
SETR-E2 Conception de circuits

Document de conception contrôleur d'interruptions

Louison Gouy et Mathis Briard
December 18, 2022



ÉCOLE POLYTECH DE NANTES
ELECTRONIQUE ET TECHNOLOGIE NUMÉRIQUE

Enseignant référent : Sébastien LE NOURS

Contents

1	Introduction	5
1.1	Contextualisation	5
2	Cahier des charges	6
2.1	Objectif du circuit	6
2.2	Fonctionnalités attendues	6
2.3	Utilisation du circuit	6
2.4	Chronogrammes caractéristiques	9
2.5	Contraintes du projet	11
3	Spécifications	12
3.1	Caractérisation de l'environnement	12
3.2	Entrées et sorties du composant	13
3.3	Spécifications fonctionnelles	15
3.4	Spécification des registres	16
3.5	Écriture des procédures de base	19
3.6	Spécifications opératoires	20
3.6.1	Modalités de vérification du circuit	20
3.7	Spécifications technologiques	21
3.7.1	Contraintes de répartition	21
3.7.2	Contraintes technologiques	21
4	Conception	22
4.1	Délimitation fonctionnelle	22
4.2	Décomposition fonctionnelle et introduction des interfaces	22
4.3	Raffinage du bloc <code>Interface</code>	23
4.4	Raffinage du bloc <code>Traitement</code>	24
4.5	Écriture des algorithmes	25
4.5.1	Algorithme <code>Interface_Write</code>	25
4.5.2	Algorithme <code>Interface_Read</code>	26
4.5.3	Algorithme <code>Masquage</code>	27
4.5.4	Algorithme <code>Solutionneur de priorité</code>	27
4.5.5	Algorithme <code>Gestion adresse de branchement</code>	28
5	Validation	29
5.1	Test unitaire	29
5.1.1	Interface Write	29
5.1.2	Interface Read	30
5.1.3	Solutionneur de priorités	31
5.1.4	Masque	31
5.1.5	Gestion branchement (blx)	32
5.2	Test intégration	32
5.2.1	Configuration	32
5.2.2	Interruptions simultanées	34
6	Conclusion	35
	Acronyms	36

List of Figures

1	Schéma de câblage de l'IP à concevoir, du contrôleur mémoire et du processeur	6
2	Entrées et sorties de l'IP à concevoir	7
3	Chronogramme spécification du bus	9
4	Chronogramme spécification signal IT au processeur	10
5	Chronogramme spécification 2 signaux IT cas 1	10
6	Chronogramme spécification 2 signaux IT cas 2	11
7	Planification du projet avec ses jalons	11
8	Comportement du système à microprocesseur	12
9	Comportement de l'entité source d'interruptions	13
10	Entrées et sorties du circuit à concevoir	13
11	Automate du contrôleur d'interruptions	16
12	Caractéristiques du FPGA Zynq 7000	21
13	Délimitation fonctionnelle du contrôleur d'interruptions	22
14	Décomposition fonctionnelle et introduction des interfaces	22
15	Raffinage et décomposition du bloc Interface	23
16	Raffinage et décomposition du bloc Traitement	24
17	Exemple de cycle en V	29
18	Chronogramme validation de l'interface write	30
19	Chronogramme validation de l'interface read	30
20	Chronogramme validation solutionneur priorité	31
21	Chronogramme validation bloc masquage	31
22	Chronogramme validation bloc gestion branchement	32
23	Chronogramme validation globale configuration	33
24	Chronogramme validation globale interruptions simultanées	34

List of Tables

1	Sens et rôle des signaux	8
2	Sens et rôle des signaux	13
3	Sous-relations et type de données	14
4	Synthèse des spécifications fonctionnelles	15
5	Vecteurs d'interruptions	16
6	Organisation des registres du contrôleur d'interruption (vue globale)	17
7	Détails du registre CTRL_IT_EN	18
8	Détails du registre CTRL_IT_PEND	18
9	Détails du registre CTRL_IT_BRA	18
10	Détails du registre CTRL_IT_ADDR_N	18
11	Détails du registre CTRL_IT_PRIO_N_N+1	19

Résumé

La méthodologie de conception de systèmes électroniques est une discipline essentielle lorsqu'il s'agit, dans le cadre d'une future carrière d'ingénieur, de réaliser un produit répondant aux besoins d'un client. Et ce domaine est d'autant plus indispensable dans un monde où les systèmes numériques sont de plus en plus complexes et leur conceptions demandent une bonne structuration et une réelle méthodologie. Ce rapport s'inscrit dans la présentation d'une conception d'un contrôleur d'interruption par l'utilisation d'une méthode précise, applicable dans un ensemble de secteurs d'activités comme l'automobile, le médical, l'aéronautique, maritime, que ce soit dans le civil ou le militaire.

1 Introduction

1.1 Contextualisation

La formation ETN (Électronique et technologies numériques) option SETR (Systèmes embarqués temps-réel) offerte par l'école polytechnique de l'Université de Nantes propose d'aborder diverses branches de l'électronique, des systèmes temps-réel aux systèmes à microprocesseur en passant par la conception de SoC (système sur puce). Cet ensemble de domaines techniques nécessite des compétences en matière de méthodologie de conception. Ce rapport s'inscrit dans la conception d'un contrôleur d'interruptions avec la méthode MCSE. La méthode MCSE (Méthode de conception des systèmes électroniques), née à Ireste par l'impulsion de Jean-Paul Calvez, cette méthode a été implantée au sein d'un outil nommée CoFluent rachetée par Intel® depuis 2011. Cette méthode fait désormais partie de la culture de la formation et constitue l'outil de conception premier de l'ingénieur ETN. Ce rapport se décompose en diverses parties. Il s'agira dans un premier temps de rappeler le cahier des charges de la conception de l'IP contrôleur d'interruptions. Pour l'ensemble de ce rapport, les diverses phases de spécifications et conceptions s'appuient sur les deux ouvrages de Jean-Paul Calvez. [1] [2]

Ce travail de conception a pour but de placer les étudiants dans un contexte industriel. Le cahier des charges fournit prend la forme d'un exemple réel où les spécifications du client sont exprimées.

2 Cahier des charges

Cette partie présente le cahier des charges du périphérique. Elle intègre les quelques points fournis par le sujet auquel s'ajoutent les contraintes imaginées par les étudiants.

2.1 Objectif du circuit

Le contrôleur d'interruptions a pour rôle d'informer le processeur sur l'occurrence d'une interruption valide. Il fournira alors l'adresse de la prochaine instruction à exécuter.

2.2 Fonctionnalités attendues

Les fonctions de service du circuit sont :

1. Masquer et démasquer chaque interruption individuellement
2. Contenir et permettre la configuration du vecteur d'exception
3. Permettre de configurer la priorité des interruptions
4. Ne fournir au Central Processing Unit (CPU) que les interruptions valides
5. Prendre en compte les priorités dans la génération des demandes au CPU

2.3 Utilisation du circuit

Il s'agit ici de donner l'utilisation du circuit en présentant le schéma de câblage du contrôleur d'interruptions ainsi que la définition des signaux logiques d'entrées et sorties. L'Intellectual Property (IP) à concevoir s'interface avec le bus de données, le bus d'adresses ainsi qu'un ensemble de signaux de temporisation et de contrôle. Il est possible de retrouver des signaux de temporisation comme le nWAIT et des signaux de contrôle comme le nRST ou le RnW. Leur rôle est présenté table (1).

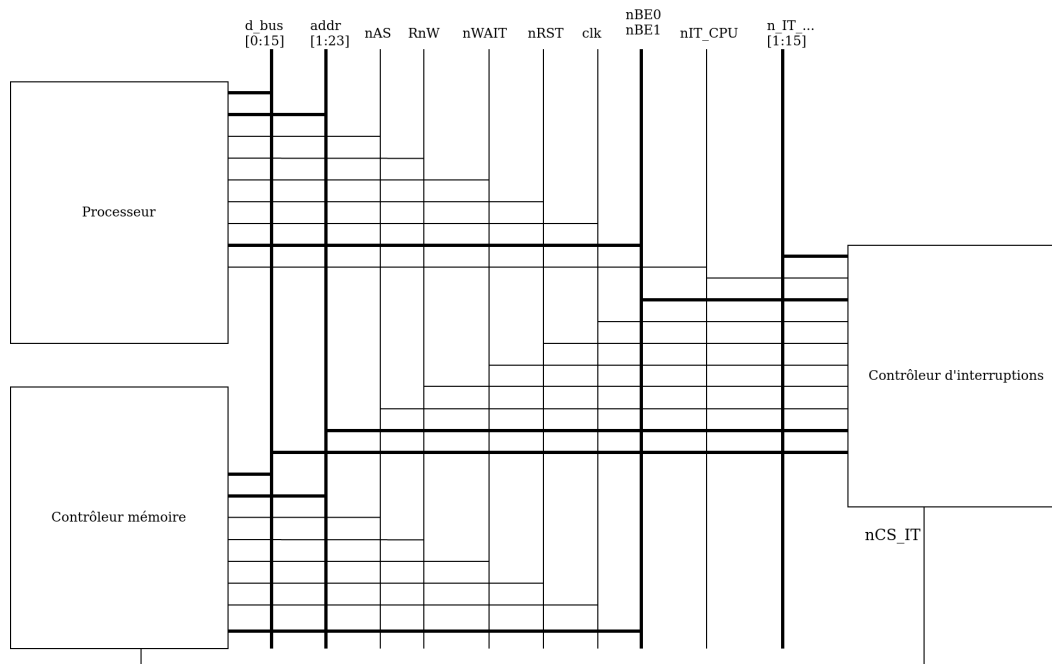


Figure 1: Schéma de câblage de l'IP à concevoir, du contrôleur mémoire et du processeur

Le contrôleur d'interruptions est également câblé avec plusieurs autres IPs du SoC. Il y a par exemple le processeur avec lequel le signal nIT_CPU est commun. D'autre part le contrôleur mémoire est connecté avec le contrôleur d'interruptions par le signal nCS_IT. L'ensemble des périphériques du SoC peut également envoyer un signal d'interruption représenté par nIT_...

Le schéma présenté ci-dessus ne précise pas le sens des signaux. Il n'est donc pas possible de savoir quelles sont les entrées et sorties du contrôleur d'interruptions. Également, les noms des 4 interruptions externes et des 11 autres provenant de divers périphériques ne sont pas donnés. La figure 3 et le tableau 1 présentent les entrées et sorties du point de vue de l'IP à concevoir ainsi que le rôle de chaque signaux.

