الجمهورية الجزائرية الديمقراطية الشعبية
République Algérienne démocratique et populaire

وزارة التعليم العالي و البحث العلمي
Ministère de l'enseignement supérieur et de la recherche scientifique

جامعة سعد دحلب البليدة
Université SAAD DAHLAB de BLIDA

كلية التكنولوجيا
Faculté de Technologie

قسم الإلكترونيك
Département d'Électronique

Mémoire de Projet de Fin d'Études

présenté par

OUABEL Amina

pour l'obtention du diplôme Master en Électronique option : ***system de vision et robotique***

Thème

Description VHDL d'un réseau de neurones impulsionnels générique (modèle Izhikevich)

Proposé par : Mme. BOUGHERIRA Hamida

Année Universitaire ***2014-2015***

Remerciement

Je tenais tout d'abord à remercier DIEU tout puissant qui m'a donnée durant toutes ces années la santé, le courage et la foi pour en arriver là.

J'exprime mes sincères remerciements à l'égard de Mme BOUGHERIRA.H pour avoir accepté d'encadrer ce travail, et pour son aide et ses précieux encouragements, et Mr RELLAM.W qui m'a suivi pendant cette thèse pour la mise en œuvre de ce modeste travail.

Mon infinie gratitude va à tous mes enseignants qui ont contribué à ma formation et à tous les enseignants du département d'électronique et aux membres du jury qui ont accepté d'évaluer mon travail.

Un grand merci à Mr Naamane, Mr Saidani Abderrahmane...

Je tenais aussi à remercier ma famille en particulièrement mes parents , mes frères, ma jumelle, sans oubliant mon mari et sa famille, et mes chère-sœurs Kawther et Asma…

Finalement, je remercie tous ceux qui ont participé de prés ou de loin à l'achèvement de ce travail.

Dédicaces

Je dédié ce modeste travail à ma famille, particulièrement à ***ma mère, mon père, ma jumelle et mes frères,*** *et ma petite sœur* ***Asma,***

Je n'oublie pas ma belle famille, en particulier mon mari ***Ishak*** *pour son aide et ses précieux encouragements,*

J'offre ce travail aussi à tous mes amis surtout mon amie ***Kawther****…pour leur soutien et l'assistance permanente…*

J'offre ce travail aussi, à mon oncle Saidani Abderahmane pour son aide et son soutien…

Résumé

L'objectif de ce travail consiste à faire une description en VHDL d'un réseau de neurones impulsionnels (modèle Izikhevich), dans le but de construire un outil EDA intégrable à la bibliothèque du langage VHDL. Nous avons implémenté, le modèle du neurone impulsionnel Izhikevich en VHDL en utilisant deux versions (réel et virgule fixe). Une fois les résultats concluants pour un neurone impulsionnel d'Izhikevich, nous avons constitué un package pour un neurone générique.

Les mots clés :

Spiking neurones, FPGA, VHDL, modèle Izhikevich

Abstract

The objective of this work is to make a VHDL description of a spiking neural network (Izikhevich model) in order to build an integrated EDA tool in the library of VHDL. We implemented the pattern of spiking neuron in VHDL using two versions (real and fixed-point). After the conclusive results for a spiking neuron Izhikevich, we formed a package for a generic neuron.

ملـخـص

الهدف من هذا العمل هو وصف الشبكة العصبية النبضية بواسطة اللغة التقنية (في أش دي أل) من أجل بناء آلية التصميم الإلكتروني مدرجة في مكتبة هذه اللغة التقنية.

نفذنا نموذج الخلية العصبونية النبضية (ايـزيـكـيـفـيـتـش) بواسطة اللغة التقنية (في أش دي أل) باستخدام نسختين (الأعداد الحقيقية وأعداد ذو نقطة ثابتة). وبمجرد محاكاة ناجحة لهذا الوصف, نقوم بإنشاء خلية عصبونية عامة .

Table des matières

Liste des figures

Liste des Tableaux

Introduction générale

S'il est possible de considérer le cerveau comme un calculateur, il est essentiel de comprendre comment les neurones encodent l'information et comment les assemblées de neurones traitent cette information, pour permettre de développer de nouvelles approches en traitement de l'information. Cette rencontre entre neurosciences et informatique constitue un courant appelé neurosciences computationnelles

Les modèles de neurones impulsionnels, constituent une nouvelle vague connexionniste, et une alternative aux réseaux de neurones classiques (dits de deuxième génération), pour une modélisation neuronale plus proche du modèle biologique. En effet, ce type de modèle prend en compte le décours temporel du potentiel membranaire d'un neurone et modélise explicitement l'émission de ses potentiels d'action.

Objectifs

Cette étude est motivée par le désir de développer une plateforme de traitement de l'information par les réseaux de neurones impulsionnels, qui sera utilisée dans le développement de modèles de traitement de l'information dans le domaine de la vision artificielle. Les modèles neuromimétiques à codage impulsionnel peuvent combiner les avantages des techniques analogiques pour le calcul (compacité et faible consommation des composants VLSI) avec la robustesse des techniques digitales pour la transmission de l'information. Ces modèles sont donc particulièrement attractifs pour des applications embarqués et des implantations matérielles.

Toutefois, il n'existe pas actuellement de modèle de réseau de neurones impulsionnels standard, et en particulier aucun algorithme d'apprentissage parfaitement adapté, ce qui freine leur utilisation dans des applications réelles. Les recherches actuelles demandent des travaux de simulation de réseaux de grandes tailles, combinant des résultats théoriques et des connaissances biologiques, afin d'obtenir des systèmes complètement fonctionnels.

L’objectif de ce travail consiste à faire une description en VHDL d’un réseau de neurones impulsionnels générique (modèle Izikhevich), dans le but de construire un outil EDA intégrable à la bibliothèque du langage VHDL, afin de reproduire un processus de perception de couleurs par la rétine.

Plan de la thèse

La thèse est basée sur deux parties principales : la partie théorique et la partie pratique.

La partie théorique contient deux chapitres : dans le premier chapitre nous allons débuter par une étude bibliographique sur le fondement biologique des réseaux de neurones impulsionnels. Ensuite, nous allons présenter un aperçu sur les modèles généraux de ces réseaux et déduire notre choix de modèle neuronal, et dans le deuxième chapitre, nous allons présenter quelques notions de base sur les circuits FPGA, et la méthodologie de conception de ces circuits.

La partie pratique contient aussi deux chapitres : nous allons présenter dans le troisième chapitre la méthodologie de la conception de notre circuit, et la description VHDL de notre modèle *IZHIKEVICH.* Et dans le dernier chapitre, nous allons présenter et discuter les résultats que nous avons obtenus dans notre travail.

Partie Théorique

Chapitre (I)

Spiking Neurones

I.1.Introduction

Environ la moitié du néocortex des primates non humains est couverte par des aires visuelles. Chez l'Homme, on estime qu'elles représentent 20 à 25 % du cortex, et qu'y opèrent environ 5 milliards de neurones.

Parmi les neurones artificiels, les réseaux de neurones de 3éme génération dits impulsionnels sont ceux qui se rapprochent le plus du système neuronal biologique.

Les neurones impulsionnels prennent en compte une caractéristique fondamentale des neurones biologiques : la capacité d'encoder l'information sous forme d'événements discrets. Nous nous sommes intéressées à l'apport de ce type de modèles dans le cadre d'un projet de recherche (équipe bio-rétine, Université de Blida) qui vise à modéliser le processus de perception de couleurs par la rétine ; les contraintes du projet nous ont orientés vers le choix de modèles simples, adaptés à la rapidité de traitement requise. Nous justifions ainsi notre choix des neurones impulsionnels par deux de leurs avantages : un réalisme biologique suffisant, et une complexité calculatoire satisfaisante

I.2.Les neurones impulsionnels

Les premiers modèles, fondés sur le neurone de Mc Culloch et Pitts, étaient fondés sur des sorties digitale et synchrones (neurones binaires). La seconde génération de modèles a défini des sorties analogiques et toujours synchrones, permettant des tâches plus complexes de régression et de classification probabiliste. La troisième génération de modèles (Maass, 1997), est basée sur les neurones impulsionnels où le temps devient un élément explicite du modèle et l'activité est digitale et asynchrone.

Il est connu que les neurones biologiques sont des systèmes dynamiques à temps continu qui communiquent par l'intermédiaire d'impulsions, appelées potentiels d'action ou « spikes ». Ainsi, les neurones déchargent avec une latence plus ou moins longue et dépendante de l'activation qu'ils reçoivent. Avec ce type de codage, la transformation se fait du domaine analogique vers le domaine temporel.

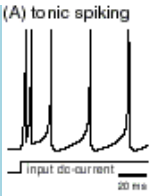
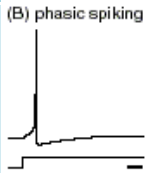
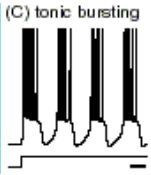
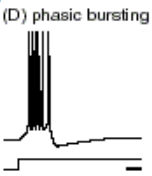
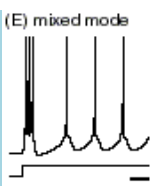
Un important domaine de recherche en neuroscience tend d'élucider le moyen par lequel les neurones biologiques encodent l'information sensorielle. Il était jusqu'alors généralement accepté que seule la fréquence d'émission des impulsions véhicule l'information utile. Dans un contexte de traitement rapide de l'information, de nombreuses données expérimentales ont cependant mis en évidence le rôle de la structure temporelle des impulsions dans le codage neuronal. D'autres types de codage ne prenant en compte que quelques spikes ont alors été proposés comme alternatives, par exemple un codage par ordre de décharge utilisant l'asynchronisme entre les neurones artificiels basés sur ce codage a montré qu'il est possible d'effectuer des taches de reconnaissance visuelle extrêmement rapidement avec seulement un spike par neurone.

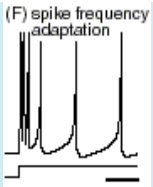
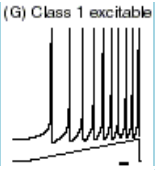
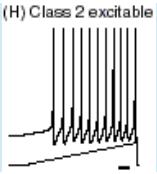
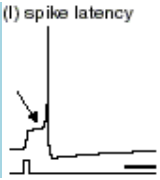
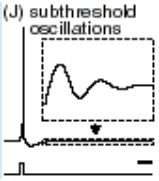
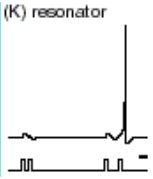
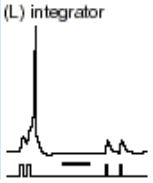
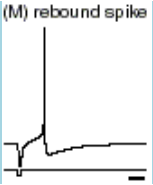
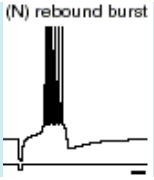
I.2.1.Les modèles biophysiques (les comportements dynamiques des neurones impulsionnels)

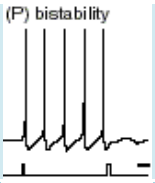
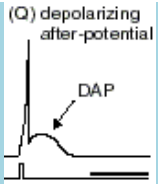
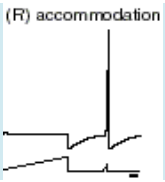
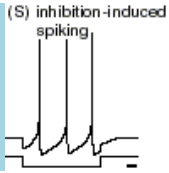
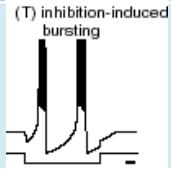
Les modèles computationnels de neurone impulsionnels ont l'avantage de proposer une intégration temporelle de l'information. La réponse du neurone à une stimulation n'est plus une valeur représentant le taux de décharge (principe du neurone à seuil ou sigmoïdal), mais le temps d'émission du prochain spike. L'information transmise n'est plus cadencée par la présentation d'un stimulus. Un spike est émis à l'instant où l'excitation du neurone dépasse le seuil. Il est donc nécessaire d'adopter une horloge virtuelle permettant, au sein d'un RNA, de temporiser les émissions de spikes des neurones, qui peuvent alors être considérées comme des événements datés.

Les modèles de neurone impulsionnels possèdent la propriété de détecter la corrélation temporelle des potentiels d'action afférents, surpassant en cela les modèles de neurone classiques.

Pour tester cette propriété, on injecte des impulsions de Courant Continu via une électrode attachée au neurone, et on affiche son potentiel de membrane.

