le signal Rq aux valeurs voulues : 1 si une lecture de la vitesse est demandée, 0 sinon.

Pour observer le comportement du circuit nous observons les chronogrammes des signaux externes (clk, acknowledge, impulsion, raz, request, vitesse), mais nous observons aussi certains signaux internes qui nous permettent de suivre plus facilement le déroulement de la simulation, par exemple la sortie des compteurs ou encore les états de la machine d'état.

B. Exemples de fonctionnement

- Premier chronogramme (cf page suivante)

La ligne *I* représente les impulsions envoyées par le moteur, *V* est la sortie du composant représentant la vitesse. Vers 3700 ns le moteur a effectué son premier tour car le compteur d'impulsion arrive à 29. *cmpt30* passe donc à 1 pendant un cycle d'horloge afin d'indiquer qu'il faut afficher la nouvelle valeur de *V*. *V_int* qui sauvegarde la valeur de la vitesse est mis à la nouvelle valeur de la vitesse, c'est à dire la valeur du compteur *V_temp*. Au coup d'horloge suivant *V* affiche la nouvelle valeur de la vitesse qui est 93 dans notre cas. Les compteurs de vitesse et d'impulsions sont remis à zéro. On peut vérifier que le compteur modulo 30 est bien incrémenté à chaque impulsion *I*. De même le compteur qui indique la vitesse par tour est compte à chaque front montant de l'horloge.

Pendant ce temps la machine d'état est restée à l'état d'attente car il n'y a pas eu de demande de lecture. On simule à 4000 ns un *Request* du système extérieur. Deux coups d'horloge plus tard (le temps de certifier que *V* est fixe en passant par l'état *Request* de notre machine d'état), *Ack* passe à 1 pour indiquer qu'une lecture est possible. Lorsque la lecture a été effectuée (*Rq* passe à zéro), *Ack* repasse à zéro pour indiquer au système extérieur qu'il peut à nouveau demander une lecture.

- Deuxième chronogramme

Pour vérifier que *V* est bien bloqué lorsqu'il y a une lecture on ré effectue une simulation mais en demandant une lecture (c'est à dire *Request* à 1) avant la fin du premier tour du moteur (passage de 0 à 106 de *V*). La valeur de *V* doit rester fixe et ne changer qu'à la fin du Request en prenant la nouvelle valeur. L'étage de sauvegarde de la partie logique n'est utile que dans cette éventualité.

On remarque effectivement que sur le chronogramme, à la fin du premier tour du moteur vers 3700 ns, *V* ne change pas. En effet sa valeur est garantie stable car il y a une lecture en cours. *Stop* est à 1 et bloque le changement de *V*. On peut observer que *V_int* a pris la nouvelle valeur de la vitesse mais qu'elle n'est pas affectée à *V*. Dès que la lecture est terminée, *Stop* repasse à 0 et *V* prend la nouvelle valeur de la vitesse qui avait été sauvegardée dans *V_int*. Le fonctionnement est conforme à nos attentes.

V. Optimisation

Le logiciel BuildGates permet une optimisation des circuits décrits en VHDL pour un gain de place et/ou de vitesse du circuit une fois implémentés.

Le rapport de l'optimisation permet dans certains cas de détecter des erreurs de conception pour un circuit. La présence de latch non voulues par exemple, ou d'un nombre plus important de bascules que prévu indiquent des erreurs non détectées à la compilation.

Pour vérifier que le nombre de bascules correspond à notre description du système, nous avons calculé manuellement le nombre de bascules prévues pour notre circuit :

Machine d'état : 2 bascules (Pour avoir 3 états).

Process détecteur de fronts : 2 bascules.

Process Compteur modulo 30 : 5 bascules pour le compteur, 1 pour le signal cmpt30 donc 6 bascules.

Process Compteur de tours : 21 bascules car il y a une bascule par bits du bus.

Process Sauvegarde du nombre de tours : 21 bascules.

Process Sortie : 21 bascules.

Ce qui fait un total de 73 bascules.

A. Synthèse

La commande *do_build_generic* fait la synthèse du VHDL et donne dans le rapport les éléments séquentiels du circuit.