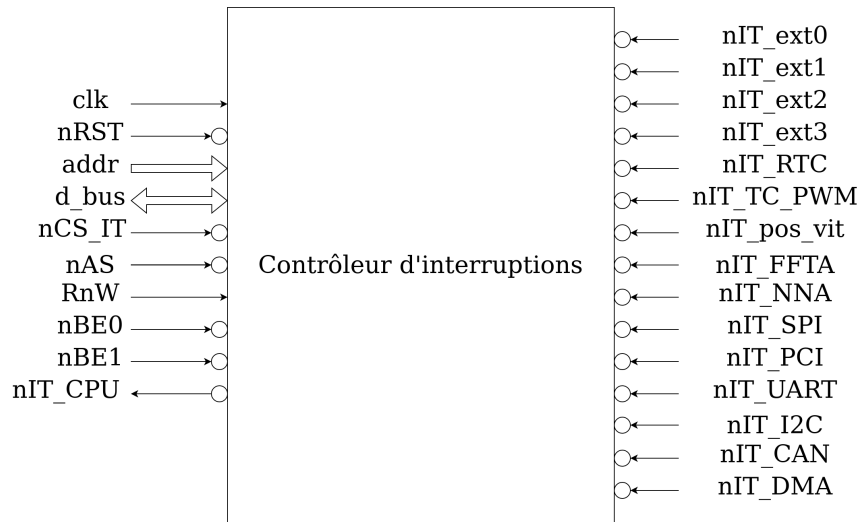


Figure 2: Entrées et sorties de l'IP à concevoir

Nom	Sens	Rôle
clk	Entrée	Signal d'horloge
nRST	Entrée	Signal de réinitialisation
addr	Entrée et sortie	Bus d'adresses
d_bus	Entrée et sortie	Bus de données
nCS.IT	Entrée	Signal de sélection du périphérique en cas d'opérations de lecture ou d'écriture
nAS	Entrée	Signal indiquant la présence d'une valeur sur le bus d'adresse
RnW	Entrée	Signal d'écriture ou de lecture 0 : écriture 1 : lecture
nBE0	Entrée	Signal indiquant la structure de la mémoire 0 : little-endian 1 : big-endian
nBE1	Entrée	Signal indiquant la taille de la donnée 0 : 8 bits 1 : 16 bits
nIT_CPU	Sortie	Signal pour le processeur avertissant qu'une interruption est demandée de la part d'un périphérique
nIT_ext0	Entrée	Signal d'interruption extérieure numéro 0
nIT_ext1	Entrée	Signal d'interruption extérieure numéro 1
nIT_ext2	Entrée	Signal d'interruption extérieure numéro 2
nIT_ext3	Entrée	Signal d'interruption extérieure numéro 3
nIT_RTC	Entrée	Signal d'interruption provenant du périphérique RTC
nIT_TC.PWM	Entrée	Signal d'interruption provenant du Timer et PWM
nIT_pos_vit	Entrée	Signal d'interruption provenant du périphérique de mesure position et vitesse
nIT_FFTA	Entrée	Signal d'interruption provenant de l'accélérateur transformée de Fourier discrète
nIT_NNA	Entrée	Signal d'interruption provenant de l'accélérateur réseau de neurones
nIT_SPI	Entrée	Signal d'interruption provenant du périphérique de communication SPI
nIT_PCI	Entrée	Signal d'interruption provenant du périphérique de communication PCI
nIT_UART	Entrée	Signal d'interruption provenant du périphérique de communication UART
nIT_I2C	Entrée	Signal d'interruption provenant du périphérique de communication I2C
nIT_CAN	Entrée	Signal d'interruption provenant du périphérique de communication CAN
nIT_DMA	Entrée	Signal d'interruption provenant du périphérique d'accès direct à la mémoire.

Table 1: Sens et rôle des signaux

Les noms des signaux présentent un suffixe n signifiant que ceux-ci sont actifs à l'état bas. Par exemple, le signal de sélection nCS.IT est actif à l'état bas. Ainsi un signal de sélection à l'état logique 0 signifie que le contrôleur d'interruptions est sélectionné pour une opération de lecture ou d'écriture dans un des registres.

Conventionner ces signaux comme actif à l'état bas n'est pas anodin. Historiquement,

pour des anciennes technologies Transistor Transistor Logic (TTL), les signaux actifs à l'état bas sont davantage robustes aux bruits. À titre d'exemple, un signal de sélection CS actif à l'état haut sélectionne un périphérique lorsque celui-ci est à l'état logique 1. Pour des raisons purement physiques (glitch sur le signal, baisse de tension à cause de la résistivité des interconnexions, chute de l'alimentation à cause d'un fort tirage de courant), ce signal à l'état 1 peut passer à l'état de haute impédance Z, voire à l'état bas. Un tel problème causerait la perte de données à lire ou écrire mais plus généralement des problèmes de sécurité.

2.4 Chronogrammes caractéristiques

Cette partie détaille les contraintes temporelles vis-à-vis des entrées sorties. Il est possible d'identifier deux types d'échanges pour ce périphérique. Le premier via le bus est illustré avec le chronogramme figure 3.

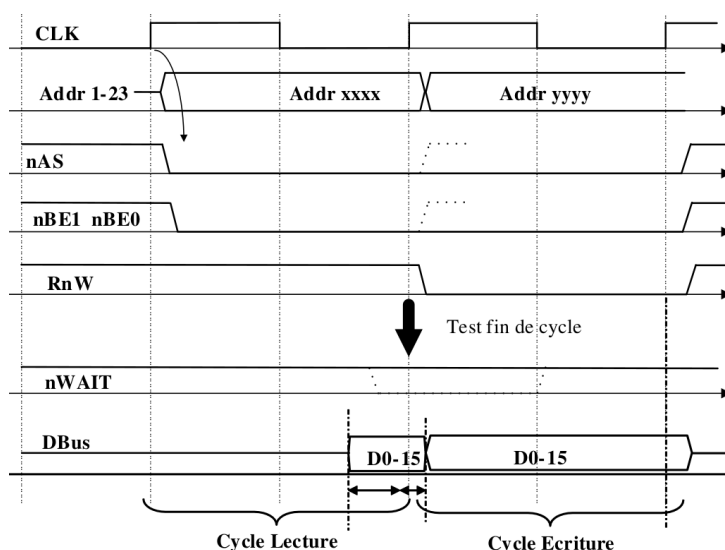


Figure 3: Chronogramme spécification du bus

On retrouve une partie des signaux définis dans la figure 2. Lors du premier front d'horloge une valeur est placée sur le bus d'adresse comme l'indique sa valeur et nAS. Le périphérique doit alors lire ou écrire la valeur correspondante sur le bus de données avant le prochain front montant. C'est ce qu'indique la flèche centrale sur cette figure. Respecter cette contrainte permettra d'assurer la compatibilité entre les entités communiquant sur le bus. Ce dernier permet un échange bidirectionnel entre le processeur et le contrôleur. Il servira à accéder par lecture et écriture aux différents registres. Parmi eux, on identifie le vecteur d'exception qui sauvegarde les adresses mémoire auxquelles le processeur va brancher. Le registre de masquage permettra de désactiver individuellement chaque entrée. La priorité par défaut de chaque entrée pourra également être modifiée.

Le second type de communication a pour but de notifier le CPU lors d'interruptions. Elles peuvent provenir des autres périphériques présents dans le microcontrôleur ou de sources externes. L'objectif est d'informer le processeur via le signal nIT_CPU en suivant le régime de priorités. La sensibilité du périphérique est définie sur l'état des signaux et non les fronts. La gestion d'une interruption commence lorsque le signal émetteur passe à l'état bas et s'achève lorsqu'il passe à l'état haut.

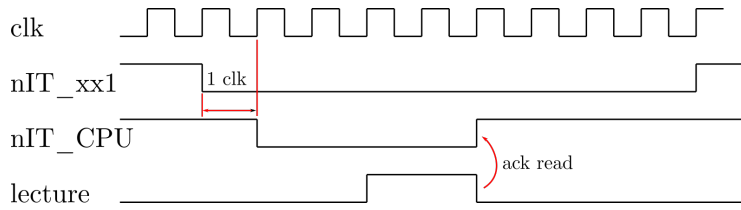


Figure 4: Chronogramme spécification signal IT au processeur

Le chronogramme figure 4 illustre le cas d'un signal d'interruption provenant d'un périphérique et la séquence induite concernant le signal `nIT_CPU`. Une demande d'interruption s'identifie par le passage à l'état bas de `nIT.xxx`. Après 1 cycle de traitement visant à confirmer la validité de la requête en consultant le registre de masquage, le processeur est informé, lui aussi, par le passage à l'état bas sur `nIT_CPU`. Il sauvegarde alors le contexte et lit le registre de branchement. Cette action agit comme un **acquiescement** et le signal `nIT_CPU` est passé à l'état haut. Le processeur se charge d'informer le périphérique émetteur, souvent à la fin de la routine, afin qu'il baisse le drapeau dans son registre d'état. Tant que cela n'est pas fait, aucune demande d'interruption de niveau inférieur ou égal ne sera transmis. Le chronogramme figure 4 illustre justement le cas d'une demande **moins prioritaire** qui apparaît durant l'exécution d'une première.

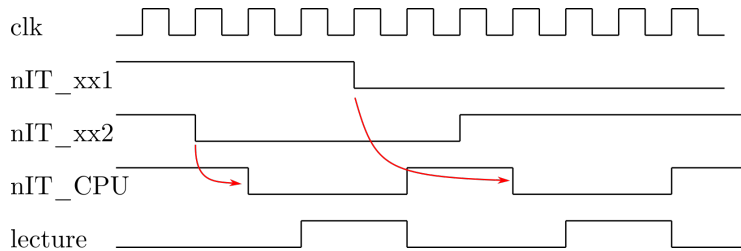


Figure 5: Chronogramme spécification 2 signaux IT cas 1

Dans cette situation, le périphérique d'indice 2 est plus prioritaire que celui d'indice 1. La demande est faite comme précédemment en passant le signal `nIT_xx2` à l'état bas. Le contrôleur réagit alors en informant le processeur. Pendant cette opération, une seconde interruption de niveau inférieur apparaît. Aucune réaction n'est attendu de la part du contrôleur qui attend la fin de l'exécution précédente pour générer un front descendant sur `nIT_CPU`. Le traitement se déroule alors comme auparavant. Il est intéressant de porter attention sur la place que joue le signal `nIT_xx2` dans ce cas. Son passage à l'état haut autorise de nouveau l'occurrence des interruptions de niveau inférieur ou égal. Il reviendra donc au développeur de placer cet acquiescement en fin de routine d'interruption sans quoi la priorisation ne sera pas garantie. Pour poursuivre la description des contraintes temporelles, on peut identifier un second cas venant de paire avec le précédent. Le chronogramme figure 6 illustre la situation où une demande **plus prioritaire** apparaît durant l'exécution d'une première.

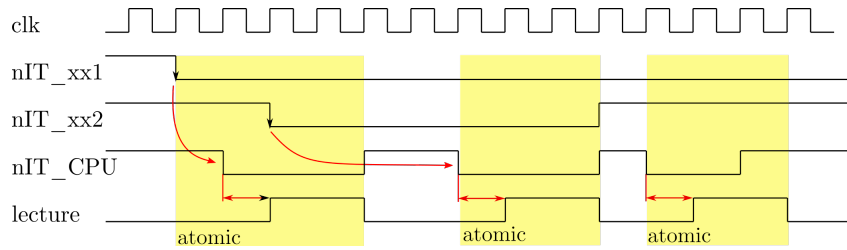


Figure 6: Chronogramme spécification 2 signaux IT cas 2

De nouveau, dans ce cas, le périphérique d'indice 2 est plus prioritaire que celui d'indice 1. Une première demande provient du signal `nIT_xx1`. Mais durant son traitement, il apparaît, une seconde interruption de niveau supérieur. Le contrôleur attend alors l'aquittement du CPU pour la première avant de l'informer pour la seconde. Cela est primordial pour éviter toute incohérence dans l'état des registres. En effet, sans cette mesure, si une interruption intervient entre la lecture du MSB et du LSB, l'adresse de branchement est corrompue. Dans la figure 6 ces zones dites "atomic" sont marquées en jaune. Lorsque le processeur a effectué la lecture servant d'aquittement, il est de nouveau notifié. Le signal `nIT_CPU` doit alors rester au minimum un cycle horloge à l'état bas afin de générer un front descendant synchrone. La suite du cycle se passe comme précédemment pour le contrôleur d'IT. C'est le processeur qui gère le rétablissement du contexte de l'interruption de plus faible priorité.

Nous venons de présenter trois situations de gestion d'interruptions. Cette analyse est loin d'être exhaustive, principalement en raison de l'infinité des instants d'occurrence possibles. Il semble toutefois que les cas traités couvrent l'ensemble des séquences d'événements. Pour vérifier la conformité d'un cas quelconque, il sera alors possible de s'y référer en identifiant la situation et en adaptant l'échelle temporelle.