Les modèles biophysiques	
Tonic Spiking : **La plupart des neurones de ce modèle sont excitables, ce qui signifie qu'ils sont inactifs, mais ils peuvent tirer des spikes lorsqu'ils sont stimulés. Tandis que l'entrée est active, le neurone continue à tirer un train des spikes.**	
PhasicSpiking : **Un neurone peut tirer un seul spike lorsque l'entrée est active, ensuite il redevient inactif.**	
Tonic Bursting : **Certains neurones tirent des rafales périodiques de spikes lorsqu'il est stimulé.**	
PhasicBursting : **neurones sont rupteurs phasiques : Ces neurones signalent le début de la stimulation par la transmission d'une salve.**	
Mixed Model (Bursting Then Spiking) : **débute par Phasic-Spiking et continue par Tonic-Spiking et iln'est pas mesurable**	

Spike Frequency Adaptation **les neurones de ce modèle tirent des pointes toniques avec la diminution de la fréquence. Autrement dit, la fréquence est relativement élevée au début de la stimulation, puis elle s'adapte.**	
Class 1 Excitability	
Class 2 Excitability	
Spike Latency	
Subthreshold Oscillations	
Frequency Preference and Resonance	
Integration and Coincidence Detection	
Rebound Spike	
Rebound Burst	
Threshold Variability	

Bistability of Resting and Spiking States	
Depolarizing After-Potentials	
Accommodation	
Inhibition-Induced Spiking	
Inhibition-Induced Bursting	

Tableau(I.1) : les modèles biophysiques des neurones impulsionnels

I.2.2.Les modèles des neurones impulsionnels

Un modèle impulsionnel est un modèle dont l'objet est de décrire la série d'impulsions générée par un neurone, une impulsion étant considérée comme instantanée et binaire. Autrement dit, la «sortie» du modèle est une suite d'instants, le temps des potentiels d'action, ce qui illustre la caractéristique fondamentale de ces neurones : l'entrée et le potentiel d'action, plus rapide, sont traités séparément. L'objet du modèle n'est pas de comprendre comment le potentiel d'action est généré, mais de décrire les séries d'impulsions produites en fonction de l'entrée.

On peut réunir tous les modèles impulsionnels au sein d'un cadre général. En effet, ils sont tous d´écrits par une équation différentielle scalaire non autonome :

$$\frac{dV}{dt} = f(t, V)$$

Avec un mécanisme de réinitialisation : lorsqu'à un instant « t », V(t) atteint le seuil (V)-seuil, il est réinitialisé à (V)-reset. On dit alors qu'une impulsion est produite au temps « t ». C'est à ces temps d'impulsion que nous nous intéressons.

Dans le tableau suivant, nous présentons les modèles de neurones impulsionnels et leurs équations :

Le modèle	Les équations
Intégrate&Fire (I&F)	$$v' = I + a - bv,$$ $$if\ v \geq vthresh, \qquad then\ v \leftarrow c$$
I&F with Adaptation	$$v' = I + a - bv + g(d - v)$$ $$\mathrm{g}' = \frac{(\mathrm{e}\delta\mathrm{t} - \mathrm{g})}{\tau}$$
Integrate-and-Fire-or-Burst	$$v' = I + a - bv + gH(v - vh)h(vT - v),$$ $$if\ v \geq vthresh, \qquad then\ v \leftarrow c$$
Resonate-and-Fire	$$z' = I + (b + w)z,$$ $$if\ Imz \geq athresh, \qquad then\ z \leftarrow z0(z)$$
Quadratic I&F	$$v' = I + a(v - vrest)(v - vthresh),$$ $$if\ v \geq vpeak, \qquad then\ v \leftarrow vrest$$
Spiking Model by Izhikevich (2003)	$$v' = 0.04v^2 + 5v + 140 - u + I$$ $$u' = a(bv - u)$$ $$if v \geq +30mV then \begin{cases} v \leftarrow c \\ u \leftarrow u + d \end{cases}$$
FitzHugh–Nagumo	$$v' = a + bv + cv^2 + +dv^3 - u$$ $$u' = \varepsilon(ev - u)$$
Hindmarsh–Rose	$$v' = u - F(v) + I - w$$ $$u' = G(v) - u$$ $$\mathrm{w}' = \frac{(\mathrm{H(v)} - \mathrm{W})}{\tau}$$
Morris–Lecar	$\mathrm{Cm}\frac{\mathrm{dV}}{\mathrm{dt}} = \mathrm{I} - \mathrm{g_L}\,(\mathrm{V}-\mathrm{V_L}) - \mathrm{g_{Ca}M_{ss}}(\mathrm{V}-\mathrm{V_{Ca}}) - \mathrm{g_K}\,(\mathrm{V}-\mathrm{V_K})$

Wilson Polynomial Neurons	$Cm\frac{dV}{dt} = - G_{Na} (V_m-E_{Na}) - G_{Na} (V_m-E_K) - G_{Na} (V_m-E_{Ca}) - G_{Na} (V_m-E_H) + I_{int}(t)$
Hodgkin–Huxley	$\frac{1}{\gamma(T)}\frac{dm}{dt} = \alpha m(V)(1-m) - \beta m(V)m = \frac{m\infty(V)-m}{\tau m(V)}$ $Cm\frac{dV}{dt} = -gNa\, m^3\, h\, (V - ENa) - gK\, n^4(V - EL) gL(V - EL) + I$

Tableau(I.2) : les modèles de neurones impulsionnels et leurs équations

Ces modèles sont importants non seulement parce que leurs paramètres sont bio-physiquement significatifs et mesurables, mais aussi parce qu'ils nous permettent d'étudier les questions liées à l'intégration synaptique, le filtrage de câble dendritique, les effets de la morphologie dendritique, l'interaction entre les courants ioniques, et d'autres questions liées à seule la dynamique des cellules.

Models	biophysically meaningful	tonic spiking	phasic spiking	tonic bursting	phasic bursting	mixed mode	spike frequency adaptation	class 1 excitable	class 2 excitable	spike latency	subthreshold oscillations	resonator	integrator	rebound spike	rebound burst	threshold variability	bistability	DAP	accommodation	inhibition-indiced spiking	inhibition-induced bursting	chaos	# of FLOPS
integrate-and-fire	-	+	-	-	-	-	-	+	-	-	-	-	+	-	-	-	-	-	-	-	-	-	5
integrate-and-fire with adapt.	-	+	-	-	-	-	+	+	-	-	-	-	+	-	-	-	-	+	-	-	-	-	10
integrate-and-fire-or-burst	-	+	+		+	-	+	+	-	-	-	-	+	+	+	-	+	+	-	-	-		13
resonate-and-fire	-	+	+	-	-	-	-	+	+	-	+	+	+	+	-	-	+	+	+	-	-	+	10
quadratic integrate-and-fire	-	+	-	-	-	-	-	+	-	+	-	-	+	-	-	+	+	-	-	-	-	-	7
Izhikevich (2003)	-	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	13
FitzHugh-Nagumo	-	+	+	-		-	-	+	-	+	+	+	-	+	-	+	+	-	+	+	-	-	72
Hindmarsh-Rose	-	+	+	+			+	+	+	+	+	+	+	+	+	+	+	+	+	+		+	120
Morris-Lecar	+	+	+	-		-	-	+	+	+	+	+	+	+		+	+	-	+	+	-	-	600
Wilson	-	+	+	+			+	+	+	+	+	+	+	+	+		+	+					180
Hodgkin-Huxley	+	+	+	+			+	+	+	+	+	+	+	+	+	+	+	+	+		+		1200

Figure(I.1) : Comparaison des propriétés neuro-informatique de dopage et l'éclatement des modèles

Nous décrivons ci-après les détails du modèle ***Spiking Model by Izhikevich (2003)*** car c'est celui que nous avons utilisé pour les travaux décrits dans ce mémoire.

On a choisi ce modèle parce qu'il permet de reproduire une vingtaine de comportements dynamiques différents, ce qui le rend précieux pour les modélisateurs qui souhaitent que leurs modèles restent proches des observations biologiques.

I.3.Le modèleI zhikevich

Le modèle Izhikevich basé sur deux équations différentielles couplées et inspiré du modèle de Fitzhugh-Nagumo (FitzHugh1961, Nagumo et al.1962), a ouvert l'accès à la reproduction de nombreux comportements observés sur les neurones biologiques. Ce modèle est décrit par les équations :

$$v' = 0.04v^2 + 5v + 140 - u + I(1)$$

$$u' = a(bv - u)(2)$$

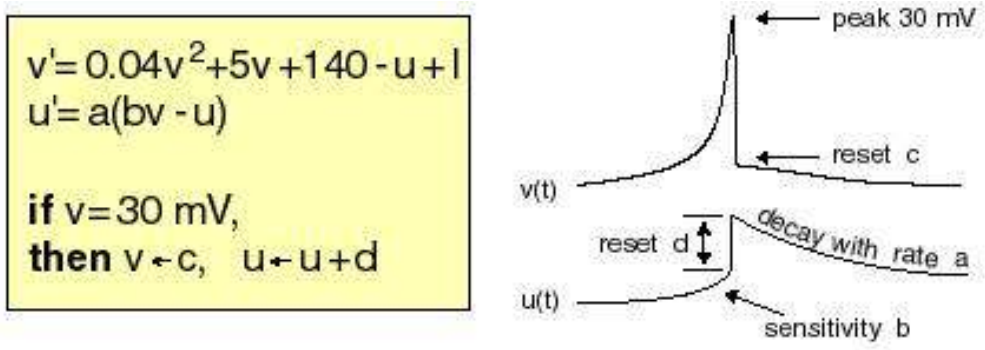
Avec :

$$if\ v \geq +30mV \qquad then \begin{cases} v \leftarrow c \\ u \leftarrow u + d \end{cases} \qquad (3)$$

Ici, **(v)** et **(u)** sont des variables sans dimension, et a, b, c et d sont des paramètres sans dimension.

Où **(v)** et **(u)** correspondent respectivement au potentiel et à la récupération de la membrane selon (Izhikevich 2003).

Après que le pic atteint son sommet (+30 mV), la tension de la membrane et la variable de rétablissement sont remis à zéro en fonction de l'équation (3).



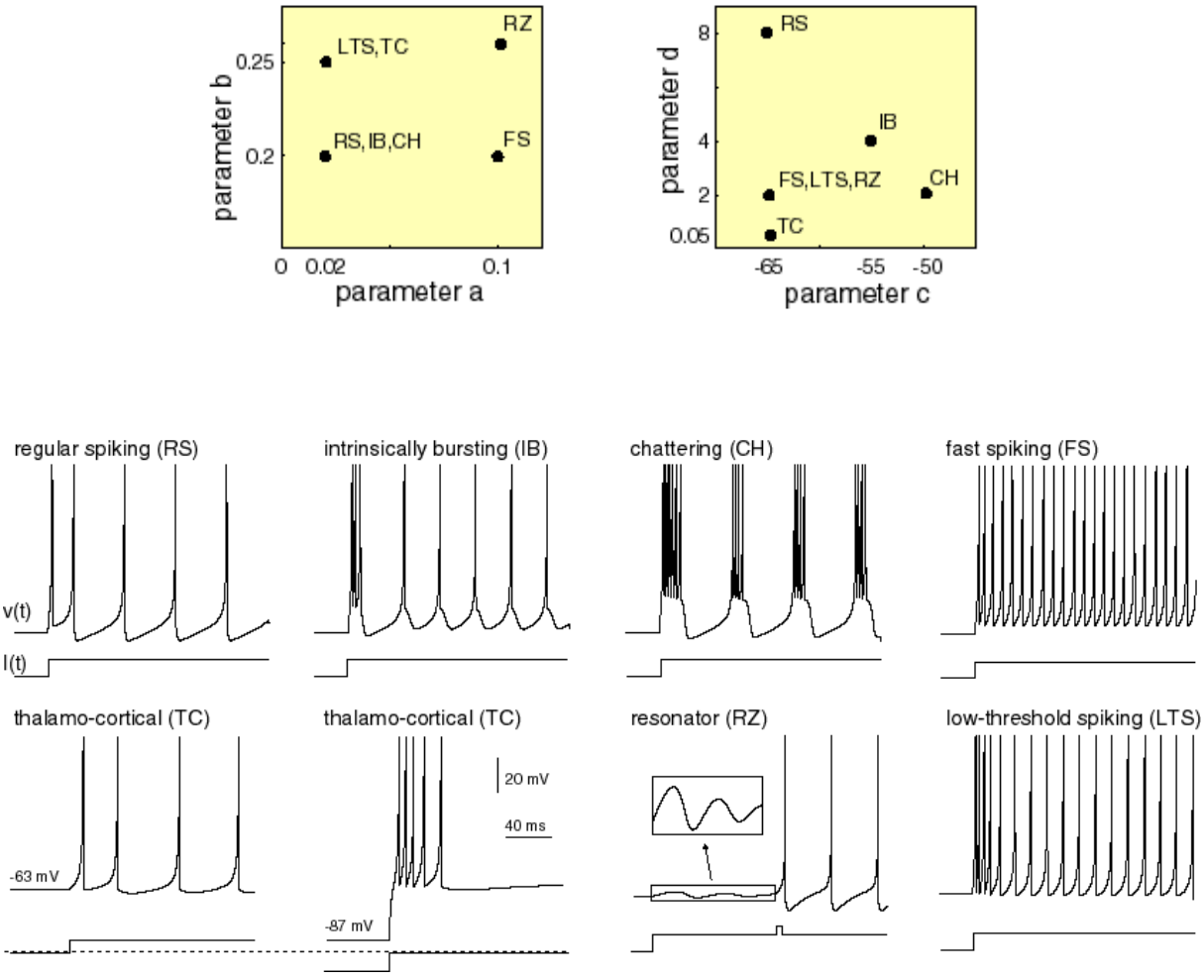
Figure(I.2) : Les paramètres du modèle Izhikevich

- **Le paramètre (a)** décrit l'échelle de temps de la variable u de récupération. Les petites valeurs se manifestent dans une récupération plus lente. Sa valeur typique est de 0.02.