Sequential Elements									
Module	File	Line	Name	Type	Width	AS	AR	SS	SR
unit1	/nfs/user/eleve/i4 /cuenods/VHDL/unit 1.vhd	26	I_1_reg	FF	1	N	Y	N	N
unit1	/nfs/user/eleve/i4 /cuenods/VHDL/unit 1.vhd	26	I_2_reg	FF	1	N	Y	N	N
unit1	/nfs/user/eleve/i4 /cuenods/VHDL/unit 1.vhd	37	cmpt30_reg	FF	1	N	Y	N	N
unit1	/nfs/user/eleve/i4 /cuenods/VHDL/unit 1.vhd	37	cmpt_reg	FF	5	N	Y	N	N
unit1	/nfs/user/eleve/i4 /cuenods/VHDL/unit 1.vhd	51	V_temp_reg	FF	21	N	Y	N	N
unit1	/nfs/user/eleve/i4 /cuenods/VHDL/unit 1.vhd	62	V_int_reg	FF	21	N	Y	N	N
unit1	/nfs/user/eleve/i4 /cuenods/VHDL/unit 1.vhd	72	V_reg	FF	21	N	Y	N	N

Sequential Elements									
Module	File	Line	Name	Type	Width	AS	AR	SS	SR
mde	/nfs/user/eleve/i4	34	present_reg	FF	2	N	Y	N	N
	/cuenods/VHDL/mde.								
	vhd								

Le total des bascules utilisées correspond au calcul manuel précédent de 73 bascules. Bien que ceci ne garantisse pas forcément un fonctionnement correct, cela permet de faire une vérification de notre description et le résultat est conforme à nos attentes. De plus, on remarque dans la colonne *Type* qu'il n'y a pas de bascules Latch.

B. Optimisation

Ensuite nous réalisons l'optimisation de notre circuit avec les paramètres par défaut pour qu'il prenne le moins de surface possible et ait une vitesse de fonctionnement satisfaisante. Le résultat la commande *do_optimize* donne la taille prise par le composant, mais ne calcul pas le temps de propagation maximum (*worst slack*) car nous n'avons pas encore mis de contrainte de temps.

Mvitesse					
Cell area	Total area	Worst slack	Local slack	CPU(s)	Mem(M)
913.50	913.50	+INF		29	60

La commande *report_area* permet d'obtenir le détail des surfaces en fonction des blocs.

Module	Wireload	Cell Area	Net Area	Total Area
Mvitesse	B0X0	913.50	0.00	913.50
mde	B0X0	26.00	0.00	26.00
unit1	B0X0	887.50	0.00	887.50
AWDP_INC_0	B0X0	99.00	0.00	99.00

C. Contraintes de temps

Nous définissons des contraintes temporelles pour s'assurer que notre circuit aura un temps de réponse suffisamment rapide. Pour cela nous définissons notre horloge à une période de 40 ns comme précisé dans le cahier des charges. Nous définissons aussi un temps de délai par rapport à l'horloge de 5 ns pour les entrées et de 25 ns pour les sorties afin de s'assurer que le circuit sera fonctionnel au maximum avec ces délais. Les valeurs sont choisies pour être cohérentes avec la fréquence de l'horloge, mais le cahier des charges ne précisant pas ces paramètres, les valeurs ne sont qu'indicatives.

```
set_input_delay -clock horloge 5 {Rq I}
set_external_delay -clock horloge 25 {V Ack}
```

Avec les contraintes temporelles, lorsque l'on refait l'optimisation, le résultat donne en plus de la surface le temps maximum de propagation. Ici la surface ne varie pas avec l'optimisation précédente car les contraintes de temps ne sont pas très élevées. Le temps de *worst slack* est de 14,0689 ns ce qui est largement inférieur à la période de l'horloge de 40 ns, donc il n'y a pas de conflit.

Mvitesse					
Cell area	Total area	Worst slack	Local slack	CPU(s)	Mem(M)
913.50	913.50	14.0689		83	65

Conclusion

Le projet présenté a été fini dans les temps et répond au cahier des charges. La simulation montre un fonctionnement correct du circuit et l'optimisation ne révèle aucune erreur.

Cependant, même si tout semble fonctionner correctement, rien ne nous assure que le programme une fois implémenté dans un FPGA fonctionnera également. Pour cela, il aurait fallu, dans la continuité de notre simulation, faire une simulation post-fit qui prend en compte les temps de propagation et de commutation des bascules (dépendante du composant choisis) et enfin implémenter réellement le programme dans un FPGA pour effectuer des tests. Malheureusement, le temps imparti pour ce projet ne nous a pas permis d'arriver jusqu'à la partie finale du développement d'un circuit.