2.5 Contraintes du projet

La section qui suit traite de la gestion du projet. Il s'agit de préciser la planification du projet avec des divers livrables à fournir à des dates précises. La figure (7) ci-dessous est le plan de développement du projet. Les losanges bleus sont les jalons à fournir. La conception de cet IP a débuté le 5 octobre 2022, c'est-à-dire à la 40^{ème} semaine de l'année. Le rendu de l'IP et du rapport de conception s'effectuera la semaine 49.

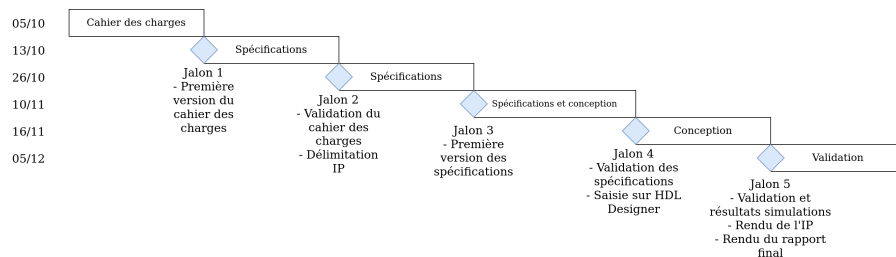


Figure 7: Planification du projet avec ses jalons

Également, le projet est contraint d'un point de vue ressources disponibles. La liste suivante présente les ressources attribuées pour la conception de ce contrôleur d'interruptions.

- 2 concepteurs
- Outils EDA de Mentor Graphics (propriété de Siemens EDA)
- Carte d'évaluation ZYNQ-7 basé sur un FPGA Zynq-7000

3 Spécifications

3.1 Caractérisation de l'environnement

Il s'agit dans cette partie de caractériser l'environnement c'est-à-dire de d'identifier les entités interagissant avec le circuit à concevoir et décrire leur évolution à l'aide d'automates. L'environnement du circuit à concevoir est constitué de trois entités :

- Le processeur
- Le contrôleur mémoire fournissant le signal de sélection nCS_IT.
- Les 15 entités informant au contrôleur d'interruptions la présence d'une interruption. Ce groupe d'entités est appelé "source d'interruptions".

L'ensemble processeur + contrôleur mémoire peut configurer le contrôleur d'interruptions et opérer des écritures et lectures au sein de ses registres. Cet ensemble sera par la suite nommé "système à microprocesseur". L'entité source d'interruptions envoie au circuit à concevoir la présence d'une interruption à traiter.

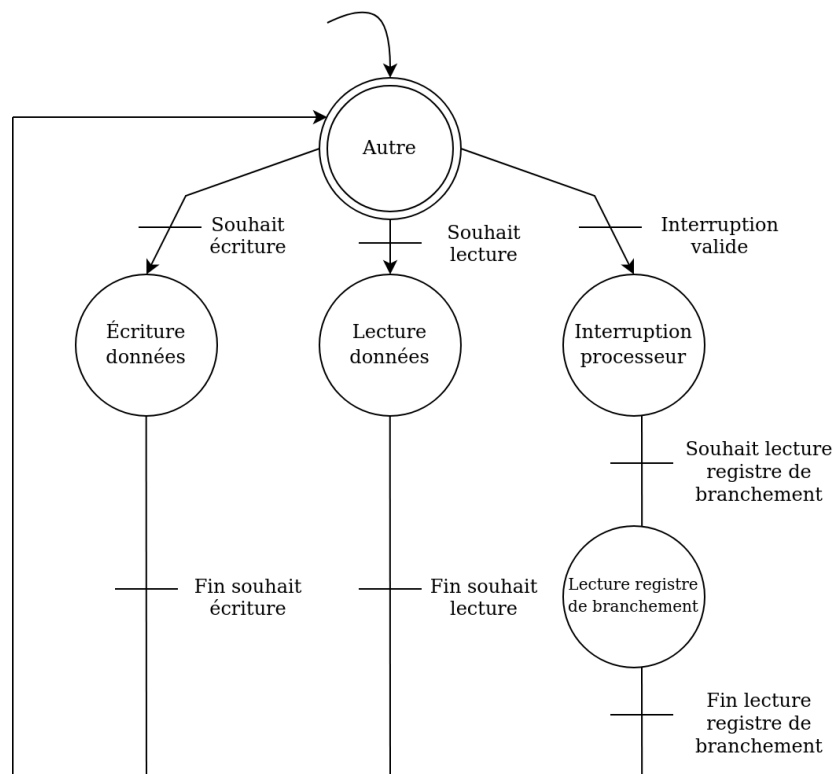


Figure 8: Comportement du système à microprocesseur

La figure ci-dessus représente le comportement de l'entité système à microprocesseur. Celui-ci possède trois cycles de fonctionnement. Le processeur peut opérer des cycles de lecture et d'écriture dans les registres du contrôleur d'interruptions. Le souhait d'écriture ou de lecture s'effectue par la mise à l'état bas du signal nCS_IT. La fin de ces cycles est notifiée par la mise à l'état haut du signal nAS. Le terme "données" peut signifier la valeur des registres de configuration et également l'adresse de branchements.

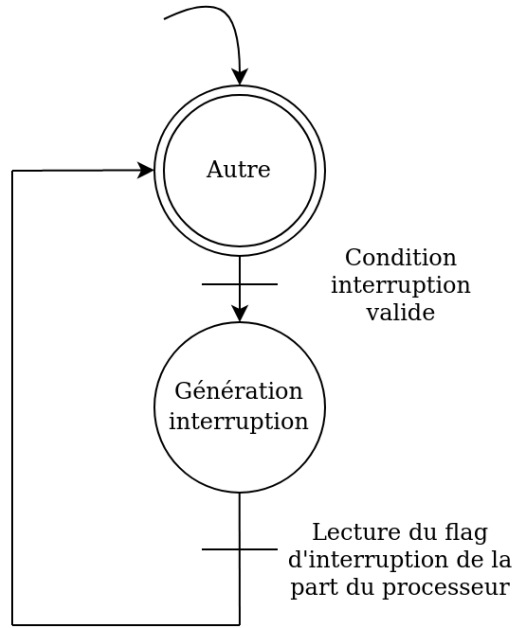


Figure 9: Comportement de l'entité source d'interruptions

La figure ci-dessus est l'automate de l'entité source d'interruptions. Lorsque la condition d'interruption est valide, le périphérique génère un signal en direction du contrôleur d'interruptions. Ce signal reste actif jusqu'à la lecture du flag d'interruption de la part du processeur.

3.2 Entrées et sorties du composant

La caractérisation de l'environnement sous forme d'automates donne les relations d'entrées et sorties du circuit à concevoir avec les diverses entités. Il est alors possible de présenter de manière structurée le circuit à concevoir et les entités de l'environnement.

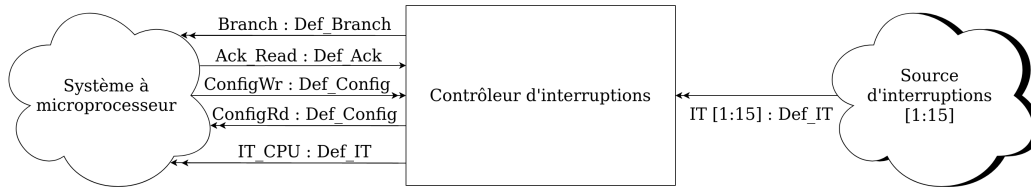


Figure 10: Entrées et sorties du circuit à concevoir

Le tableau ci-dessous récapitule les relations avec leur sens, leur catégorie et leur type.

Entités	Relation	Catégorie	Sens	Type
Système à microprocesseur	Branch	Permanent	Sortie	Def.Branch
	Ack_Read	Évènementiel	Entrée	Def_Ack
	ConfigWr	Permanent	Entrée	Def_ConfigWr
	ConfigRd	Permanent	Sortie	Def_ConfigRd
	IT_CPU	Permanent	Sortie	Def_IT
Source d'interruptions [1:15]	IT [1:15]	Permanent	Entrée	Def_IT

Table 2: Sens et rôle des signaux

La relation Ack.Read est de type Def_Ack, une donnée de type booléen (true : la lecture de l'adresse de branchement est terminée, false sinon). IT[1:15] est un tableau de 15 éléments de type Def_IT, une donnée de type booléen (true : une interruption est reçue de la part de n-ième l'IP, false sinon). IT_CPU est également de type Def_IT (true : une interruption doit être traitée par le CPU, false sinon). Chaque autre relation est une composition de sous-relations comme définie ci-après.

$$\text{Branch} = \text{ID} + \text{@blx}$$

$$\text{ConfigWr} = \text{@handler} + \text{priorité} + \text{masque}$$

$$\text{ConfigRd} = \text{ID} + \text{@handler} + \text{@blx} + \text{priorité} + \text{masque} + \text{pending} + \text{traitement}$$

ID représente l'identifiant de l'interruption. @blx est la sous-relation qui précise l'adresse de branchement de la sous-routine de l'interruption à traiter. @handler est un tableau comportant les adresses des sous-routines d'interruptions. La relation priorité indique la priorité attribuée pour une interruption, masque précise si l'interruption est masquée ou non. En ce qui concerne pending, il s'agit d'une sous-relation qui informe si une interruption est mise en attente pour être traitée ultérieurement. Enfin, traitement prends en compte si une interruption a déjà été traitée ou non avant que celle-ci soit clear par l'IP concernée.

Sous-relation	Type	Donnée
ID	Def_ID	Entier de 0 à 14
@blx	Def.@blx	Entier de 0 à $2^{22} - 1$
@handler	Def.@handler	Tableau de 30 éléments de taille Def_Data valant $16 \times \text{Def.Bit}$ avec Def_Bit 0 ou 1
priorité	Def.priorité	Entier de 0 à 7 0 : le plus prioritaire 7 : le moins prioritaire
masque	Def.masque	Booléen true : IT masquée (prise en compte) false : non masquée
pending	Def.pending	Booléen true : IT mise en attente false sinon
traitement	Def.traitement	Booléen true : IT à traiter false : IT déjà traitée

Table 3: Sous-relations et type de données

Il est possible d'introduire une relation interne nommée infoIT. Le k-ième élément de infoIT correspond à la k-ième interruption, c'est-à-dire la valeur de ID. Cet indice k varie de 0 à 14 (0-14 représentant les 15 interruptions).

$$\text{infoIT}[k] = \text{ID} + \text{priorité} + \text{masque} + \text{pending} + \text{traitement}$$

3.3 Spécifications fonctionnelles

Les spécifications fonctionnelles comprennent la liste des fonctions du système pour l'application (fonctions externes) et la description du comportement du système et de l'environnement pour ces fonctions [2].

La fonction première du contrôleur d'interruption est d'informer le processeur lorsqu'une interruption valide survient. Une interruption valide est définie comme étant non masquée et de niveau suffisant pour être prise en compte. La gestion des niveaux de priorité est la seconde fonction à considérer. Elle est directement induite par la première. Une interruption en cours ne peut être interrompue que par une interruption de niveau strictement supérieur. Lorsque le processeur acquitte la fin du traitement, les interruptions de niveaux inférieurs ou égaux peuvent être à nouveau considérées comme valides. Le contrôleur fournit un niveau de priorité par défaut pour chaque source. La troisième fonction concerne le masquage des sources d'interruption individuellement. Une source masquée ne générera pas de signal au processeur. Un événement déjà en cours de traitement par le contrôleur ne pourra être masqué. Si un signal d'interruption est généré alors qu'il est masqué, il est ignoré, mais s'il est démasqué alors qu'il est toujours actif alors il sera pris en compte. La quatrième fonction principale spécifie le vecteur d'exception. Le contrôleur lorsque qu'il informe le processeur doit immédiatement lui indiquer vers quelle adresse il doit brancher. Il est possible de configurer une adresse par source. Sa configuration doit se faire lorsque le contrôleur est désactivé. Si aucune adresse n'est présente pour une source donnée alors une par défaut sera attribuée.