- **Le paramètre (b)** décrit la sensibilité de la variable de récupération (u). Sa valeur typique est de 0.2.

- **Le paramètre (c)** décrit la valeur la réinitialisation après-pic de la variable –potentiel de la membrane (v). Sa valeur typique est de -65.

- **Le paramètre (d)** décrit la réinitialisation après-pic de la variable de récupération (u). Sa valeur typique est de 2.



Figure(I.3) : certains types de neurones correspondent à différentes valeurs des paramètres

En faisant varier les quatre paramètres a, b, c et d, on peut obtenir la plupart des réponses neuronales connues (voir tableau 2), ce qui rend ce modèle particulièrement intéressant.

Model	A	B	c	D	I
(A) tonic spiking	0.02	0.2	-65	6	14
(B) phasic spiking	0.02	0.25	-65	6	0.5
(C) tonic bursting	0.02	0 .2	-50	2	15
(D) phasic bursting	0.02	0.25	-55	0.05	0.6
(E) mixed mode	0.02	0.2	-55	4	10
(F) spike frequency adaptation	0.01	0.2	-65	8	30
(G) Class 1	0.02	-0.1	-55	6	0
(H) Class 2	0.2	0.26	-65	0	0
(I) spike latency	0.02	0.2	-65	6	7
(J) sub-threshold oscillations	0.05	0.26	-60	0	0
(K) resonator	0.1	0.26	-60	-1	0
(L) integrator	0.02	-0.1	-55	6	0
(M) rebound spike	0.03	0.25	-60	4	0
(N) rebound burst	0.03	0.25	-52	0	0
(O) threshold variability	0.03	0.25	-60	4	0
(P) bi-stability	1	1.5	-60	0	-65
(Q) DAP	1	0.2	-60	-21	0
(R) accomodation	0.02	1	-55	4	0
(S) inhibition-induced spiking	-0.02	-1	-60	8	80
(T) inhibition-induced bursting	-0.026	-1	-45	0	80

Tableau(I.3) : Les différentes réponses qui peuvent reproduire (de ce modèle)

I.4.Conclusion

Dans ce chapitre, nous avons présenté les neurones impulsionnels, et les différents modèles biophysiques des neurones impulsionnels. Ensuite nous avons considéré divers modèles de neurones impulsionnels et classé selon : le nombre de fonctions neuro-computationnelles qu'ils peuvent reproduire, et l'efficacité de leur mise en œuvre.

A la fin, nous avons présenté en détail le modèle que nous avons choisi **«*Spiking Model by Izhikevich (2003)*»** et que nous avons utilisé pour les travaux décrits dans ce mémoire, en justifiant ainsi notre choix des neurones impulsionnels.

Nous allons implanter ce modèle de réseau de neurones sur un circuit FPGA, pour cela, nous allons présenter dans le chapitre suivant quelques notions sur les circuits FPGA, et la description en langage VHDL.

Chapitre (II)

Les Circuits FPGA

II.1. Introduction :

L'implémentation matérielle (hard) de tout algorithme mathématique, est beaucoup plus puissante et plus rapide (x10000, x100000…), que son implémentation logicielle. Les circuits logiques programmables, en particulier les FPGA, et les outils de développement qui leur sont associés (VHDL, ISE, QUARTUS…), permettent d'élaborer un circuit intégré numérique personnalisé, de n'importe quel algorithme. Le but de notre travail est d'implanter un réseau de neurones impulsionnels (le modèle Izhikevich) sur circuit FPGA, en passant par une description en langage VHDL. Donc nous allons présenter dans ce chapitre quelques notions sur les circuits FPGA.

II.2. Généralités sur FPGA :

II.2.1. Historique :

Les premiers circuits intégrés reconfigurables de type **PFGA** (Field Programmable Gate Arrays) ont été commercialisés par la société Xilinx en 1985. Depuis cette date, d'autres concurrents sont apparus sur un marché qui n'a pas cessé de croitre. Au fur et à mesure que la complexité de FPGA s'est développée, leurs possibilités d'emploi se sont accrues, jusqu'à concurrencer sérieusement les circuits spécifiques pour des petits volumes de production (jusqu'à quelque milliers de composants). C'est pourquoi nous envisageons de plus en plus de développer l'utilisation des FPGA dans le cadre des applications spécifiques. Ils apportent une grande facilité d'emploi et permettent une réduction du coût de développement de l'électronique embarqué.

II.2.2. Définition d'un FPGA :

FPGA (Field Programmable Gate Arrays) est un circuit intégré composé d'un réseau de cellules programmables. Chaque cellule est capable de réaliser une fonction désirée.

II.3. Architecture interne des FPGA :

Un FPGA appelé aussi ***LCA*** (*Logic Cell Array*) signifiant réseau de cellules logiques, c'est un composant électronique qui contient un grand nombre d'éléments logiques programmables reliés entre eux par une matrice de routages programmables. Les FPGA sont composés de blocs logiques programmables ***CLB*** et de blocs d'entrées/sorties ***IOB*** positionnées sur la périphérie du composant.

La structure d'un FPGA diffère d'un constructeur à un autre mais elle garde la même architecture globale illustrée par la figure suivante.

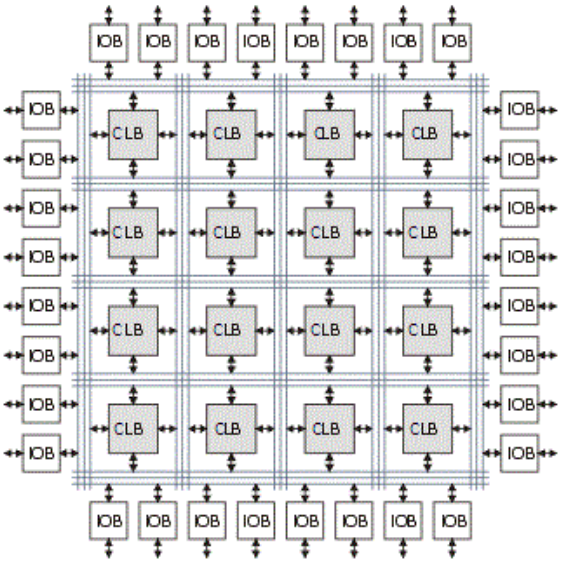


Figure (II.1) : Structure globale d'un FPGA

L'architecture d'un FPGA pour la famille Xilinx se présente sous forme de deux couches :

- Une couche active appelée circuit configurable
- Une couche de configuration

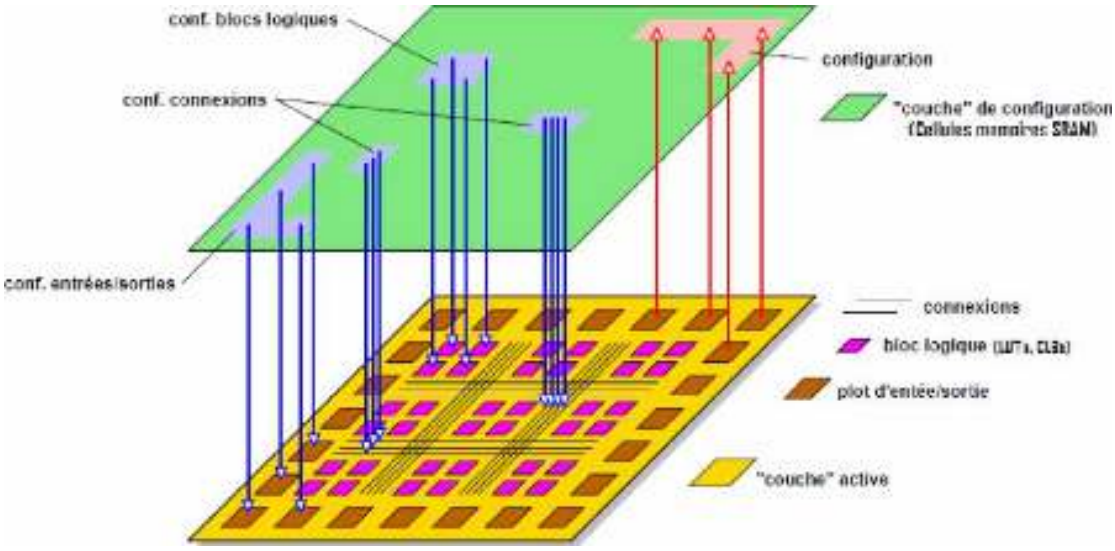


Figure (II.2) : Architecture d'un FPGA

II.3.1. La couche active :

Une couche active ou couche logique qui comprend des ressources logiques, des entrées/sorties, des sources de routage, et éventuellement des blocs dédiés.

II.3.1.1. Les blocs logiques programmables :

Les cellules CLB nous permet d'implémenter des fonctions logiques combinatoires ou séquentielles complexes car chaque CLB est constitué d'une partie combinatoire et d'une partie séquentielle. Chaque fonction est décomposée en petites fonctions booliennes qui peuvent être contenues par des cellules SLICES. Ces derniers comportent des LUT pour la partie calculatoire qui sont des petites mémoires de 2^nmots de 1 bit, adressées par n bits provenant de la matrice de routage et une ou des bascules (généralement de type D «flip-flop») contrôlées par un signal horloge pour la partie de mémorisation. Chaque CLB est composé de quatre slices repartis sur deux colonnes, chaque slice étant composé à son tour de deux cellules logiques (Logic Cell) disposées en colonne.

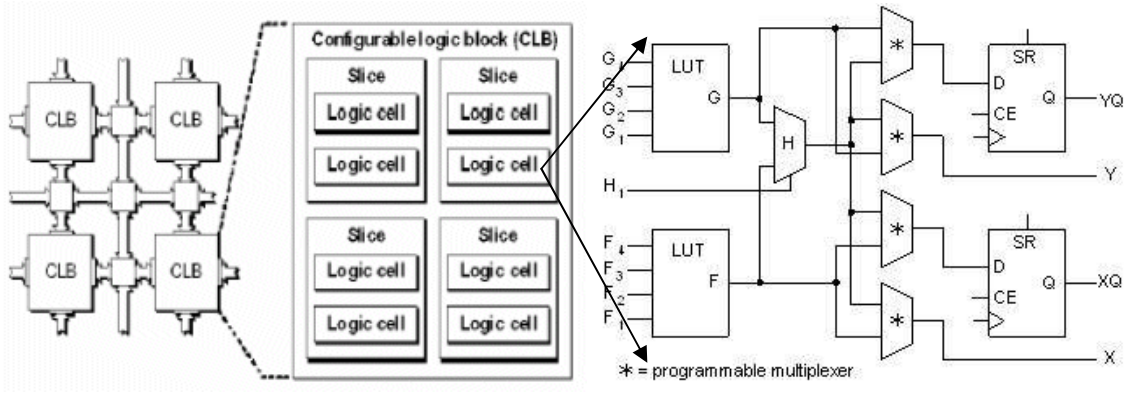


Figure (II.3) : Architecture d'un CLB et de sa cellule logique de type SPARTAN

II.3.1.2. Les blocs d'entrée/sortie IOB (Input/Output Blocks) :

Les blocs d'entrées/sorties sont positionnées à la périphérie du circuit FPGA. Ces IOB permettent l'interface entre les broches du composant FPGA et la logique interne developpée à l'intérieur du composant, et chaque IOB peut etre configuré en entrée, en sortie, en signal bidirectionnel ou être inutilisé (état haut impédance).