ANNEXE 1 : CODES SOURCES

- Entité principale

```

library ieee;
use ieee.std_logic_1164.all;

entity Mvitesse is
    port (
        --Entrees
        clk : in std_logic;
        Raz : in std_logic;
        I : in std_logic;
        Rq : in std_logic;
        --Sorties
        V : out std_logic_vector(20 downto 0);
        Ack : out std_logic);
begin
end entity;

architecture cool of Mvitesse is
    signal      Stop : std_logic; --Bloque la sortie
begin

-- Machine d'état
    mde : entity work.mde(cool)
        port map (clk=>clk, Raz=>Raz, Rq=>Rq, Ack=>Ack, Stop=>Stop);

-- Génération Vitesse
    cmpts : entity work.unit1(cool)
        port map (clk=>clk, Raz=>Raz, I=>I, V=>V, Stop=>Stop);

end architecture;

```

- Machine d'état

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity mde is
    port (
        --Entrees
        clk : in std_logic;
        Raz : in std_logic;
        Rq : in std_logic;
        --Sorties
        Ack : out std_logic;
        Stop : out std_logic); --Bloque la sortie
end entity mde;

architecture cool of mde is
    type state is (Attente, Request, Acknowledge);
    signal present, futur : state;
begin

```

```

evolution: process(present, Rq)
begin
case present is
when Attente => futur<=Attente;
    if Rq='1' then futur<=Request;
    end if;
when Request => futur<=Acknowledge;
when Acknowledge => futur<=Acknowledge;
    if Rq='0' then futur<=Attente;
    end if;
end case;
end process evolution;

synchro: process(clk, Raz)
begin
if Raz = '0' then present <= Attente;
elsif rising_edge(clk) then present<=futur;
end if;
end process synchro;

action: process(present)
begin
Stop<='0';
Ack<='0';
case present is
when Attente => Stop<='0'; Ack<='0';
when Request => Stop<='1'; -- Fixe la sortie
when Acknowledge => Stop<='1'; Ack<='1';
end case;
end process action;

end architecture cool;

```

- Bloc Combinatoire

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity unit1 is
    port (
        --Entrees
        I : in std_logic;
        clk : in std_logic;
        Raz : in std_logic;
        Stop : in std_logic; --En provenance du graphe d'etat
        --Sorties
        V : out std_logic_vector(20 downto 0));
end entity unit1;

architecture cool of unit1 is
    signal cmpt30 : std_logic;
    signal I_f, I_1, I_2 : std_logic;
    signal V_temp, V_int : std_logic_vector(V'range);
    signal cmpt : natural range 0 to 30;
begin

```

```
I_f <= I_1 and not(I_2);

-- process detecteur de front
process(Raz, clk)
begin
  if Raz='0' then I_1 <= '0'; I_2 <= '0';
  elsif rising_edge(clk) then
    I_2<=I_1;
    I_1<=I;
  end if;
end process;

-- process du compteur modulo 30
process(Raz, clk) is
begin
  if Raz='0' then cmpt <= 0; cmpt30<= '0';
  elsif rising_edge(clk) then
    if cmpt = 29 then
      cmpt30 <= '1';
      cmpt <= 0;
    elsif I_f = '1' then cmpt <= cmpt+1; cmpt30 <= '0';
    else cmpt <= cmpt; cmpt30 <= '0';
    end if;
  end if;
end process;

-- process du compteur de tour
process(Raz, clk) is
begin
  if Raz='0' then V_temp<=(others=>'0');
  elsif rising_edge(clk) then
    if cmpt30='1' then V_temp <= (others=>'0');
    else V_temp <= std_logic_vector(unsigned(V_temp)+1);
    end if;
  end if;
end process;

-- process sauvegarde du nb de tour
-- on aurait pu grouper ce process avec le compteur de tour
-- mais pour un soucis de clareté, nous les avons séparés.
process(Raz, clk) is
begin
  if Raz='0' then V_int<=(others=>'0');
  elsif rising_edge(clk) then
    if cmpt30='1' then
      V_int<=V_temp;
    end if;
  end if;
end process;

-- fixation de la valeur
process(Raz, clk) is
begin
  if Raz='0' then V<=(others=>'0');
  elsif rising_edge(clk) then
    if Stop='0' then
      V<=V_int;
    end if;
  end if;
end process;
```

```
end architecture cool;
```

- Fichier de simulation

```
library ieee;
use ieee.std_logic_1164.all;

entity Sim is
begin
end entity;

architecture cool of Sim is
--Entrees fictives
signal clk : std_logic := '0';
signal Raz : std_logic;
signal I : std_logic := '0';
signal Rq : std_logic := '0';
--Sorties fictives
signal V : std_logic_vector(20 downto 0);
signal Ack : std_logic;
begin

    Mvitesse : entity work.Mvitesse(cool)
        port map (clk=>clk, Raz=>Raz, Rq=>Rq, Ack=>Ack, V=>V, I=>I);

    clk <= not clk after 20 ns;
    Raz <= '0', '1' after 15 ns;
    I <= not I after 63 ns;
    Rq <= '1' after 4000 ns, '0' after 4100 ns;

end architecture;
```