Nom	Description
fn1	Informé le processeur
fn2	Gérer les niveaux de priorité
fn3	Masquer individuellement les sources
fn4	Configurer le vecteur d'exception

Table 4: Synthèse des spécifications fonctionnelles

Le tableau 4 synthétise les spécifications fonctionnelles. Il permet de contrôler que chaque point est bien traité tout au long de la conception. Il n'est cependant pas précis et devra être traité avec la description détaillée ci-dessus.

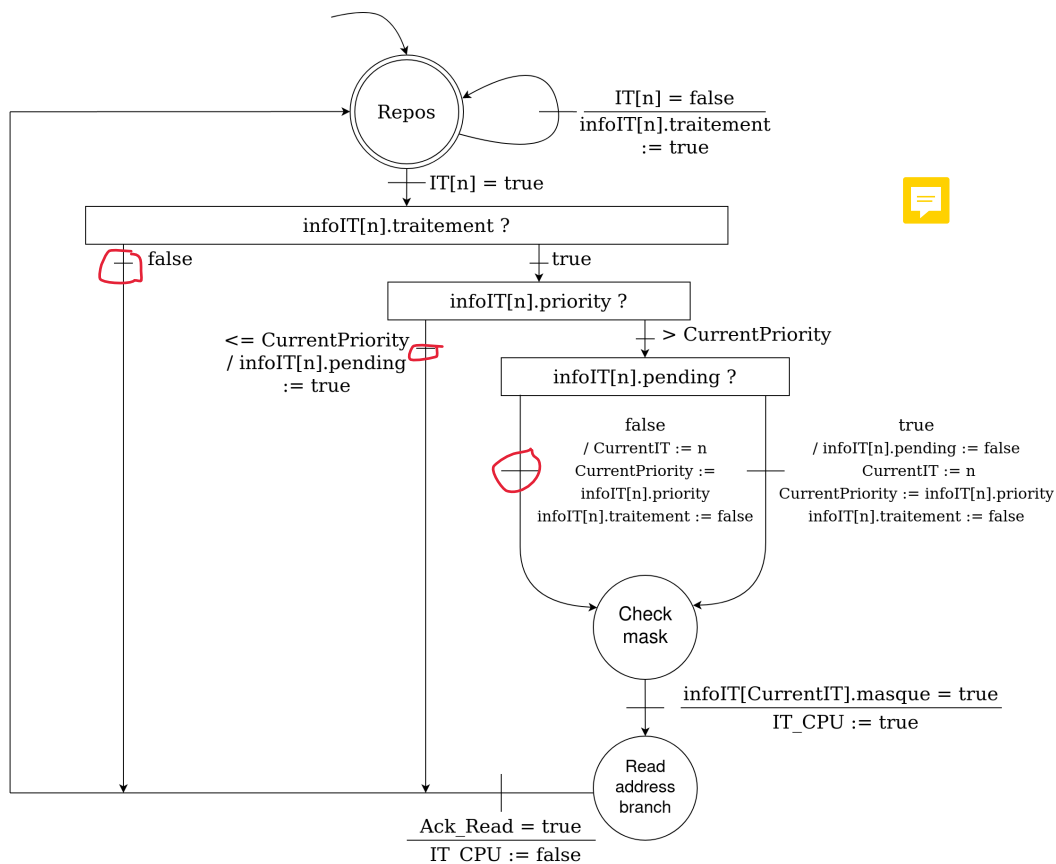


Figure 11: Automate du contrôleur d'interruptions

ID	Interruptions correspondantes	Adresses de branchement par défaut
0	nIT_ext0	0x000000
1	nIT_ext1	0x000000
2	nIT_ext2	0x000000
3	nIT_ext3	0x000000
4	nIT_RTC	0x000000
5	nIT_TC.PWM	0x000000
6	nIT_pos_vit	0x000000
7	nIT_FFTA	0x000000
8	nIT_NNA	0x000000
9	nIT_SPI	0x000000
10	nIT_PCI	0x000000
11	nIT_UART	0x000000
12	nIT_I2C	0x000000
13	nIT_CAN	0x000000
14	nIT_DMA	0x000000

Table 5: Vecteurs d'interruptions

3.4 Spécification des registres

Le tableau 6 ci-dessous donne l'organisation des registres dans le contrôleur. Une vue détaillée par chacun d'eux est présentée ensuite.


Adresse	Nom	Type	Reset	Description
0x00000000	CTRL_IT_EN	RW	0x0000	Registre activation contrôleur d'IT
0x00000002	CTRL_IT_MSQ	RW	0x0000	Registre de masquage
0x00000004	CTRL_IT_PEND	R	0x0000	Registre IT suspendues (pending)
0x00000006– 0x00000008	CTRL_IT_BRA	R	0x0000	Registre de branchement
 0x0000000A– 0x0000000C	CTRL_IT_ADDR_0	RW	0x0000	Adresses de branchement ch 0
0x0000000E– 0x00000010	CTRL_IT_ADDR_1	RW	0x0000	Adresses de branchement ch 1
0x00000012– 0x00000014	CTRL_IT_ADDR_2	RW	0x0000	Adresses de branchement ch 2
0x00000016– 0x00000018	CTRL_IT_ADDR_3	RW	0x0000	Adresses de branchement ch 3
0x0000001A– 0x0000001C	CTRL_IT_ADDR_4	RW	0x0000	Adresses de branchement ch 4
0x0000001E– 0x00000020	CTRL_IT_ADDR_5	RW	0x0000	Adresses de branchement ch 5
0x00000022– 0x00000024	CTRL_IT_ADDR_6	RW	0x0000	Adresses de branchement ch 6
0x00000026– 0x00000028	CTRL_IT_ADDR_7	RW	0x0000	Adresses de branchement ch 7
0x0000002A– 0x0000002C	CTRL_IT_ADDR_8	RW	0x0000	Adresses de branchement ch 8
0x0000002E– 0x00000030	CTRL_IT_ADDR_9	RW	0x0000	Adresses de branchement ch 9
0x00000032– 0x00000034	CTRL_IT_ADDR_10	RW	0x0000	Adresses de branchement ch 10
0x00000036– 0x00000038	CTRL_IT_ADDR_11	RW	0x0000	Adresses de branchement ch 11
0x0000003A– 0x0000003C	CTRL_IT_ADDR_12	RW	0x0000	Adresses de branchement ch 12
0x0000003E– 0x00000040	CTRL_IT_ADDR_13	RW	0x0000	Adresses de branchement ch 13
0x00000042– 0x00000044	CTRL_IT_ADDR_14	RW	0x0000	Adresses de branchement ch 14
0x00000046	CTRL_IT_PRIO_0.1	RW	0x0000	Registre des priorités
0x00000048	CTRL_IT_PRIO_2.3	RW	0x0000	Registre des priorités
0x0000004A	CTRL_IT_PRIO_4.5	RW	0x0000	Registre des priorités
0x0000004C	CTRL_IT_PRIO_6.7	RW	0x0000	Registre des priorités
0x0000004E	CTRL_IT_PRIO_8.9	RW	0x0000	Registre des priorités
0x00000050	CTRL_IT_PRIO_10.11	RW	0x0000	Registre des priorités
0x00000052	CTRL_IT_PRIO_12.13	RW	0x0000	Registre des priorités
0x00000054	CTRL_IT_PRIO_14	RW	0x0000	Registre des priorités

Table 6: Organisation des registres du contrôleur d'interruption (vue globale)

Le registre CTRL_IT_EN contrôle l'activation de la partie opérative du composant. Lorsque le bit **Enable** vaut zero, les interruptions ne sont plus prises en compte, il est toujours possible d'écrire et lire l'état des registres.

Bit	Nom	Fonction
[0]	Enable	Active ou lit l'état d'activation du périphérique
		Lecture 0 Le périphérique est désactivé
		1 Le périphérique est activé
		Écriture 0 Désactivation du périphérique
		1 Activation du périphérique
[15 : 1]	-	Réservés

Table 7: Détails du registre CTRL_IT_EN

Le registre CTRL_IT_PEND contient les sources d'interruptions actives, celles qui ne sont pas masquées.

Bit	Nom	Fonction
[14 : 0]	Pending	Lit l'état de suspension (pending) de la source
		Lecture 0 La source est désactivé
		1 La source est activé
		Écriture 0 No effect
		1 No effect
[15]	-	Réservé

Table 8: Détails du registre CTRL_IT_PEND

Le registre CTRL_IT_BRA indique l'adresse de branchement de l'interruption en cours. Son contenu n'est valide que lorsque le processeur a été notifié pas le signal nIT_CPU.

Bit	Nom	Fonction
[15 : 0]	BRA_LSB	Lit 16 LSB adresse de branchement
[7 : 0]	BRA_MSB	Lit 8 MSB adresse de branchement
[15 : 8]	-	Réservés

Table 9: Détails du registre CTRL_IT_BRA

Le tableau 10 décrit l'organisation des registres permettant la configuration de l'adresse de branchement en fonction du numéro d'interruption. Une lecture peut être faite à tout instant. Une écriture doit être faite lorsque le périphérique est désactivé (CTRL_IT_EN = 0x0000). Cette règle est fixée afin d'éviter l'occurrence d'une interruption entre les deux cycles d'écritures nécessaires au chargement d'une adresse.

Bit	Nom	Fonction
[15 : 0]	ADDR_LSB	Lit et écrit 16 LSB adresse de branchement
[7 : 0]	ADDR_MSB	Lit et écrit 8 MSB adresse de branchement
[15 : 8]	-	Réservés

Table 10: Détails du registre CTRL_IT_ADDR_N

Le tableau 11 décrit l'organisation des registres permettant la configuration du niveau de priorité. Pour simplifier le programme C, les données sont alignées sur un octet. Si le nombre de sources d'interruption est impaire, le dernier registre ne contiendra qu'une seule valeur. Dans notre cas, nous avons 15 sources, CTRL_IT_PRIO_N_14 ne contient que la priorité 14 sur les bits de 0 à 2.

Bit	Nom	Fonction
[2 : 0]	PRIO_N	Lit et écrit le niveau de priorité de N
[7 : 3]	-	Réservés
[10 : 8]	PRIO_N+1	Lit et écrit le niveau de priorité de N+1
[15 : 11]	-	Réservés

Table 11: Détails du registre CTRL_IT_PRIO_N_N+1

3.5 Écriture des procédures de base

Dans cette partie, des codes C sont donnés en exemple pour illustrer l'utilisation du périphérique.

```

1  #include <stdint.h>
2  #define CTRL_IT_BASE_ADDR 0xF0F0F0
3
4  typedef enum {
5      nIT_ext0 = 0, nIT_ext1, nIT_ext2, nIT_ext3, nIT_RTC, nIT_TC_PWM,
6      nIT_pos_vit, nIT_FFTA, nIT_NNA, nIT_SPI, nIT_PCI, nIT_UART,
7      nIT_I2C, nIT_CAN, nIT_DMA, NB_IT
8  } ID_IT_t;
9
10 typedef struct {
11     uint8_t ch : 3;
12     uint8_t RESERVED : 5;
13 } CTRL_IT_PRIO_t;
14
15 typedef struct {
16     uint16_t CTRL_IT_EN;
17     uint16_t CTRL_IT_MSK;
18     uint16_t CTRL_IT_PEND;
19     void (*handler)(void);
20     /* array of function pointer */
21     void (*CTRL_IT_ADDR[NB_IT])(void);
22     CTRL_IT_PRIO_t prio[NB_IT];
23 } CTRL_IT_t;
24
25 CTRL_IT_t * CTRL_IT = (CTRL_IT_t * ) CTRL_IT_BASE_ADDR;
```