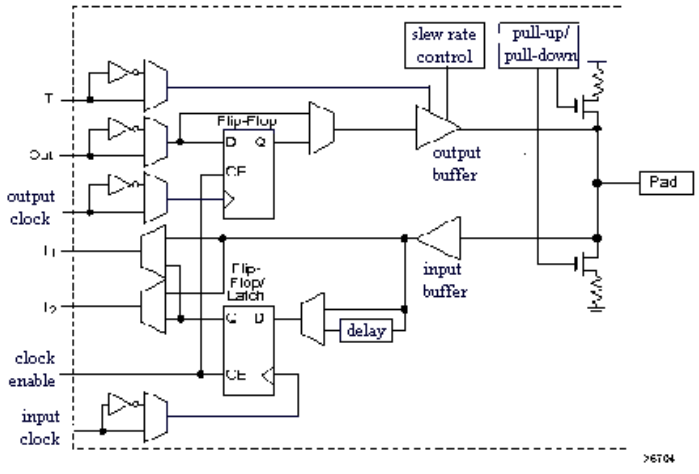


Figure (II.4) : Schéma d'un bloc d'entrée/sortie

- **Configuration en entrée :**

Premièrement, le signal d'entrée traverse un buffer qui, selon sa programmation, peut détecter soit des seuils TTL, soit des seuils CMOS. Il peut être routé directement sur une entrée directe de la logique du circuit FPGA ou sur une entrée synchronisée. Cette synchronisation est réalisée à l'aide d'une bascule de type D, le changement d'état peut se faire sur un front montant ou descendant. De plus, cette entrée peut être retardée de quelques nanosecondes pour compenser le retard pris par le signal d'horloge lors de son passage par l'amplificateur. Le choix de la configuration de l'entrée s'effectue grâce à un multiplexeur (Program Controlled Multiplexer). Un bit positionné dans une case mémoire commande ce dernier.

- **Configuration en sortie :**

Nous distinguons les possibilités suivantes :

- Inversion ou non du signal avant son application à l'IOB.
- Synchronisation du signal sur des fronts montants ou descendants d'horloge.
- Mise en place d'un « pull-up » ou « pull-down »dans le but de limiter la consommation des entrées/sorties inutilisées.
- Chaque sortie peut délivrer un courant de 12mA. Ainsi toutes ces possibilités permettent au concepteur de connecter au mieux une architecture avec les périphériques extérieurs.

II.3.1.3. Les interconnexions :

Dans les FPGA on peut distinguer deux types d'interconnexions (locales et globales). La composante locale a pour rôle d'interconnecter les éléments des blocs logiques et d'assurer la communication entre les blocs logiques directement adjacents sans avoir recours à la composante globale. La deuxième composante assure la circulation des données à l'échelle du FPGA entre les blocs logiques éloignés.

Au niveau local, on trouve donc des interconnexions configurables au sein des blocs logiques et entre les cellules logiques élémentaires. Réalisé au moyen de multiplexeurs, de buffers trois états et de portes passantes. On trouve également un groupe d'interconnexions vers les blocs logiques directement adjacents selon le même principe. Il s'agit d'interconnexions courtes présentant de faibles retards si on les compare aux interconnexions à longue distance.

Au niveau global, on trouve un réseau d'interconnexions horizontales et verticales à l'échelle de circuit situé entre les blocs logiques. L'échange des données entre les blocs logiques et ces interconnexions globales se fait par l'intermédiaire de Cellules d'Interconnexions Locales (CIL). L'aiguillage des données au sein du quadrillage d'interconnexions globales est assuré par des Cellules d'Interconnexions Globales(CIG) placées à chacune de ces intersections

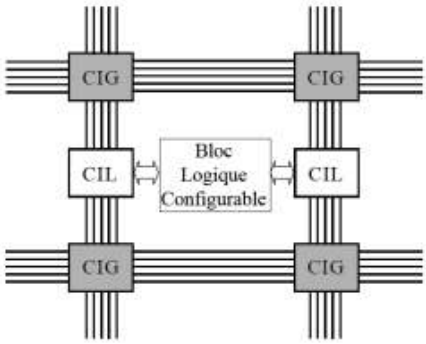


Figure (II.5): Réseau d'interconnexions global

Ces cellules (CIL et CIG) sont constituées principalement de multiplexeurs et de portes de transmission à un ou deux transistors. On trouve également de nombreux buffers et buffers trois états afin de régénérer régulièrement les signaux le long des interconnexions.

Il existe également un ou plusieurs réseaux d'interconnexions configurables dédiées aux signaux d'horloge. Ils comportent principalement des inverseurs et des buffers trois états.

II.3.2. Une couche de configuration :

la couche de configuration appelée mémoire de configuration permet de programmer électriquement les caractéristiques des ressources de la couche active. En effet, toutes les ressources logiques du FPGA sont contrôlées par le contenu de la « mémoire de configuration » (chez Xilinx cette mémoire est à base de cellules **SRAM** (**S**tatic **R**andom **A**ccess **M**emory) volatiles, le FPGA doit donc être reconfiguré à chaque mise sous tension). Leur contenu fixe l'équation des LUTs, le routage des signaux, les entrées/sorties et leurs tensions ainsi que les paramètres de toutes les autres ressources du FPGA.

Les cellules SRAM consiste en deux inverseurs CMOS connectés en boucle pour former un bistable. L'état de cette cellule peut être modifié par un signal électrique externe (la ligne B). La cellule RAM est une structure de stockage volatile. La figure(6) représente une cellule SRAM à 5 transistors(a) et une cellule SRAM à 6 transistors(b). Malgré son coût, c'est la méthode utilisée dans les FPGA les plus performants à ce jour.

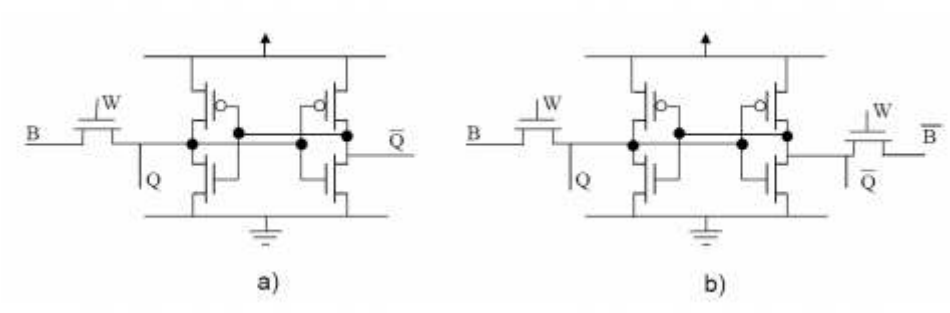


Figure (II.6) : Cellules SRAM à 5 transistors (a) et à 6 transistors (b)

II.4. Méthodologie de conception :

II.4.1. Les outils de CAO (*C*onception *A*ssistée par *O*rdinateur) pour la configuration d'un FPGA :

Le rôle principal confié aux outils de CAO se résume en 4 étapes qui sont : la description, la simulation, la synthèse, le placement et le routage et en dernier la configuration du FPGA. Un design peut être conçu à l'aide d'un éditeur schématique lors de la conception des circuits simples ou d'un outil de programmation utilisé pour les circuits complexes.

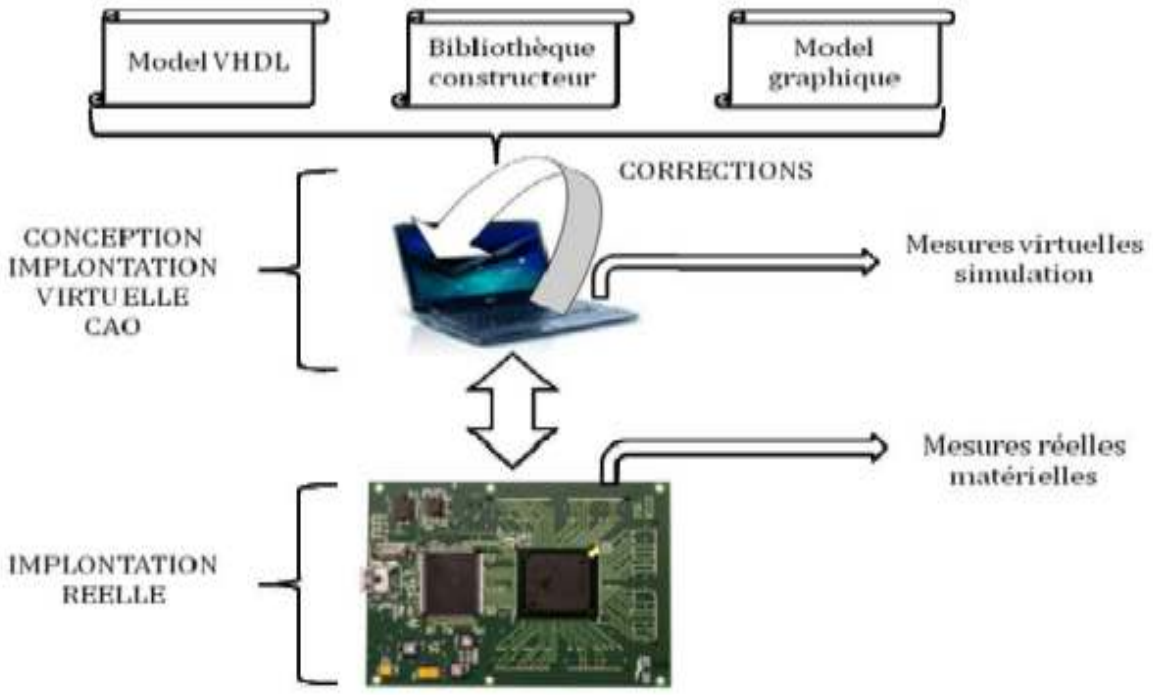


Figure (II.7) : Mode d'exécution matériel des outils de CAO

II.4.1.1. Spécification du design :

Pour réaliser un circuit, il faut d'abord envisager l'architecture globale de ce circuit et spécifier les trois éléments suivants :

-Le nombre de broches d'entrées/sorties et leurs localisations dans le composant FPGA

- La spécification de la fréquence d'horloge du système.

-La spécification de la mémoire requise pour l'application.

II.4.1.2. Développement du design :

-Spécification de la méthodologie de design (Outil de développement utilisé)

- La saisie du circuit - Codage RTL (VHDL, Verilog ...)

- Graphique (Machine à état)
- Saisie HDL (**H**ardware **D**escription **L**anguage)

- La simulation (Prés et Post synthèse)

II.4.1.3. Synthèse :

L'outil de synthèse a pour objectifs de minimiser la surface de silicium, e temps de propagation ainsi que la consommation. Cet outil permet de convertir la représentation du design à partir du code HDL fourni pour produire une représentation au niveau de portes logiques. Cette phase s'occupe de déterminer quelles sont les structures susceptibles pour répondes à un cahier des charges étudié et de produire un code sous forme d'un fichier.

II.4.1.4. Placement et Routage :

Le placement et le routage sont réalisés en définissant les chemins qui relient l'ensemble des blocs logiques choisies pour notre application à travers un algorithme de routage qui est sensé de faire l'aiguillage des données qu'il reçoit vers leurs destinations par action sur les nœuds de routage.

Plusieurs traitements sont necessaires pour obtenir un fichier de configuration :

-Partitionnement : Les équations logiques spécifiques de notre application sont regroupées en un autre ensemble équivalent d'équations. Chaque équation de ce nouvel ensemble peutêtre implémentée dans un seul bloc logique du composant FPGA.

-Placement : Des blocs logiques sont sélectionnés et affectés au calcul des nœuds du réseau booléen.

-Routage : Les ressources d'interconnexions sont affectées à la communication de l'état des nœuds du réseau vers les différents blocs logiques.

-Génération des données numériques de configuration : Les informations abstraites de routage, de placement et les équations implantées dans les blocs sont transformées en un ensemble de valeurs binaires, fournies sous forme d'un fichier appelé « bitstream » qui va être envoyé vers le FPGA via une interface de configuration.

II.4.1.5. Intégration et implémentation :

L'implémentation est la réalisation proprement dite qui consistre à mettre en œuvre l'algorithme sur l'architecture du circuit configurable cible, c'est-à-dire à compiler, à charger, puis lancer l'excécution sur un ordinateur ou calculateur. C'est une étape de programmation physique et de tests électriques qui cloturent la réalisation du circuit. La figure suivante résume un peu l'ensemble de ces étapes.