Le code ci-dessus définit la structure du contrôleur IT en langage C. Le résultat est le type CTRL_IT_t. Une instance globale est créée et initialisée avec l'adresse de base du périphérique. Sa valeur est choisie arbitrairement dans cet exemple. Les trois premiers éléments de la structure sont déclarés sur 16 bits bien que certains champs soient réservés. Il revient au développeur d'appliquer un masque cohérent en se basant sur le détail des registres page précédente. Les éléments **handler** et CTRL_IT_ADDR sont définis comme des pointeurs de fonction, car les données contenues sont des adresses de fonction. Concevoir conjointement l'organisation des registres et le programme C permet d'aboutir à un compromis entre complexité matérielle et logicielle. Par exemple, nous avons cherché à représenter la priorité sous forme d'un tableau d'octet. La complexité du VHDL est acceptable et le développeur peut accéder à la priorité avec un tableau où l'indice est le numéro d'interruption.



```

1 void UART_Handler(void)
2 {
3     /* Do somethink */
4     /* clear UART IT flag */
5 }
6
7 void CTRL_IT_init_UART(void)
8 {
9     /* Disable CTRL IT */
10    CTRL_IT->CTRL_IT_EN = 0x0000;
11    /* Mask the IC channel */
12    CTRL_IT->CTRL_IT_MSK = 1UL << nIT_UART;
13    /* Set handler addr */
14    CTRL_IT->CTRL_IT_ADDR[nIT_UART] = &UART_Handler;
15    /* Enable CTRL IT */
16    CTRL_IT->CTRL_IT_EN = 0x0001;
17 }

```

Les deux procédures ci-dessus présentent un exemple d'initialisation du périphérique pour le canal Universal Asynchronous Receiver Transmitter (UART). Il est préférable d'effectuer la configuration lorsqu'il est désactivé. Pour rappel, l'écriture dans le registre `CTRL_IT_ADDR` implique nécessairement sa désactivation. L'énumération du type `ID_IT_t` facilite l'écriture par l'usage de `nIT_UART`. L'adresse de la fonction `UART_Handler` est affectuée dans le registre de configuration des adresses de branchement. Le périphérique est activé à nouveau à la fin de la configuration. Dans cet exemple, la fonction à appeler est créée, il revient à l'utilisateur de la compléter en prenant soit de la finir par acquitter sur le périphérique en question (clear flag).

3.6 Spécifications opératoires

Les spécifications opératoires concernent les définitions des principales opérations du contrôleur d'interruptions et de leur vérification. Il s'agira de définir les modalités de vérification du circuit pour diverses opérations.

3.6.1 Modalités de vérification du circuit

Opération d'écriture et lecture dans un registre

Un banc de test sera mis en place afin d'écrire puis lire dans l'ensemble des registres de l'IP. Les phases d'écriture et lecture seront réalisées par affectation des signaux d'interfaçage entre le contrôleur d'interruptions et le système à microprocesseur (bus de données, bus d'adresses, signaux de contrôle).

Opération de réinitialisation de l'IP

Un banc de test sera mis en place afin de réinitialiser l'entièrete du contrôleur d'interruptions. Une telle réinitialisation sera faite par le biais du signal de contrôle `nRST` synchrone à l'horloge.

Traitement d'une seule interruption

Un banc de test sera mis en place afin de traiter une interruption. Il s'agira de traiter l'interruption comme l'indique la figure (4).

Traitement de deux interruptions de priorité différente

Un banc de test sera mis en place afin de traiter deux interruptions de priorité différente. Il s'agira de traiter l'interruption comme l'indique la figure (6).

3.7 Spécifications technologiques

3.7.1 Contraintes de répartition

La contrainte de répartition est utile lorsque la présence des calculateurs (des processeurs) sont géographiquement répartis. Dans le cadre de la conception du contrôleur d'interruptions, il n'y a aucune contrainte de répartition. En effet, l'IP se trouve dans la même puce que le processeur.

3.7.2 Contraintes technologiques

L'IP à concevoir se situe dans le flow de conception FPGA et non ASIC. Le contrôleur d'interruptions sera donc implémenté sur un FPGA. La carte d'évaluation utilisée sera la Zedboard basée sur un FPGA Zynq-7000. La table ci-dessous présente quelques caractéristiques du Zynq-7000.

Système à microprocesseur	Dual-core ARM Cortex-A9 d'architecture ARMv7-A avec une fréquence max de 766 MHz
Mémoires	32 kB de cache de niveau 1 par processeur 512 kB de cache de niveau 2, 256 kB de RAM
FPGA Artix-7	53 200 LUTs et 106 400 bascules

Figure 12: Caractéristiques du FPGA Zynq 7000

4 Conception

4.1 Délimitation fonctionnelle

La section suivante est un rappel de la délimitation du circuit à concevoir. Celle-ci présente le contrôleur d'interruptions d'un point de vue extérieur avec les 6 relations définies précédemment.

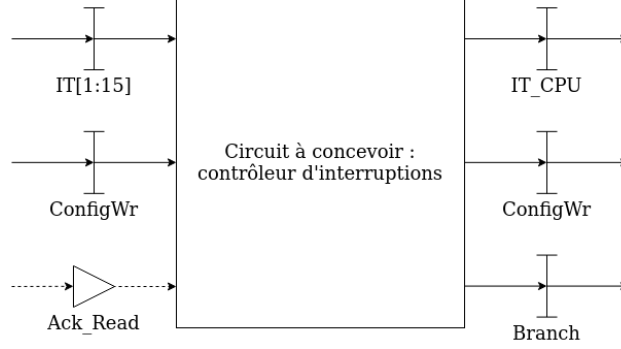


Figure 13: Délimitation fonctionnelle du contrôleur d'interruptions

Il s'agira par la suite de se placer dans un point de vue interne afin de présenter les différents blocs composant le contrôleur d'interruptions avec ses relations internes. L'évènement **Ack_Read** permet d'acquitter la lecture de l'adresse de branchement par le processeur. Cette relation peut être déduite selon le contenu du bus d'adresse (si la lecture est faite dans le registre **blx** alors il y a acquittement).

4.2 Décomposition fonctionnelle et introduction des interfaces

La figure ci-dessous présente le circuit interne observé de l'intérieur avec l'introduction des signaux physiques du circuit. Il existe deux blocs nommés **Interface** et **Traitement**.

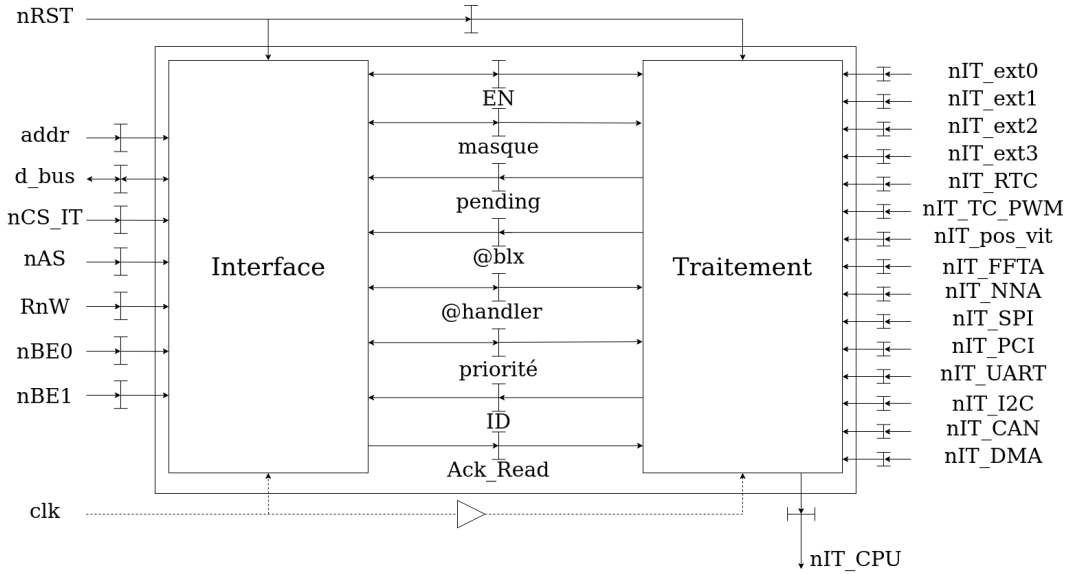


Figure 14: Décomposition fonctionnelle et introduction des interfaces

Le bloc **Traitement** est chargé de signaler le CPU d'une interruption à traiter à la suite de multiples opérations (traitement des priorités, opération de masquage, mise en suspens). Le bloc **Interface** permet de faire la jonction entre les signaux de contrôle et bus du système à microprocesseur et l'IP à concevoir. **Ack_Read** est en sortie du bloc **Interface** et avertit le bloc **Traitement** de la lecture de l'adresse de branchement. Il ne s'agit plus d'un événement mais d'une relation. C'est une variable partagée qui engendrera une action sur niveau et non sur front.

4.3 Raffinage du bloc Interface

La section suivante présente un raffinement, c'est-à-dire de réaliser une seconde décomposition en se plaçant à l'intérieur du bloc **Interface**. Ce bloc est composé de **Interface_Write** et **Interface_Read**.

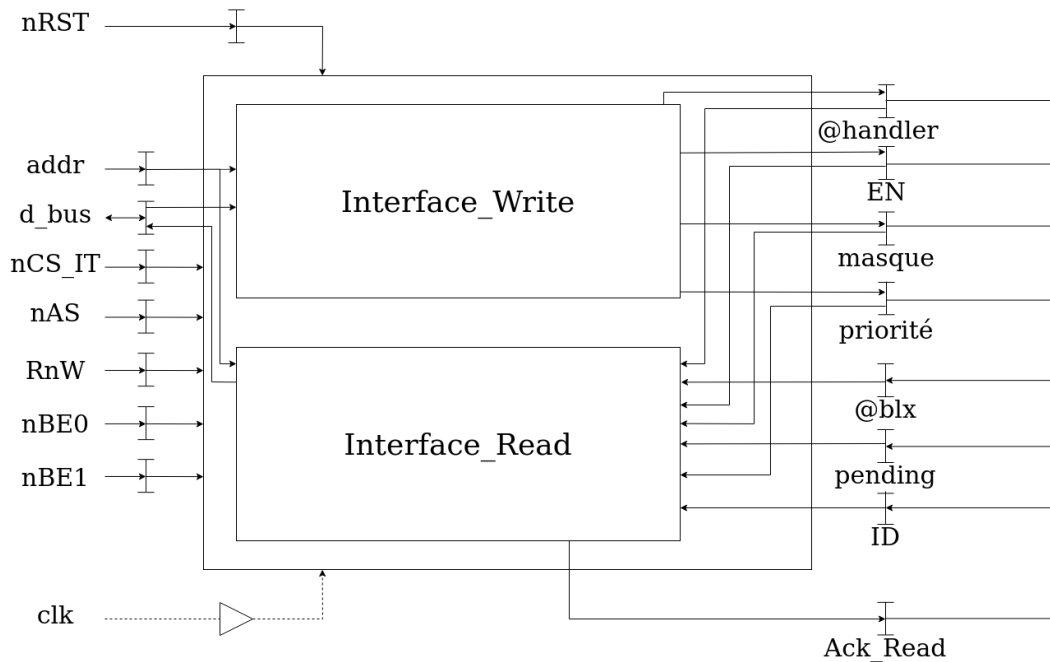


Figure 15: Raffinage et décomposition du bloc **Interface**

Cette décomposition en bloc hiérarchique permet de mettre en lumière deux fonctions du bloc **Interface**. Celui-ci peut écrire dans les registres du contrôleur d'interruptions par le biais du bloc **Interface_Write** et les lire avec **Interface_Read**.

4.4 Raffinage du bloc Traitement

Après le raffinement du bloc **Interface**, il s'agit désormais de décrire le point de vue interne du bloc **Traitement**. La structure est donnée ci-dessous.

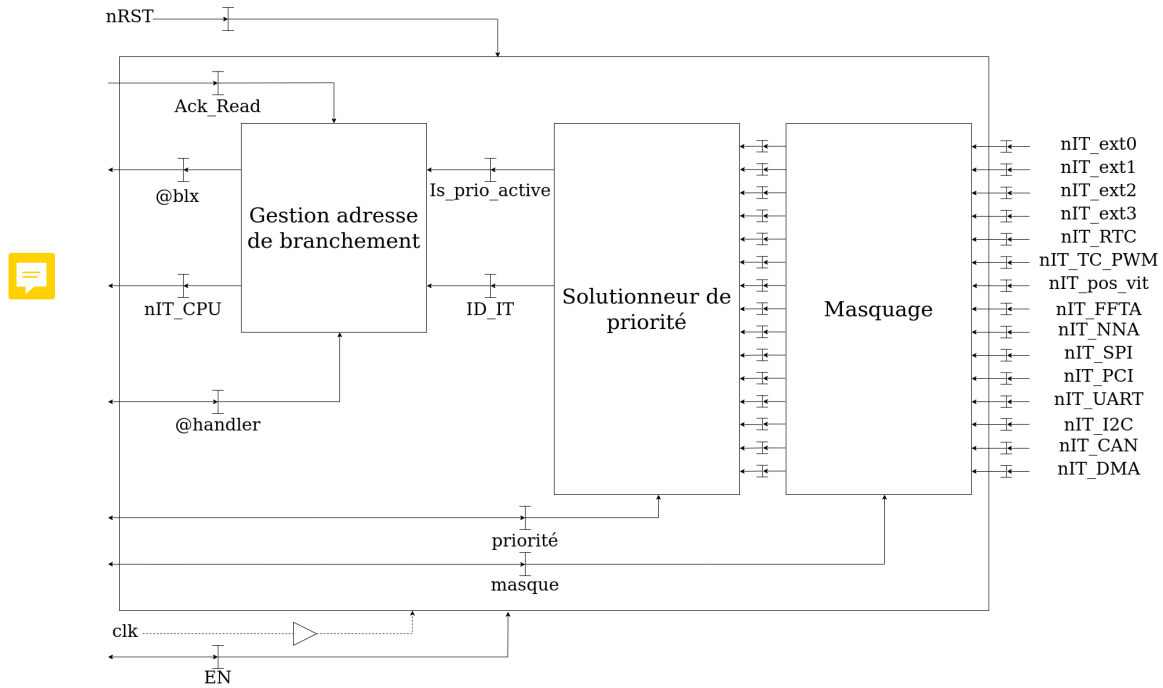


Figure 16: Raffinage et décomposition du bloc **Traitement**

Ce bloc présente trois éléments appelés respectivement **Gestion adresse de branchement**, **Solutionneur de priorité** et **Masquage**. Le bloc **Masquage** utilise les 15 interruptions ainsi que le registre masque pour filtrer les interruptions que l'utilisateur souhaite traiter. Le sous-bloc **Solutionneur de priorité** est utilisé pour traiter les interruptions selon leur niveau de priorité et d'indiquer l'identifiant de l'interruption à traiter. Le signal `Is_prio_active` informe au bloc **Gestion adresse de branchement** si l'interruption en cours de traitement est toujours active, c'est-à-dire à l'état bas. Le bloc **Gestion adresse de branchement** récupère l'identifiant de l'IT correspondant à l'indice du registre `@handler` contenant l'adresse de branchement à placer dans `@blx`. Pour que le processeur puisse lire l'adresse de branchement le signal `nIT_CPU` est activé. Suite à la lecture de la part du processeur le signal `Ack_Read` est actif afin d'informer le contrôleur d'interruption la fin de la phase de lecture et remettre `nIT_CPU` à l'état logique '1'.

4.5 Écriture des algorithmes

4.5.1 Algorithme Interface.Write

L'algorithme ci-dessous représente le comportement de `Interface.Write`. Ce bloc permet l'écriture synchrone par le processeur dans les registres internes du contrôleur d'interruption.

```
1  action Interface_Write sur clk avec
2  (
3  entree var d_bus          : Def_data;
4  entree var addr           : Def_addr;
5  entree var nRST           : Def_Bit;
6  entree var nCS_IT         : Def_Bit;
7  entree var nAS            : Def_Bit;
8  entree var RnW            : Def_Bit;
9  sortie var EN             : Def_Bit;
10 sortie var vect_priorite   : Def_vect_priorite;
11 sortie var vect_handler    : Def_vect_handler;
12 sortie var masque         : Def_masque;
13 );
14
15 const addr_EN              : Def_addr = 0x000000;
16 const addr_masque          : Def_addr = 0x000002;
17 const addr_vect_handler    : Def_addr = 0x00000A;
18 const addr_vect_priorite   : Def_addr = 0x000044;
19
20 begin
21
22 cycle:
23 begin
24     cycle H:
25     begin
26         if (nRST=0) then
27             EN := 0;
28             priorite := 0;
29             vect_handler := 0;
30             masque := 0;
31         end if;
32         if (nCS_IT=0) et (RnW=0) et (nAS=0) then
33             case addr of
34                 addr_EN:
35                     EN := d_bus;
36                 addr_masque:
37                     masque := d_bus;
38                 addr_vect_handler:
39                     vect_handler := d_bus;
40                 addr_vect_priorite:
41                     vect_priorite := d_bus;
42             end if;
43         end cycle H;
44     end cycle;
45 end Interface_Write;
```

4.5.2 Algorithme Interface_Read

L'algorithme ci-dessous représente le comportement de `Interface_Read`. Ce bloc permet la lecture asynchrone par le processeur dans les registres internes du contrôleur d'interruption. Un cas particulier est à considérer lors de la lecture de l'adresse de branchement `blx`. Le signal `Ack_Read` passe à l'état logique 1 pour un cycle horloge afin de signaler au bloc `Traitement` la lecture de l'adresse de branchement par le processeur. `Ack_Read` est utilisé comme condition de transition de la machine état `Gestion adresse de branchement`.

```
1  action Interface_Read sur message nRST, EN, vect_priorite, vect_handle,
2  masque, pending, blx avec
3  (
4  entree var addr          : Def_addr;
5  entree var nCS_IT        : Def_Bit;
6  entree var nAS           : Def_Bit;
7  entree var RnW           : Def_Bit;
8  entree var nRST          : Def_Bit;
9  entree var EN            : Def_Bit;
10 entree var vect_priorite  : Def_vect_priorite;
11 entree var vect_handler   : Def_vect_handler;
12 entree var masque        : Def_masque;
13 entree var pending       : Def_pending;
14 entree var blx           : Def_blx;
15 sortie var d_bus         : Def_data;
16 sortie var Ack_Read      : Def_bit;
17 );
18
19 var TriState              : HauteImpedance;
20 const addr_EN             : Def_addr = 0x000000;
21 const addr_masque         : Def_addr = 0x000002;
22 const addr_pending        : Def_addr = 0x000004;
23 const addr_blx            : Def_addr = 0x000006;
24 const addr_vect_handler   : Def_addr = 0x00000A;
25 const addr_vect_priorite  : Def_addr = 0x000044;
26
27 begin
28
29 cycle:
30 begin
31   if (nRST=0) then
32     Ack_Read := 0;
33   end if;
34   d_bus := TriState;
35   Ack_Read := 0;
36   if (nCS_IT=0) et (RnW=1) et (nAS=0) then
37     case addr of
38       addr_EN:
39         d_bus := EN;
40       addr_masque:
41         d_bus := masque;
42       addr_vect_handler:
43         d_bus := vect_handler;
44       addr_vect_priorite:
45         d_bus := vect_priorite;
46       addr_pending:
47         d_bus := pending;
48       addr_blx:
49         d_bus := blx;
50         Ack_Read := 1;
51     end if;
52   end cycle;
53 end Interface_Read;
```

4.5.3 Algorithme Masquage

L'algorithme ci-dessous est celui du bloc Masquage. Le langage utilisé est également le Pascal. Le type Def_IT_xxx est un entier de 0 à 14.

```
1 action Masquage sur message nIT_xxx avec
2 (
3   entree var nIT_xxx      : Def_IT_xxx;
4   entree var mask         : Def_masque;
5   sortie var nIT_xxx_masked : Def_IT_xxx;
6 );
7
8 begin
9
10 cycle:
11 begin
12   nIT_xxx_masked := nIT_xxx and not mask;
13 end cycle;
14 end Masquage;
```

4.5.4 Algorithme Solutionneur de priorité

Il s'agit ensuite de l'algorithme représentant le comportement du sous-bloc Solutionneur de priorité. Lors du passage à l'état logique 1 d'une des interruptions, la priorité correspondante à cette IT est évaluée avec la priorité max (c'est-à-dire la priorité de l'interruption en cours). Si la priorité de la i-ème IT est supérieure à la priorité de l'IT en cours, celle-ci passe comme IT à traiter.

```
1 action Solutionneur_de_priorite sur message nIT_xxx avec
2 (
3   entree var nIT_xxx      : Def_IT_xxx;
4   entree var priority_vector : Def_vect_priorite;
5   sortie var ID_IT        : Def_ID;
6 );
7
8 var max_prio      : integer = 0;
9 var temp_ID_IT    : Def_ID = 0;
10 var IT_size       : integer = 15;
11 var ID            : Def_ID;
12
13 begin
14
15 cycle:
16 begin
17   for i := 0 to IT_size-1 do
18   begin
19     if nIT_xxx(i) = 1 then
20       if priority_vector(i) > max_prio then
21         max_prio := priority_vector(i);
22         temp_ID_IT := ID(i);
23       else if (priority_vector(i)=max_prio) and (i>temp_ID_IT) then
24         temp_ID_IT := ID(i);
25       end if;
26     end if;
27   end for;
28
29   ID_IT := temp_ID_IT;
30
31 end cycle;
32 end Solutionneur_de_priorite;
```



4.5.5 Algorithme Gestion adresse de branchement

Pour finir il reste l'algorithme de Gestion adresse de branchement. Il s'agit d'une machine état séquentielle finie chargée de signaler le CPU d'une IT à traiter, placer l'adresse de la routine d'interruption dans le registre blx et attendre la lecture. Il est possible de remarquer la structure typique d'une machine à état finie. Il y a le bloc de sortie de la machine à état finie qui est combinatoire puis le bloc calculant l'état suivant synchrone à l'horloge du système.

```
1  action Gestion_blx sur clk, message nRST, message state avec
2  (
3  entree var clk           : Def_bit;
4  entree var nRST          : Def_bit;
5  entree var ID_IT         : Def_ID;
6  entree var Ack_Read      : Def_bit;
7  entree var Is_IT_active  : Def_bit;
8  entree var vect_handler  : Def_vect_handler;
9  sortie var nIT_CPU       : Def_bit;
10 sortie var blx           : Def_addr;
11 );
12
13 var state                 : Def_state;
14 var prev_ID_IT           : Def_ID;
15
16 begin
17
18 cycle:
19 begin
20   if nRST = 0 then
21     state := WAIT_IT;
22     prev_ID_IT := 0;
23     nIT_CPU := 1;
24     blx := 0;
25   else
26     case state of
27       WAIT_READ :
28         prev_ID_IT := ID_IT;
29         nIT_CPU := 0;
30         blx := vect_handler(ID_IT);
31       NEXT_IT :
32         nIT_CPU := 1;
33     end if;
34   cycle H:
35   begin
36     case state of
37       WAIT_IT :
38         if is_IT_active = 1 then
39           state := WAIT_READ;
40         end if;
41       WAIT_READ :
42         if Ack_Read = 1 then
43           state := NEXT_IT;
44         end if;
45       NEXT_IT :
46         if prev_ID_IT <> ID_IT then
47           state := WAIT_READ;
48         else if
49           state := WAIT_IT;
50         end if;
51     end cycle H;
52   end cycle;
53 end Gestion_blx;
```

5 Validation

Cette partie vise à tester et valider le comportement des blocs fonctionnels. Elle se positionne sur la pente remontante du cycle en V. Comme le détaille, J.P. Calvez dans *Spécification et conception des systèmes - une méthodologie* [2], la seconde moitié du cycle commence par les éléments élémentaires pour remonter progressivement jusqu'au produit complet.