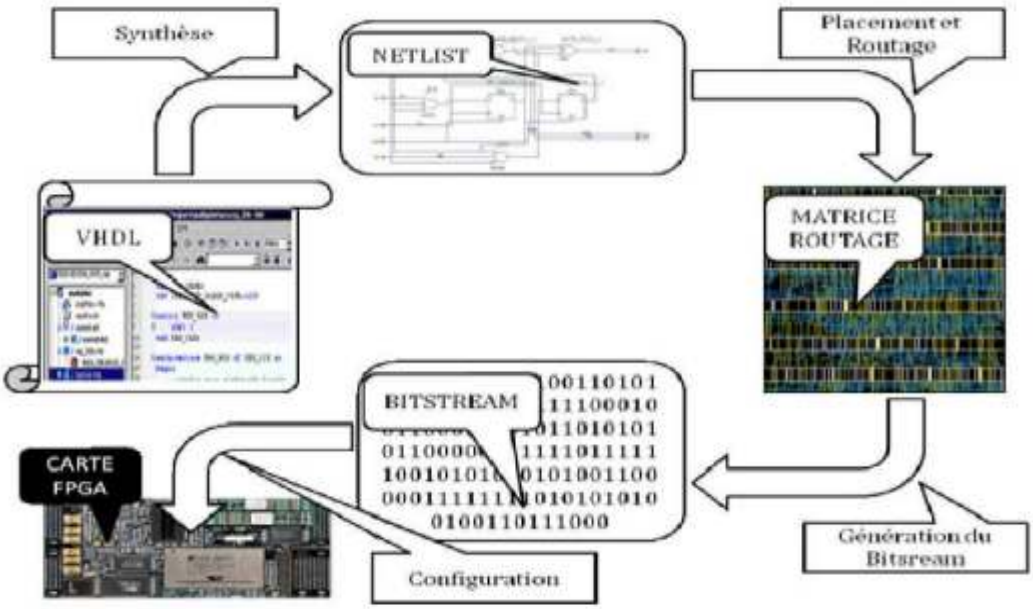


Figure (II.8) : Cycle de programmation d'un FPGA en utilisant les outils de CAO

II.4.2. Langage VHDL :

II.4.2.1. Historique :

Dans les années 80, le département de la défense aux Etats-Unis fait un appel d'offre pour voir un langage de description matérielle numérique unique. Le langage **VHDL** (**V**hsic **H**ardware **D**escription **L**anguage) est inventé pour répondre à ces critères. Ce langage se base sur le **VHSIC** (**V**ery **H**igh **S**peed **I**ntegrated **C**ircuit) qui est un projet de recherche memé par le groupement IBM/Texas. Ce langage est ouvert au domaine public en A985 et devendra une norme en 1987 sous la dénomination de IEEE 1076-1987. Des changements minimes se feront pour la seconde normalisation en 1993 et il portera le nom VHDL'93. La denière évolution de la norme est la norme IEEE 1076-2001.

Cependant, la norme IEEE 1076-2001 ne permeet pas seule la description mixte (numérique et analogique). Le Verilog et le VHDL ont des capacités techniques équivalentes. Et donc le choix du langage revient à l'utilisateur et aux outils qui les métrisent et par fois ce choix s'impose à travers les logiciels de simulation et de synthèse disponibles.

II.4.2.2. Description :

L’ambition des concepteurs du langage est de fournir un outil de description qui permet de créer des modèles de simulation. Initialement réservé au monde des circuits numériques, VHDL est en passe d’être étendu aux circuits analogiques.

Deux intérets majeurs du langage sont :

- Des niveaux de description très divers : VHDL permet de représenter le fonctionnement d’une application tant du point de vue système que du point de vue circuit, en descendant jusqu’aux opérateurs les plus élémentaires. A chaque niveau, la description peut être structurelle (portrait des interconnexions entre des sous-fonctions) ou comportementale (langage évolué).
- Son aspect « nom propriétaire » : le développement des circuits logiques a conduit chaque fabriquant à développer son propre langage de description. VHDL est en passe de devenir un langage commun aux nombreux outils de CAO, indépendants ou liés à des producteurs de circuits, des outils d’aide à la programmation.

II.4.2.3. VHDL par rapport aux autres langages :

Bien que VHDL soit maintenant largement accepté et adopté, il n'est pas le seul langage dedescription matériel. Pendant ces trente dernières années, beaucoup d'autres langages ont étédéveloppés, ont évolués et sont encore utilisés aujourd'hui par les concepteurs decircuitsintégrés. Créé pour être un standard, VHDL doit son succès à la fois à sesprédécesseurs et à la maturité de ses principes de base.

Du point de vue de leur syntaxe, VHDL s'est largement inspiré du langage ADA, Verilogressemble au langage C et Pascal, et M est totalement fondé sur le langage C. VHDL estcertainement le plus difficile à utiliser car il reste très général. De plus, il demande àl'utilisateur d'avoir des habitudes de programmeur (compilation séparée, langage fortement typé, notion de surcharge, etc.). Comparé à Verilog et M qui restent proches de la réalité physique,VHDL est sans aucun doute plus complexe.

II.5. Comparaison entre les FPGA et les autres circuits spécifiques :

La comparaison et donc le choix entre les différentes technologies devient une étape délicate et primodiale, car elle conditionne non seullement la conception mais aussi toute l’évolution du produit à concevoir. De plus, elle détermine le coût de réalisation et de la rentabilité économique du produit. Ceci implique qu’il faut faire une étude comparative des circuits existant sur le marché pour faire ce choix, en se basant sur lusieurs critères dont on peut citer le rapport coût / souplesse d’utilisation et les quantités à produire.

II.5.1. Comparaison entre les PLD et les ASIC :

Un premier choix doit être fait entre les ASIC et les PLD. Les avantages des PLD par rapport aux ASIC sont les suivants :

- ils sont entièrement programmables par l'utilisateur,
- Ils sont généralement reprogrammables dans l'application, ce qui facilite la mise au point etgarantit la possibilité d'évolution,
- les délais de conception sont réduits, il n'y a pas de passage chez le fondeur.

En revanche, les inconvénients des PLD par rapport aux ASIC sont les suivants :

- ils sont moins performants en terme de vitesse de fonctionnement (d'un facteur 3),
- le taux d'intégration est moins élevé (d'un facteur 10 environ),
- Le programmation coûte les 2/3 de la surface de silicium.

De plus, le coût de l'ASIC est beaucoup plus faible que le coût du PLD.

II.5.2. Comparaison entre les FPGA et les EPLD :

Si un PLD est choisi, il faut savoir si on doit utiliser un EPLD ou un FPGA. En réalité, le choix est assez facile à faire. Le domaine d'utilisation des FPGA est celui des prédiffusés, par exemple les fonctions logiques ou arithmétiques complexes ou le traitement du signal. Ledomaine d'utilisation des EPLD est plutôt celui des PAL, par exemple les machines d'étatcomplexes. Il est à noter qu'un marché important des PAL et des EPLD est la correction deserreurs de conception dans les ASIC afin d'éviter un aller-retour coûteux chez le fondeur.

II.5.3. Seuil de rentabilité entre un FPGA et un ASIC :

Après avoir opté dans notre choix pour les FPGA par rapport aux autres circuits intégrés, et après la mise en œuvre de notre système, la production devienne la prochaine étape dans le cycle de développement. Avec le taux d'intégration qui devient de plus en plus important, les FPGA sont devenues un moyen très intéressant en fabriquant des petites et moyennes quantités, mais en grandes productions les ASIC deviennent plus important en terme de coût comme illustré dans la figure suivante. La question qui se pose est : combien d'unités doit en produire pour que l'ASIC soit plus rentable que le FPGA ? Ou quel est le seuil de production entre les FPGA et les ASIC ?

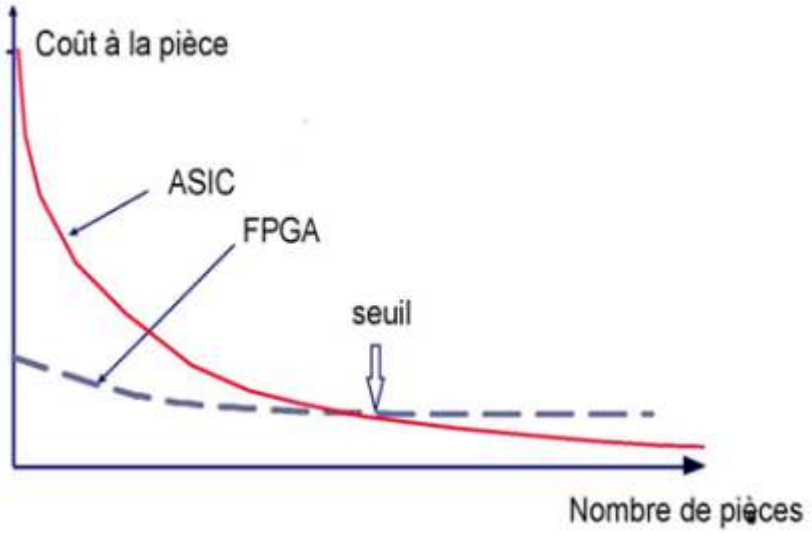


Figure (II.9) : Seuil de rentabilité entre les FPGA et les ASIC

La détermination du coût d'un circuit intégré se résume au nombre de puces que l'on peut fabriquer sur une tranche de silicium càd sur la surface de la puce. Pour cela, il faut étudier les éléments qui agissent sur la taille de la puce, qui se réduise aux nombres d'entrées/sorties et le nombre de portes suffisantes pour réaliser une fonction logique.

Une analyse rapide peut donner un ordre de grandeur du seuil de rentabilité entre un FPGA et un ASIC. La formule de base du seuil de rentabilité est la suivante :

Seuil de rentabilité **=** NRE **+** (développement et outils) **+** (X unités * prix à l'unité)

NRE(**N**on **R**ecurring **E**xpenses) : les fixes de mise en œuvre

On obtient pour les ASIC et les FPGA les deux formules suivantes :

ASIC **=$**25 000 NRE **+ $**79000(développement et outils) **+** (X unités * **$** 13)

FPGA =0 NRE **+ $**25 000(développement et outils) **+** (X unités * **$** 79)

Les chiffres permettant de quantifier les seuils de rentabilité entre ASIC et les FPGA. L'ordre de grandeur du seuil de rentabilité est le suivant :

Jusqu'à 5000 piéces	**Plus de 5000 piéces**
FPGA	**ASIC**

Voici un tableau récapitulatif de comparaison des différents circuits :

	ASIC	FPGA	DSP
Performance	Très élevée	Elevée	Faible
Taille	Faible	Moyenne	Elevée
Consommation	Faible	Moyenne	Très élevée
Intégration	Système sur puce	Système sur puce	Composants supplémentaires
Souplesse	Fonctions figées	Reconfigurable	Programmable
Mise en œuvre	Complexe	Complexité moyenne	Complexité moyenne
Cout du composant	Très élevé	Moyen	Faible

Tableau(II.1) : comparaison entre les FPGA et les autres circuits spécifiques

II.6. Avantages et inconvénients des FPGA :

Avantages	Inconvénients
-Technologie facile à maitriser - Temps de développement réduit - Reprogrammable - Idéal pour le prototypage - Coût peu élevé - Parallélisme de traitement - Flexibilité et la possibilité de réduire fortement les délais de développement et de commercialisation - La configuration, parfois en temps réel	- Performances non optimisées - Temps de réponse long par rapport aux ASIC

Tableau(II.2) : Avantages et inconvénients des FPGA

II.7.Différent domaines d'application des FPGA :

Les FPGA on fait révolutionner certain domaines de contrôle numérique et de plus en plus utilisés pour intégrer des architectures numériques complexes. Ils sont devenus les plus populaires en matière d'implantation et de prototype des circuits numériques après leur apparition sur le marché en 1984. La clé maitresse de leurs réussites est l'aspect de programmation de ces derniers. Leurs utilisations actuelles couvrent les deux domaines : civil et militaire. Parmi ces applications nous citons :

- **Informatique :** Périphériques spécialisés
- **Machinerie industrielle :** Controleur pour machines
- **Télécommunication :** Traitement d'images, Filtrage
- **Instrumentation :** Equipement médical, Prototypage
- **Transport :** Contrôle d'avions et métros
- **Aérospatiale :** Satellites
- **Militaire :** Radar, Communication protégée, la détection ou la surveillance..

II.8. Conclusion :

Dans ce chapitre, nous avons présenté des généralités sur les circuits FPGA, et leur architecture interne. Ensuite nous avons vu la méthodologie de conception de ces circuits, en expliquant les outils de CAO de leur configuration, ainsi que le langage de description VHDL.

A la fin, nous avons terminé par une comparaison entre les circuits FPGA et d'autres circuits spécifiques, et nous avons cité les avantages et les inconvénients de ces circuits.