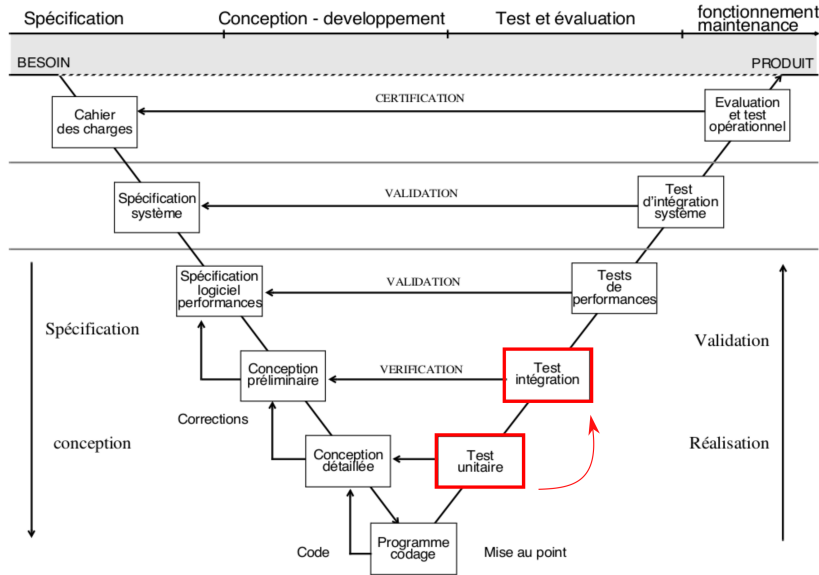


Figure 17: Exemple de cycle en V

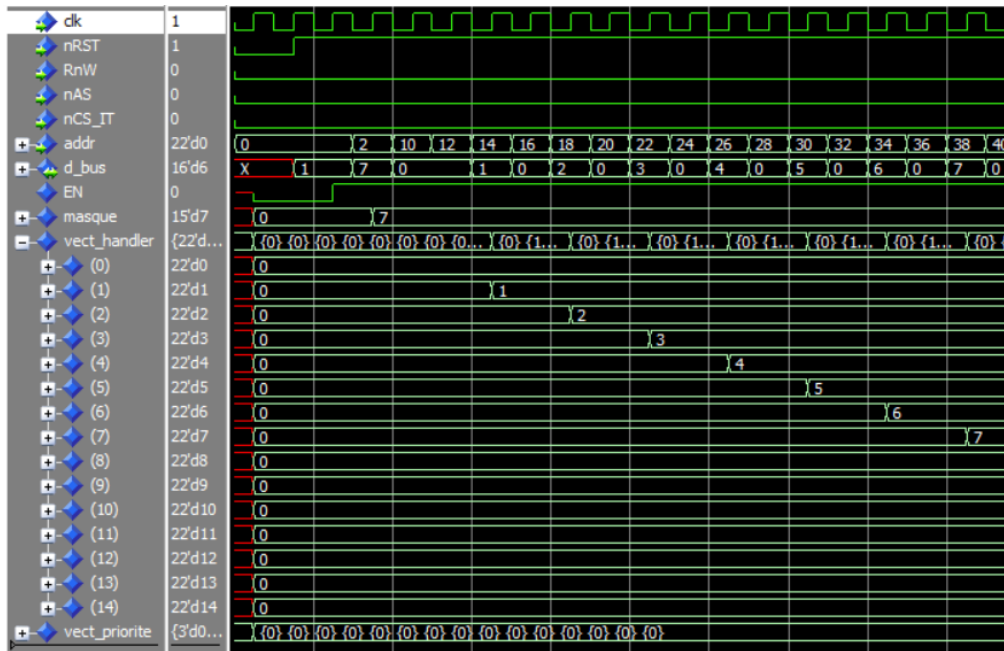
Dans un premier temps une attention particulière sera portée à vérifier le bon fonctionnement des blocs élémentaires de notre conception détaillée. Dans un second, un test plus proche des conditions d'utilisation se focalisera sur le périphérique avec toutes les entités intégrées. N'ayant aucune information sur la technologie à utiliser dans le cahier des charges, ce rapport se contentera de ces deux phases de test. En effet, il n'est pas possible de réaliser la partie test de performance où intégration système.

5.1 Test unitaire

Le test unitaire vise à isoler chaque bloc unitaire afin de vérifier son fonctionnement. Le comportement des blocs adjacents est alors simulé par le testbench. Dans ce contexte, il est possible de moduler les signaux d'entrée de manière plus flexible. Il serait naïf pour un ingénieur débutant de négliger cette étape en passant directement à la phase d'intégration. En effet, elle permet d'identifier un certain nombre d'erreurs et de les résoudre dans un environnement de test simplifié. Dans un cadre industriel cela permet de limiter les coûts d'intégration [2].

5.1.1 Interface Write

Le bloc d'interface write synchrone à l'horloge du système s'inscrit dans le bloc interface et permet la frontière entre les bus et signaux de contrôle du processeur et les registres et signaux internes du contrôleur d'interruptions. Il s'agit dans ce test unitaire de valider le bon fonctionnement de l'écriture de valeurs dans les registres. Cela est réalisé en plaçant une valeur à écriture dans le bus de données, et placer dans le bus d'adresse l'adresse du registre. Ainsi à chaque front montant de l'horloge, si les signaux nCS.IT, nAS et RnW sont à l'état bas alors l'écriture est effectuée.



Les chronogrammes ci-dessus représentent le résultat de la simulation de l'interface écriture. Au démarrage de la simulation, le signal de réinitialisation est actif à l'état bas. Il y a donc une demande de reset qui est prise en compte au premier front montant de l'horloge et inscrit pour chaque registre interne la valeur 0. Puis des cycles d'écriture sont réalisés dans l'ordre suivant, écriture dans le registre d'activation de l'IP EN, écriture dans le registre de masquage des IT et écriture successive dans le tableau des routines d'interruptions. Comme le montre la figure 18, l'état des registres est bien maintenu par des bascules et les valeurs sont cohérentes. Cela valide le fonctionnement de l'interface d'écriture.

5.1.2 Interface Read

L'interface read permet au processeur de lire l'état des registres. Pour valider le comportement de ce bloc, une étape d'écriture préalable est nécessaire. On utilise pour cela le test précédent. Il faudra ensuite vérifier que le bus de données prenne bien les valeurs contenues dans les registres lorsque l'adresse correspondante est sélectionné. Une attention particulière doit être portée sur la conformité des chronogrammes vis-à-vis des spécifications du cahier des charges. Une lecture ne doit prendre qu'un cycle horloge. Pour cette raison l'entité est asynchrone.

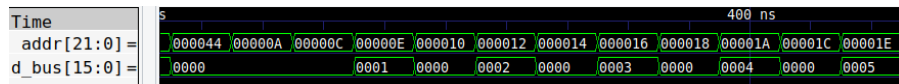


Figure 19: Chronogramme validation de l'interface read

La figure 19 présente le test de l'interface. On effectue une lecture des adresses de branchement écrites dans le test précédent. On observe les valeurs incrémentées de 1 et espacé d'une lecture à zéro. Cette lecture correspond aux bits de poids forts. Le comportement est validé en comparant les adresses et valeurs des bus du test précédent avec ceux de la figure 19. Les contraintes de temps sont également respectées, car la lecture s'effectue un 1 cycle horloge.

5.1.3 Solutionneur de priorités

Le bloc solutionneur de priorités prend en entrée les 15 sources d'interruptions provenant des autres périphériques et le registre des priorités. Il doit, à partir de cela, présenter en sortie le numéro de l'interruption active la plus prioritaire. La gestion des interruptions n'est pertinente, dans un processeur, que si la latence est réduite au minimum. C'est pourquoi nous avons fait le choix de concevoir une solution asynchrone qui assure sa fonction en moins d'un cycle horloge. La simulation suivante ne possède donc pas d'horloge. Les deux conditions de priorité sont testées. La première, si deux interruptions sont valides au même instant, c'est l'identifiant de la plus prioritaire au regard du registre `CTRL.IT_PRIO_N` qui est placé en sortie. La seconde, correspond au cas particulier où deux interruptions de même priorité seraient actives simultanément. Dans cette situation, c'est l'identifiant le plus grand qui est sélectionné.

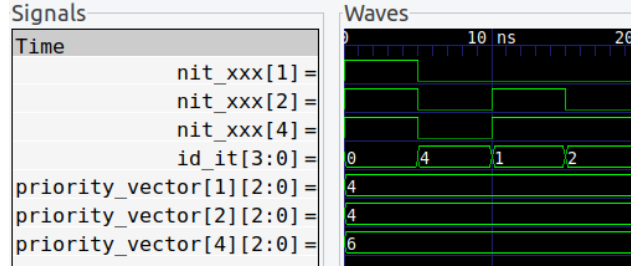


Figure 20: Chronogramme validation solutionneur priorité

La figure 20 illustre le résultat du bloc solutionneur priorité lorsqu'il est soumis à un banc de test. Bien que les stimulus d'entrée prennent un grand nombre de valeurs dans la simulation originale, seul trois cas sont présentés. Les priorités sont inchangées au cours de ce test et leur valeur sont $IT_4 = 6$, $IT_1 = 4$, $IT_2 = 4$. Dans le premier, trois interruptions sont valides simultanément. C'est l'identifiant 4 qui est placé en sortie, car sa priorité est de 6 contre 4 pour deux les autres. Dans le deuxième cas, seule une interruption est valide, il n'y a pas d'arbitrage, la valeur 1 est visible en sortie. Le dernier illustre la situation particulière où les 1 et 2 ont la même priorité. C'est donc le numéro 2 qui prend la main. Le chronogramme atteste du bon fonctionnement du bloc solutionneur de priorités.

5.1.4 Masque

Le bloc masque permet à l'utilisateur d'inhiber les sources d'interruption de son choix. Il prend en entrée les 15 sources d'interruption et le registre `CTRL.IT_MSQ`. Son fonctionnement est plutôt simple. Le registre est d'abord inversé bit à bit puis un ET logique est appliqué avec les sources. Seuls, les bits non masqués sont transmis au bloc suivant.

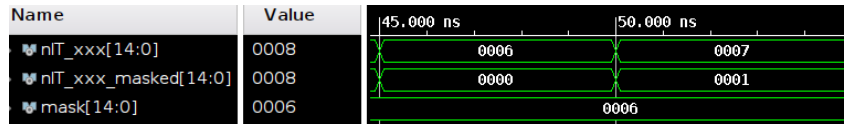


Figure 21: Chronogramme validation bloc masquage

La figure 21 illustre deux exemples du bloc masquage soumis aux stimulus du banc de test. Le masque est affecté à la valeur 6, $(110)_2$, pendant les deux situations. Les sources 1 et 2 sont donc masquées. Dans le premier cas, ces deux sources sont actives, elles sont donc inactivées sur la sortie. Dans un second temps, la source 0 émet une demande d'interruption. Cette dernière n'étant pas masquée, elle est conservée seule.

5.1.5 Gestion branchement (blx)

Le bloc gestion branchement est créé pour contrôler la temporalité des échanges avec le processeur de manière indirecte. Il génère le signal `nIT_CPU` et met à jour le registre de branchement au même instant. Il garantit la validité de sa valeur, en empêchant les réécritures tant que le processeur n'a pas confirmé la lecture. Si cela n'était pas fait, lorsqu'une interruption surviendrait entre la lecture du LSB et du MSB, l'adresse de branchement serait corrompue. Dans l'exemple présenté ci-dessous, deux interruptions surviennent de manière rapprochée. On considère que la seconde est plus prioritaire que la première et doit donc l'interrompre. Nous cherchons à montrer que cela n'est fait qu'après la prise en compte de la première.