Dans le chapitre suivant, nous allons présenter la description de notre modèle (Izhikevich) en VHDL, et son implantation sur FPGA, ainsi que les résultats de notre travail.

Partie Pratique

Chapitre (III)

Conception et Description VHDL

III.1. Introduction

Dans la partie pratique, nous allons présenter toutes les démarches nécessaires pour arriver à notre objectif - description en VHDL d'un réseau de neurones impulsionnels-

Donc, dans ce chapitre, nous allons présenter la méthodologie de conception de notre circuit et la description VHDL du modèle Izhikevich en utilisant deux versions différentes : la première utilisant la représentation des nombres en virgule fixe, la seconde utilisant des nombres réels.

III.2. Implantation sur FPGA

La dynamique des réseaux de neurones impulsionnels est décrite par des systèmes d'équations différentielles qu'il n'est en général pas possibles de résoudre analytiquement. On trouve deux approches pour simuler ces réseaux :

- L'approche événementielle : la simulation du réseau est dirigée par l'occurrence des émissions d'impulsions.
- L'approche intégration numérique : les équations différentielles sont approximées par des méthodes d'intégration numérique, l'état du réseau est évalué à des instants a priori sans rapport avec les temps d'émissions des impulsions

La principale réalisation envisagée dans cette étude est un simulateur qui se trouvera entre ces deux extrêmes. Il sera suffisamment réaliste au niveau biologique pour simuler des neurones impulsionnels et suffisamment simple pour permettre une simulation temps réel d'un réseau de grande taille (l'implantation sur un circuit).

La description de notre réseau de neurones en VHDL est faite par le logiciel ***ISE PROJECT NAVIGATOR (XILINX 13.3),*** et la simulation est faite par ***ISIM***

III.3. Méthodologie de conception

Le rôle principal confié aux outils de CAO se résume en quelques étapes qui sont : la spécification de design, la déclaration des ports d'E/S, la description VHDL, la synthèse, et la création du fichier Test-Bench et en dernier la simulation et la configuration du FPGA (après la conception). Un design peut être conçu à l'aide d'un éditeur schématique lors de la conception des circuits simples ou d'un outil de programmation utilisé pour les circuits complexes.

Les étapes principales de la méthodologie de conception de notre modèle sont présentées par la figure suivante :

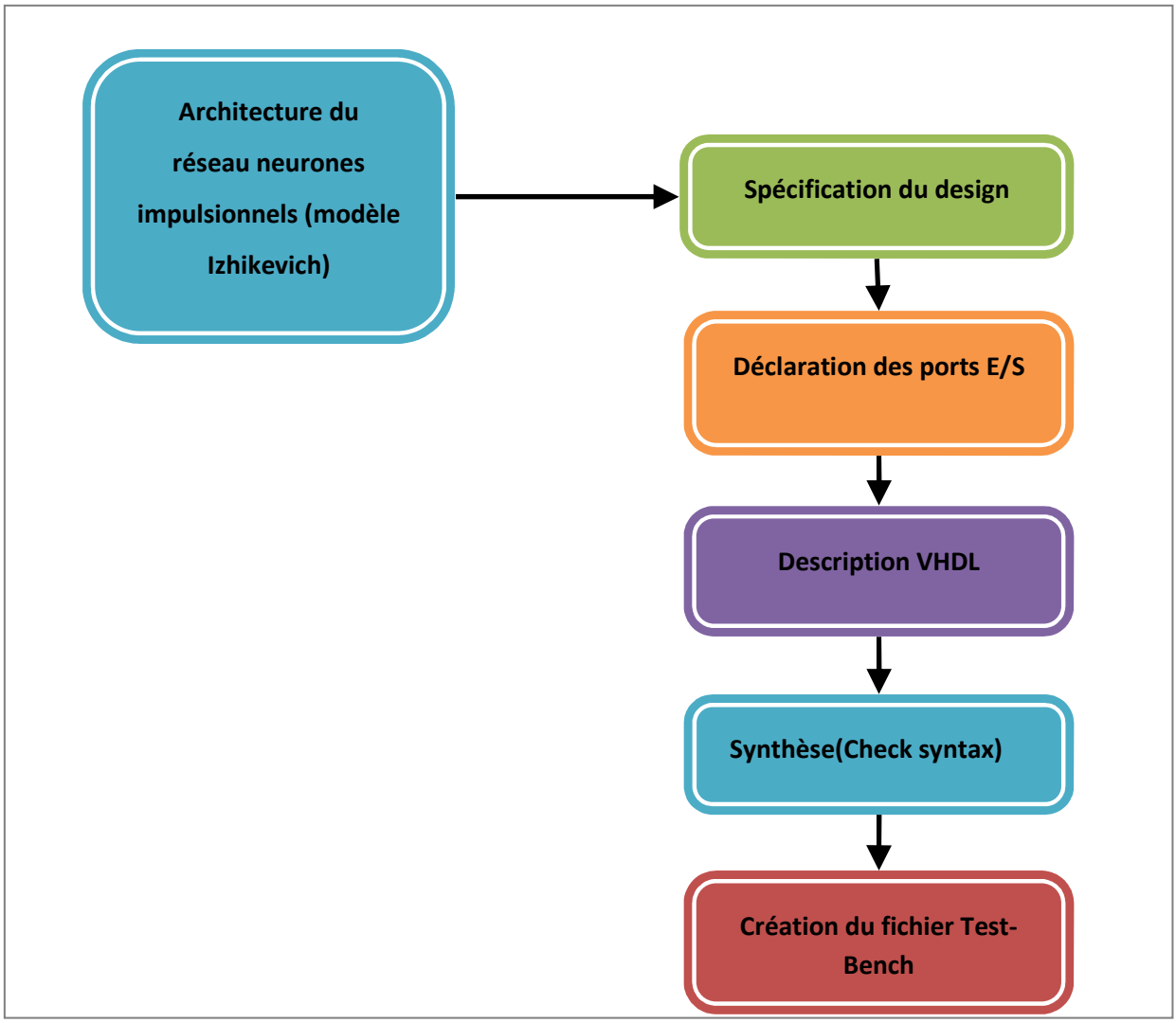


Figure (III.1) : les étapes de la méthodologie de la conception

III.3.1. L'architecture du modèle Izhikevich

Le modèle Izhikevich est basé sur les paramètres (a, b, c et d). En faisant varier ces paramètres, on peut obtenir différentes réponses neuronales. Il est décrit par les équations :

$$v' = 0.04v^2 + 5v + 140 - u + I$$

$$u' = a(bv - u)$$

Avec :

$$if\ v \geq +30mV \qquad then \begin{cases} v \leftarrow c \\ u \leftarrow u + d \end{cases}$$

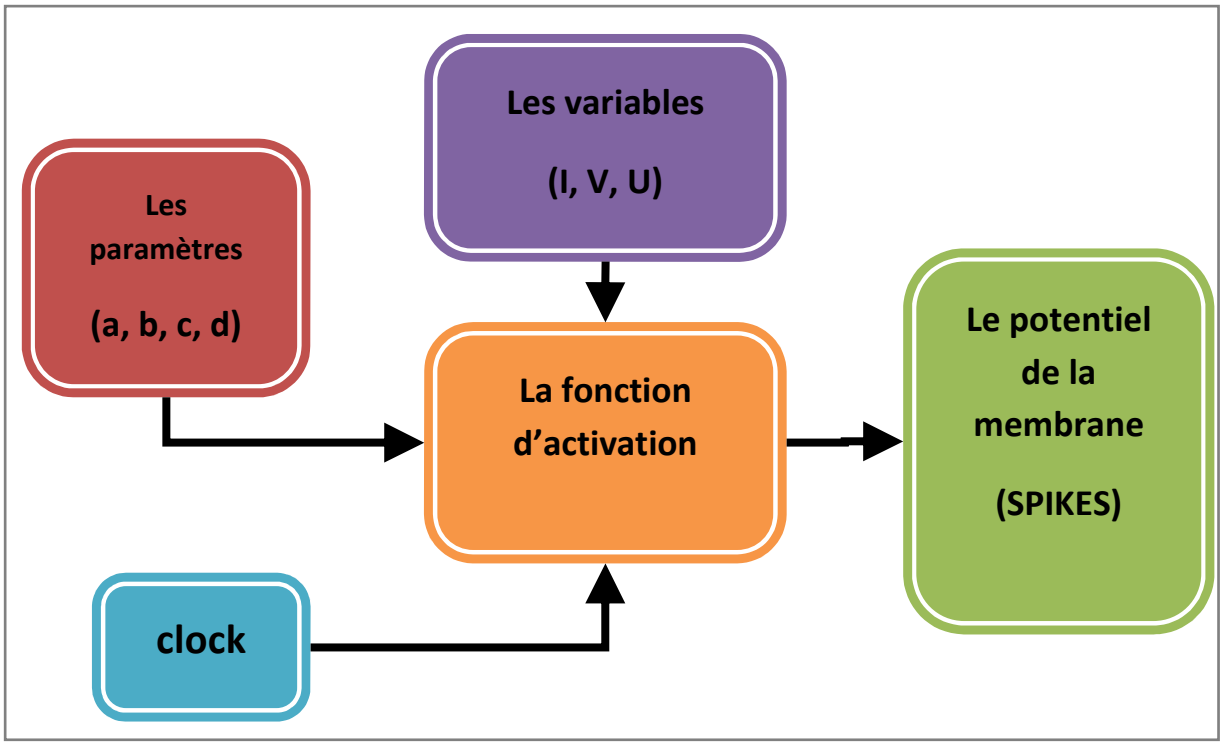


Figure (III.2) : l'architecture de modèle izhikeviich

III.3.2 Spécification du design :

III.3.2.1 Spécification du périphérique

La première étape de la spécification de design est de sélectionner : le périphérique, langage, et le simulateur

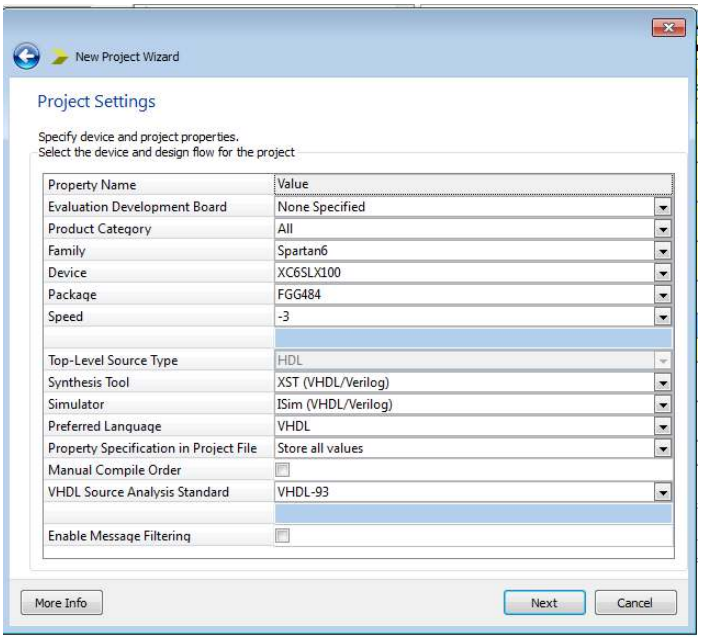


Figure (III.3) : spécification du périphérique

III.3.2.2 Sélection de type de source VHDL

Après la spécification de notre périphérique, on doit sélectionner le type-source de module

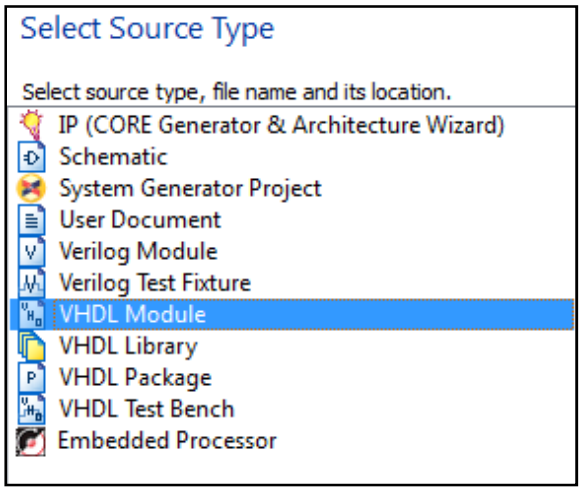


Figure (III.4) : la sélection de module VHDL

III.3.2.3 Définition du module :

Dans cette étape nous avons déclaré tous les ports d'entrées / sorties, de notre module

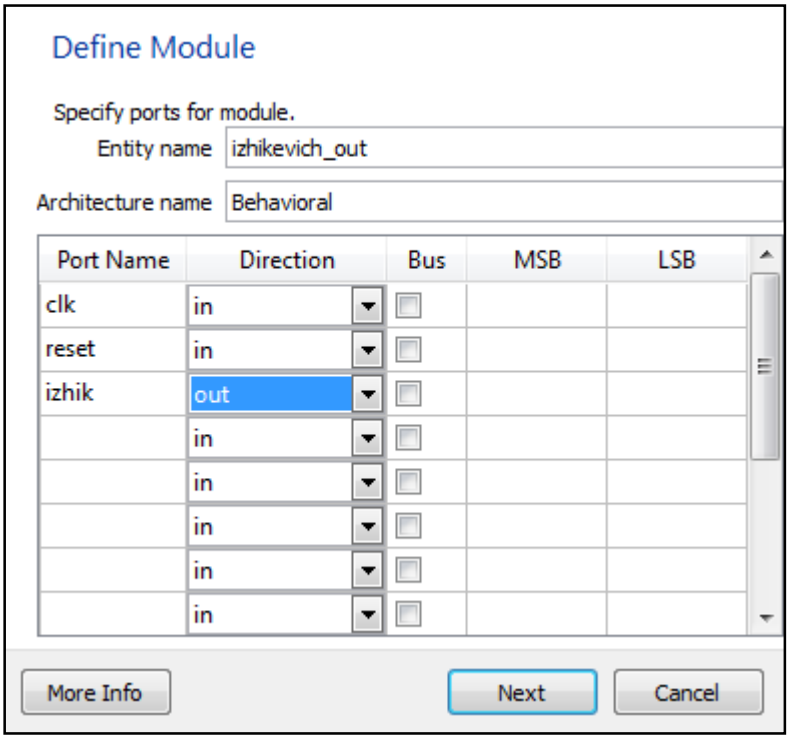


Figure (III.5) : la déclaration des ports d'E/S

III.3.3. La description en VHDL

Afin de réaliser l'opération complète modélisant la sortie d'un neurone, nous commençons par la réalisation des opérateurs arithmétiques nécessaires pour le fonctionnement. Le but d'une description VHDL étant la réalisation matérielle d'un circuit le choix de la représentation des nombres influe sur la précision des calculs, et la densité du circuit (très souvent un compromis entre précision des calculs et minimisation des circuits est à faire).

La première étape est la description VHDL des opérateurs d'addition et de multiplication. Pour la description des opérateurs arithmétiques nécessaires à la modélisation des neurones artificiels, il est important de choisir la technique de représentation des nombres. Les représentations des nombres en format simple, en virgule fixe ou en virgule flottante est toujours un choix qui dépend de l'application à implanter.

Dans notre projet nous nous intéressons à la représentation en deux types : type réel et type virgule fixe.

III.3.3.1. Représentation en utilisant « sfixed »

Les données en ***virgule fixe*** sont composées d'une partie fractionnaire et d'une partie entière pour lesquelles le nombre de bits alloués reste figé au cours du traitement. La figure (III.6) représente une donnée en virgule fixe composée d'un bit de signe et « b-1 » bits repartis entre la partie entière et la partie fractionnaire.

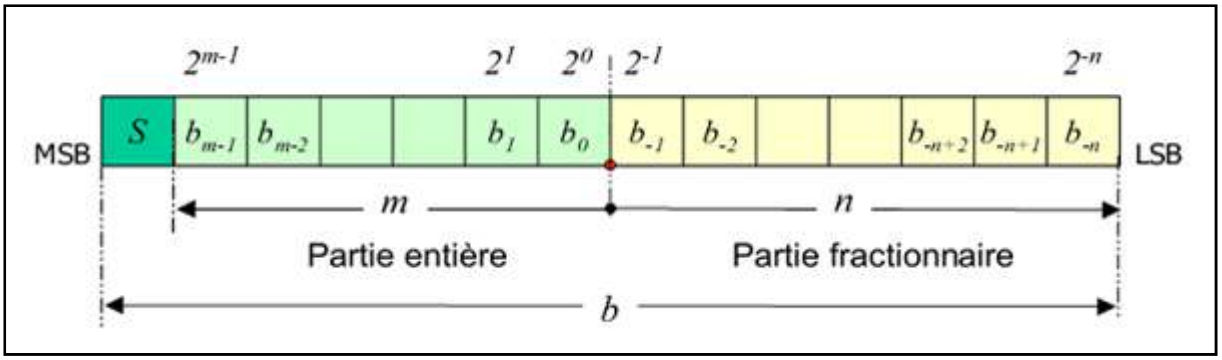


Figure (III.6) : Représentation des données en virgule fixe

III.3.3.1.1 La bibliothèque

Pour les opérateurs en type **« virgule fixe »,** il faut ajouter la bibliothèque **« library :IEEE_proposed »** et le paquetage**« IEEE_proposed.fixed_pkg » (fig.III.7)**

```
20  library IEEE;
21  use IEEE.STD_LOGIC_1164.ALL;
22  use IEEE.std_logic_signed.all;
23  use IEEE.NUMERIC_STD
24
25  library IEEE_proposed;
26  use IEEE_proposed.fixed_pkg.all;
```

Figure (III.7) : la déclaration de la bibliothèque « sfixed »

III.3.3.1.2 Les opérations arithmétiques

Pour effectuer les opérations arithmétiques (addition, multiplication...etc.) il faut que les opérandes possèdent un format commun. Pour cela nous avons utilisé la fonction **« resize ».**

Un aperçu sur la description VHDL des opérations, en utilisant la représentation en virgule fixe **(sfixed)** est donnée par la figure (III.8)

```
46  architecture Behavioral of izhikevich_neurone is
47
48  signal a:sfixed (7 downto -16);
49  signal b:sfixed (7 downto -16);
50  signal c:sfixed (7 downto -16);
51  signal d:sfixed (7 downto -16);
52
53  signal I1:sfixed (7 downto -16);
54
55  Signal count: std_logic_vector(7 downto 0):="00000000";
56
```

Figure (III.8) : la description VHDL en virgule fixe

Un aperçu sur la description VHDL en utilisant la fonction **«resize»** est donné par la figure(III.9)

```
123
124         v_temp <= resize((k*v*v)+(5*v)+w+I1-u,v);
125         v_new <= resize (v + v_temp ,v_new);
126
127         u_temp <= resize(a*(b*v-u),u);
128         u_new <= resize(u + u_temp ,u_new);
129
```

Figure (III.9) : la description VHDL avec la fonction « resize »

III.3.3.2. Représentation en utilisant « Real »

III.3.3.2.1.La bibliothèque

Pour les opérateurs en type **« real »,** nous avons utilisé le paquetage **« IEEE.math_real»**

```
1  ----------------------------------------------------
2  -----------------VHDL-Izhikevich-------------------
3  ----------------------------------------------------
4  library IEEE;
5  use IEEE.STD_LOGIC_1164.ALL;
6  use IEEE.std_logic_signed.all;
7  use IEEE.std_logic_arith.all;
8  use IEEE.math_real.all;
9
```

Figure (III.10) : la déclaration de la bibliothèque « math-real »

III.3.3.2.2.Les opérations arithmétiques

Un aperçu sur la description VHDL en utilisant la représentation en réel (real) est donné par la figure (III.11) et la figure (III.12)

```
181  architecture Behavioral of izhikevich_reel is
182
183  signal a: real:=0.02;
184  signal b: real:=0.2;
185  signal c: real:=-65.0;
186  signal d: real:=6.0;
187
188  signal I1:real:=14.0;
189
190  Signal count: std_logic_vector(7 downto 0):="00000000";
191
```

Figure (III.11) : la description VHDL en type réel

```
219
220        v_new <= v + 0.25*((k*v*v)+(5.0*v)+w+I1-u);
221        v_new <= v + v_temp;
222
223        u_new <= u + 0.25*(a*(b*v-u));
224        u_new <= u + u_temp;
225
```

Figure (III.12) : la description VHDL des opérations arithmétiques

III.3.4. La création de fichier Test-Bench

La création de fichier Test-Bench est la première étape de la simulation

III.3.4.1. La spécification du type de source

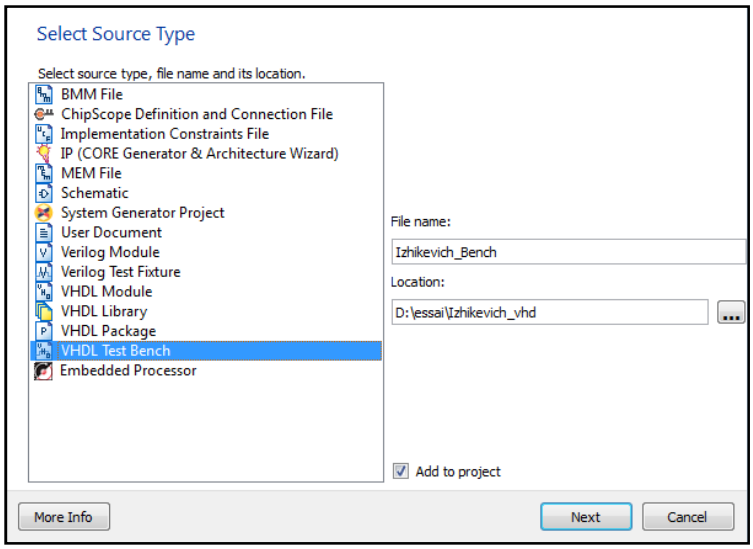


Figure (III.13) : sélection de fichier Test-Bench

III.3.4.2. La description VHDL de fichier Test-Bench

Un aperçu sur la description du fichier Test-Bench est donné par la figure suivante :

```
37
38  ARCHITECTURE behavior OF izhikevich_reel_bench IS
39
40      -- Component Declaration for the Unit Under Test (UUT)
41
42      COMPONENT izhikevich_reel
43      PORT(
44           clock : IN  std_logic;
45           reset : IN  std_logic;
46           izhikevich_out : OUT  real
47          );
48      END COMPONENT;
49
50
51     --Inputs
52     signal clock : std_logic := '0';
53     signal reset : std_logic := '0';
54
55     --Outputs
56     signal izhikevich_out : real;
```

Figure (III.14) : Aperçu sur la description du Test-Bench

Dans cette étape il faut bien choisir la période, et bien définir le processus de stimulation

III.4.Conclusion

Dans ce chapitre nous avons présenté en détails toutes les étapes pour décrire notre modèle izhikevich en VHDL, et dans le chapitre suivant nous allons présenter la simulation de cette description, et montrer tous les résultats de notre travail pour les deux versions de notre description VHDL.

Chapitre (IV)

Mise en œuvre Résultats et Discussion

IV.1. Introduction

Dans le chapitre précédent, nous avons présenté la méthodologie de conception de notre circuit et la description VHDL du modèle Izhikevich en utilisant deux versions (réel et virgule fixe). Dans ce chapitre, nous allons présenter les résultats des simulations que nous avons obtenus à partir de la description VHDL pour différentes valeurs des paramètres (a, b, c, d) du modèle Izhikevich ; nous comparerons ente eux les résultats obtenus sous xilinx pour les deux versions, ensuite nous les comparerons avec les résultats de simulation sous Matlab.