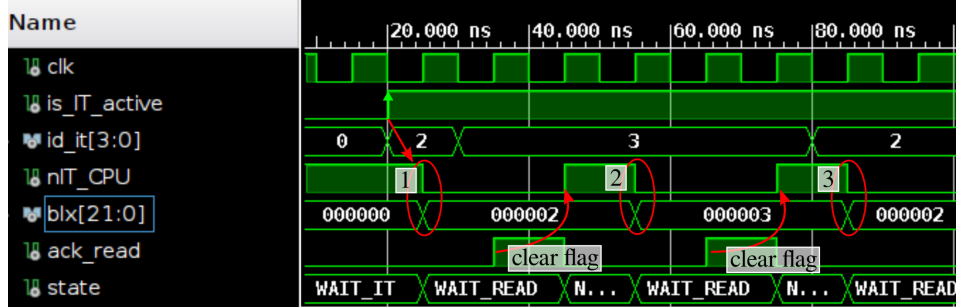


Figure 22: Chronogramme validation bloc gestion branchement

Le cas recherché est présenté figure 22. Le passage de l'entrée `is_IT_active` à l'état haut indique qu'une donnée valide est présente sur `id_IT` (cf [1]). Le gestionnaire de branchement réagit en plaçant l'adresse de branchement dans le registre `@blx` et en assignant `nIT_CPU` à 0. Les adresses de branchement sont configurées avec le numéro de l'IT pour plus de lisibilité. C'est pourquoi `@blx` vaut 2. Un court moment après, `id_IT` change de valeur, indiquant qu'une IT plus prioritaire survient. Cela n'engendre aucune réaction, car l'interruption précédente n'a pas été acquittée. Le processeur effectue une lecture dans le registre de branchement (LSB), le signal `ack_read` passe à 1. Pour rappel, `ack_read` est généré par le bloc interface read. Il y a bien eu acquittement, les données concernant la seconde interruption sont fournies (cf [2]). On observe `@blx = 3`. Ensuite, une nouvelle lecture est effectuée, on considère dans le testbench que l'interruption est traitée en 1 coup d'horloge. Le numéro d'interruption sur `id_IT` repasse donc à 2 pour finir de traiter la première. Le coup d'horloge suivant (cf [3]), l'adresse est placée dans le registre `@blx` et le CPU est informé via `id_IT`.

5.2 Test intégration

Les tests unitaires viennent d'être présentés. Comme présenté dans l'introduction de la validation, la prochaine étape selon le cycle en V (figure 17) est le test d'intégration. Il vise à rassembler les blocs fonctionnels tester dans la partie précédente dans une même entité. On veillera à les interconnecter suivant le résultat de la phase de conception.

5.2.1 Configuration

L'objectif de ce test est de simuler le comportement du processeur lors de la configuration. Deux périphériques sont activés, l'UART et le PCI. Dans un premier temps, le périphérique est désactivé. Puis les adresses de branchement sont configurées, l'une après l'autre. L'organisation des registres permet d'écrire la priorité de deux interruptions adjacentes en même temps. Nous exploitons donc cette possibilité dans cet exemple. La priorité de l'UART est fixée à 3 et celle du PCI à 5. Finalement, le périphérique est activé grâce au registre `Enable`.


```

1 void CTRL_IT_Config(void){
2     /* Disable CTRL IT */
3     CTRL_IT->CTRL_IT_EN = 0x0000;
4     /* Set handler addr UART */
5     CTRL_IT->CTRL_IT_ADDR[nIT_UART] = &UART_Handler;
6     /* Set handler addr PCI */
7     CTRL_IT->CTRL_IT_ADDR[nIT_PCI] = &PCI_Handler;
8     /* Set priority PCI = 5, UART = 3*/
9     CTRL_IT->prio.[nIT_PCI] = (uint16_t)((0x3 << 8) | 0x5);
10    /* Enable CTRL IT */
11    CTRL_IT->CTRL_IT_EN = 0x0001;
12 }

```

Une fois le comportement souhaité établi, il convient de le traduire en VHDL pour le testbench. Cette étape aboutie à une expansion importante du nombre de lignes. C'est pourquoi on ne présente ici la conversion d'une seule ligne afin d'exposer la méthode. La dernière ligne visant à activer le périphérique est choisie pour des raisons de simplicité.

```

1 -- Enable CTRL IT
2 -- In C : CTRL_IT->CTRL_IT_EN = 0x0001;
3 RnW <= '0'; nAS = '0';
4 addr <= addr_EN;
5 d_bus <= (0 => '1', others => '0');

```

Le comportement souhaité, sous-entendu par la ligne en C, est l'écriture d'un 1 dans le registre **Enable**. Les signaux de contrôle sont affectés de sorte à indiquer une écriture. L'adresse du registre est placée sur le bus d'adresse. Le bus de données est affecté à '1'. Cet exemple plutôt simple présente la méthode. La même démarche est mise en œuvre pour les autres lignes C. Dans la mesure où la donnée à transmettre est plus petite ou égale à la taille du bus de données, les instructions peuvent se faire en un cycle horloge. L'écriture des adresses de branchement sont donc faites en deux coups d'horloges. Le chronogramme 23 présente le comportement du processeur simulé par le testbench et le résultat du périphérique soumis à ces requêtes.

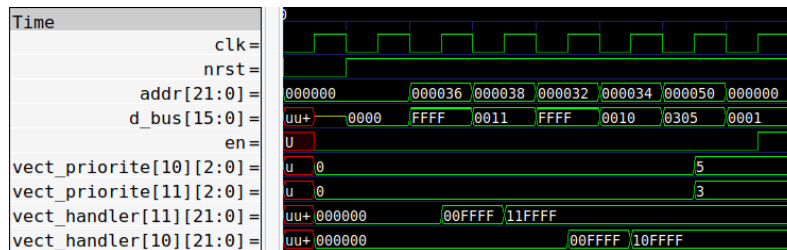


Figure 23: Chronogramme validation globale configuration

On observe les 7 cycles d'écriture attendue, dans l'ordre suivant :

- CTRL_IT_EN
- CTRL_IT_ADDR_11 LSB
- CTRL_IT_ADDR_11 MSB
- CTRL_IT_ADDR_10 LSB
- CTRL_IT_ADDR_10 MSB
- CTRL_IT_PRIO_10_11

Nous cherchons à vérifier que les cycles d'écritures aboutissent tous en une valeur stockée dans un registre. Cela est visible sur ce chronogramme. Le fonctionnement de ce test en bien celui souhaité. Par la même occasion, nous pouvons valider le bon fonctionnement du reset ayant lieu au début de la simulation. Tous les registres affichés prennent la valeur 0.

5.2.2 Interruptions simultanées

Une fois la configuration du périphérique effectuée, il est possible de tester sa réaction à l'occurrence d'interruptions. Nous choisissons de présenter un cas de complexité modérée. Deux interruptions de priorité différentes apparaissent simultanément. Le comportement attendu est le traitement de la plus prioritaire puis, après acquittement du CPU, la prise en charge de la seconde. La figure 24 présente le comportement du périphérique soumis à deux interruptions simultanées.

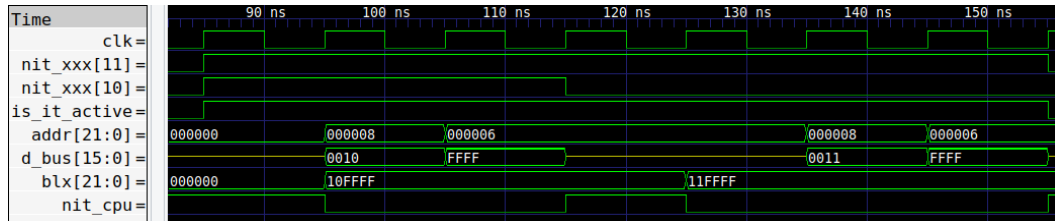


Figure 24: Chronogramme validation globale interruptions simultanées

Les signaux d'interruptions sont visibles au premier coup d'horloge. Elles sont accompagnées du signal `is_it_active` qui comme son nom l'indique passe à haut si une interruption doit être traitée. Le bloc gestion blx analyse, la requête fournie par le solutionneur de priorité et affecte la bonne valeur dans le registre de branchement en un cycle horloge. La valeur en question est l'adresse de `PCI_handler` que l'on reconnaît par le préfixe 10 comme son numéro d'interruption. Ce premier comportement satisfait nos attentes, c'est bien PCI la source la plus prioritaire. Le CPU ayant été notifié par le signal `nIT_CPU`, lit l'adresse de branchement. Cela correspond à deux cycles horloge, le registre du LSB en interprété comme acquittement. Il baisse ensuite la source d'interruption sur le périphérique émetteur. L'interruption PCI est considérée comme complètement traitée à 125ns sur la figure 24. À l'instant suivant (135ns), le processeur est notifié pour prendre en compte la seconde, initialement moins prioritaire. Comme pour la première, l'adresse de branchement est placée dans le registre `blx`. La lecture par le processeur prend deux cycles. Toutes les interruptions sont traitées à 155ns, le signal `is_it_active` repasse à zéro.

Le fonctionnement du périphérique est validé par ce test ayant pour ambition de s'approcher au maximum d'un cas réel.

6 Conclusion

Le projet SoC de l'option SETR était consacré à la conception d'une IP, plus particulièrement d'un contrôleur d'interruptions. Il s'agissait, en se basant sur un cahier des charges, d'effectuer les phases de spécifications et conception à l'aide de la méthode MCSE. La réalisation par l'écriture des algorithmes en langage de description matériel a été réalisée par la suite. Elles étaient accompagnées de bancs de test afin de valider le bon comportement individuel et collectif des blocs hiérarchiques de l'IP.

La spécification consistait en la caractérisation de l'environnement, définir les entrées et sorties et donner les fonctionnalités du circuit à concevoir. Cela a permis d'obtenir un comportement du contrôleur d'interruptions, sa structure mémoire interne à l'aide de la spécification des registres et l'emploi de l'IP. La conception était la mise en place des blocs et sous-blocs internes, leur organisation et leur comportement à l'aide d'algorithme haut-niveau.

Le principal avantage de la MCSE est de structurer convenablement la spécification et la conception, en raffinant au fur et à mesure. Une telle méthodologie est adaptée à la conception de circuit numérique. En effet, la conception de systèmes sur puce est basée sur la conception hiérarchique. C'est-à-dire que la bonne conception d'une IP (ici une IP soft car décrit au niveau RTL et non GDSII) repose sur sa décomposition en blocs eux-mêmes décomposés en sous-blocs. Cela permet de vérifier et valider individuellement puis en commun l'ensemble des blocs.

La suite de cette conception consisterait à émuler le contrôleur d'interruptions afin de valider son comportement sur une cible FPGA dotée d'un ensemble de ressources logiques. A l'aide d'outil comme Xilinx SDK, il serait par la suite possible de réaliser du codesign et valider le contrôleur d'interruptions d'un point de vue matériel et également logiciel. Si le souhait in fine est de graver le circuit sur silicium, des étapes d'implémentations physiques sont requises. Il s'agirait là de lancer une synthèse logique avec une technologie de cellules standards précises (45 nm par exemple). Suite à cela des boucles itératives d'analyse de temps et de consommation seraient effectuées sur différents scénarios PTV (Process de gravure, Température et Voltage) afin de corriger des violations de timing ou des surconsommations pouvant nuire au bon fonctionnement de l'IP.

Acronyms

CPU Central Processing Unit 6, 9, 11

IP Intellectual Property 6

TTL Transistor Transistor Logic 9

UART Universal Asynchronous Receiver Transmitter 20

References

- [1] J.P. Calvez. *Spécification et conception des systèmes - une méthodologie*. 1991.
- [2] J.P. Calvez. *Spécification et conception des systèmes - étude de cas*. 1991.