IV.2. Les résultats :

Nous allons présenter les résultats de la simulation de la description VHDL des deux versions implémentées (nombres réels, et nombres en virgule fixe), les valeurs numériques obtenus sont stockées dans des fichiers texte ; nous utiliserons Matlab pour visualiser les courbes de réponse du modèle simulé sous XILINX

IV.2.1. La description et la simulation en VHDL:

Nous avons présenté les résultats de la simulation de la description VHDL pour différentes valeurs des paramètres (a, b, c, d)

IV.2.1.A. la description de la version de type réel :

Nous avons présenté les résultats de la simulation de la version en type réel

IV.2.1.A.1 Tonic Spiking (A):

La figure suivante représente un aperçu sur la description VHDL de Tonic Spiking

```
254  entity izhikevich_A is
255      Port ( clock : in  STD_LOGIC;
256             reset : in  STD_LOGIC;
257             izhikevich_out : inout  real
258
259             );
260  end izhikevich_A;
261
262  architecture Behavioral of izhikevich_A is
263
264  signal a: real:=0.02;
265  signal b: real:=0.2;
266  signal c: real:=-65.0;
267  signal d: real:=6.0;
268
269  signal v: real:=-70.0;
270  signal u: real:=-14.0;
271  signal vv: real:=0.0;
272
273  signal I1:real:=14.0;
274
```

Figure (IV.1): Un aperçu sur la description VHDL de *Tonic Spiking (A)*

La simulation en Xilinx (ISIM) de la description VHDL de Tonic Spiking est donnée par la figure suivante :

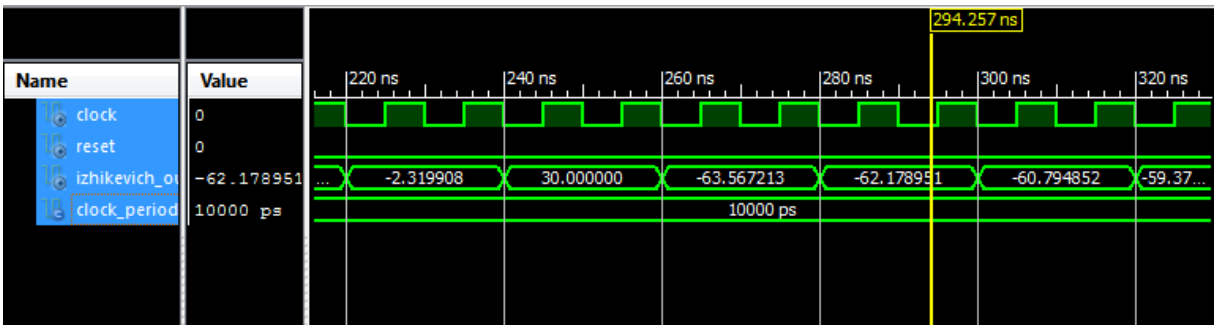


Figure (IV.2) : La simulation en ISIM de Tonic Spiking (A)

Les valeurs qui nous avons obtenu à partir de la simulation en Xilinx, nous allons les stocker dans un fichier texte (.txt) à travers un sous programme (process)

La figure ci-dessous présente le processus d'écriture et le fichier texte

```
75  --write process
76  writing :
77  process
78      file       outfile   : text is out "2.txt";
79      variable   outline   : line;
80
81     begin
82        wait until clock = '0' and clock'event;
83        if(endoffile='0') then
84
85           write(outline, dataread, right, 16, 12);
86
87           writeline(outfile, outline);
88           linenumber <= linenumber + 1;
89        else
90           null;
91        end if;
92
93  end process writing;
```

2 - Bloc-notes

Fichier Edition Format Affichage ?

```
-66.500000000000
-63.402500000000
-60.457729937500
-57.481041267418
-54.296534767340
-50.694064591467
-46.374647600220
-40.853528880771
-33.252806024534
-21.790970979532
 -2.319907606970
 30.000000000000
```

Figure (IV.3): le processus de writting, et le fichier (2.txt) de Tonic-Spiking

IV.2.1.A.2 Phasic Spiking (B):

La figure suivante représente un aperçu sur la description VHDL de Phasic Spiking

```
13  entity izhik_B is
14      Port ( clock : in  STD_LOGIC;
15             reset : in  STD_LOGIC;
16             izhikevich_out : inout  real
17
18             );
19  end izhik_B;
20
21  architecture Behavioral of izhik_B is
22
23  signal a: real:=0.02;
24  signal b: real:=0.25;
25  signal c: real:=-65.0;
26  signal d: real:=6.0;
27
28  signal v: real:=-64.0;
29  signal u: real:=-16.0;
30  signal vv: real:=0.0;
31
32  signal I1:real:=0.5;
33
```

Figure (IV.4): Un aperçu sur la description VHDL de *Phasic Spiking(B)*

La simulation en Xilinx (ISIM) de la description VHDL de Phasic Spiking est donnée par la figure suivante :

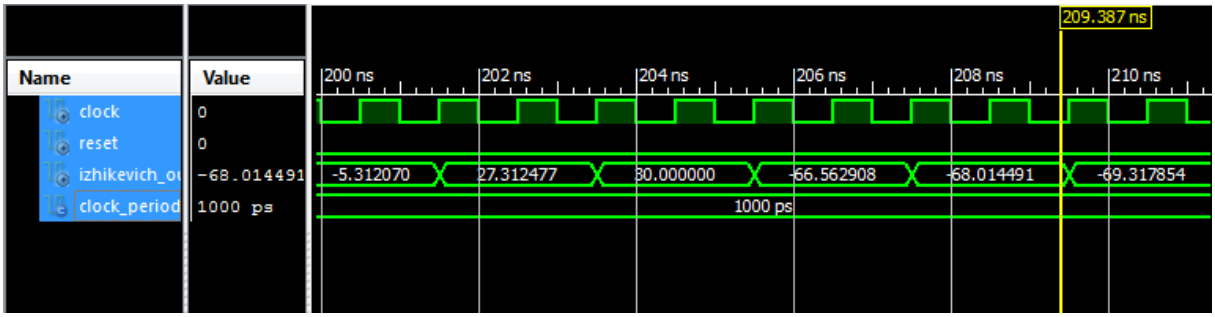


Figure (IV.5) : La simulation en ISIM de Phasic Spiking(B)

La figure ci-dessous présente le processus d'écriture et le fichier texte

```
75  --write process
76  writing :
77  process
78      file      outfile  : text is out "B.txt";
79      variable  outline  : line;
80     begin
81        wait until clock = '0' and clock'event;
82        if(endoffile='0') then
83
84           write(outline, dataread, right, 16, 12);
85
86           writeline(outfile, outline);
87           linenumber <= linenumber + 1;
88        else
89           null;
90        end if;
91
92  end process writing;
```

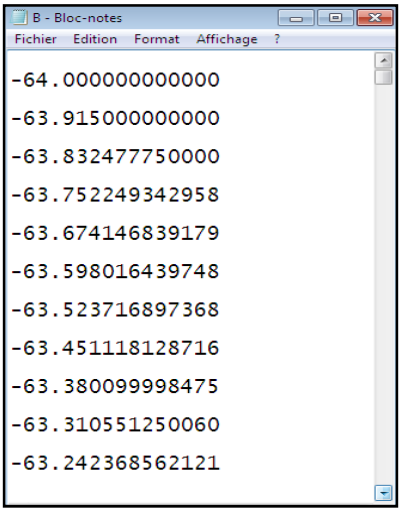


Figure (IV.6): le processus de writting, et le fichier (B.txt) de Phasic-Spiking

IV.2.1.A.3 Tonic bursting (C):

La figure suivante représente un aperçu sur la description VHDL de Tonic bursting

```
13  entity izhik_C is
14      Port ( clock : in  STD_LOGIC;
15             reset : in  STD_LOGIC;
16             izhikevich_out : inout  real
17
18             );
19  end izhik_C;
20
21  architecture Behavioral of izhik_C is
22
23  signal a: real:=0.02;
24  signal b: real:=0.2;
25  signal c: real:=-50.0;
26  signal d: real:=2.0;
27
28  signal v: real:=-70.0;
29  signal u: real:=-14.0;
30  signal vv: real:=0.0;
31
32  signal I1:real:=15.0;
33
```

Figure (IV.7): Un aperçu sur la description VHDL de Ton*ic Bursting(C)*

Le résultat de la simulation en Xilinx de Tonic Bursting (C) est donné par la figure suivante :

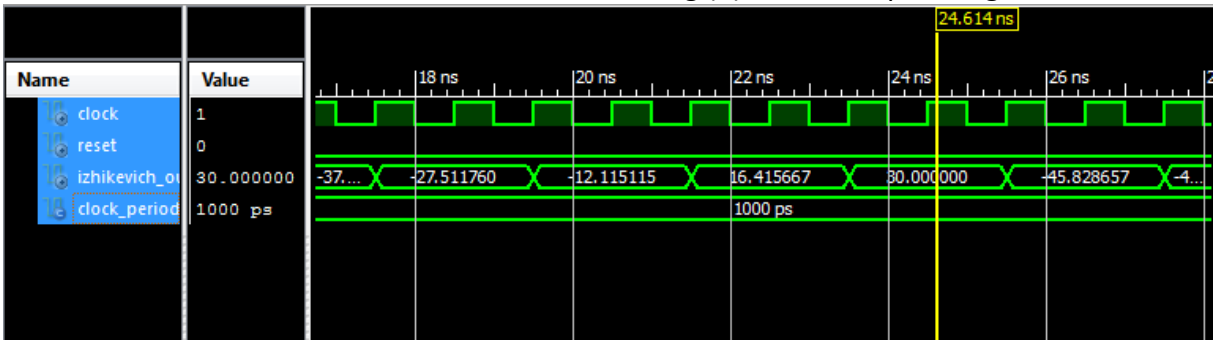


Figure (IV.8) : La simulation en ISIM de Tonic Bursting(C)

La figure ci-dessous présente le processus d'écriture et le fichier texte

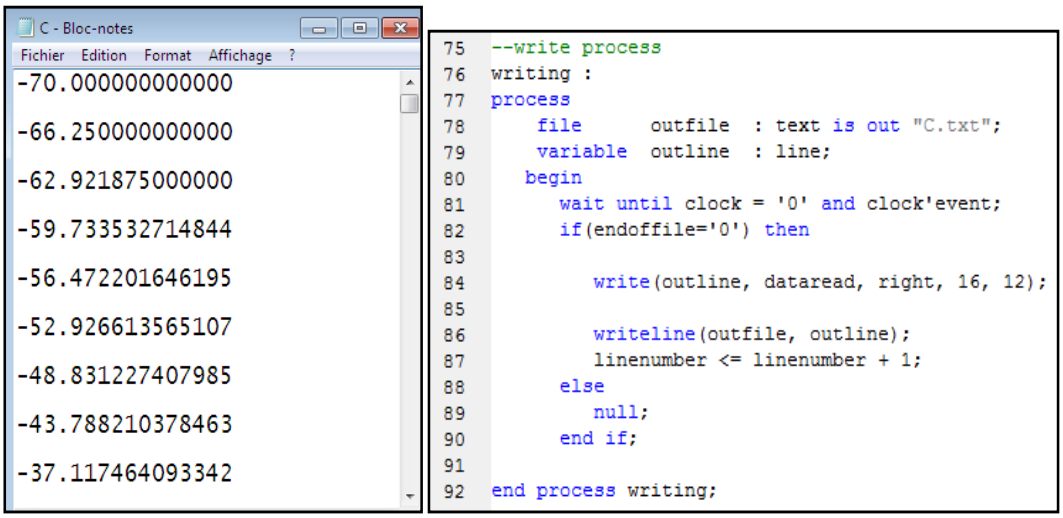


Figure (IV.9): le processus de writting, et le fichier (.txt) de Tonic Bursting

IV.2.1.A.4. Phasic bursting (D)

La figure suivante représente un aperçu sur la description VHDL de Phasic bursting

```
13  entity izhik_D is
14      Port ( clock : in  STD_LOGIC;
15             reset : in  STD_LOGIC;
16             izhikevich_out : inout  real
17
18             );
19  end izhik_D;
20
21  architecture Behavioral of izhik_D is
22
23  signal a: real:=0.02;
24  signal b: real:=0.25;
25  signal c: real:=-55.0;
26  signal d: real:=0.05;
27
28  signal v: real:=-64.0;
29  signal u: real:=-14.0;
30  signal vv: real:=0.0;
31
32  signal I1:real:=0.6;
33
```

Figure (IV.10): Un aperçu sur la description VHDL de Phasic *Bursting(D)*

Le résultat de la simulation en Xilinx de Phasic Bursting(D) est donné par la figure suivante :

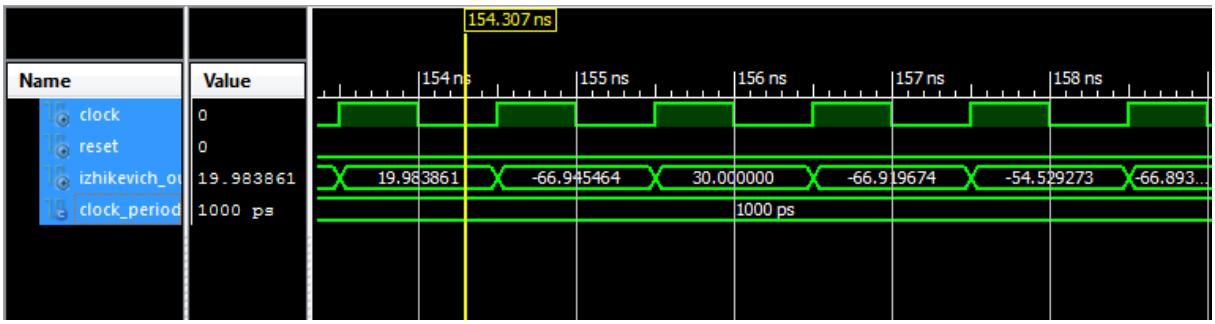


Figure (IV.11) : La simulation en ISIM de PhasicBursting(D)

La figure ci-dessous présente le processus d'écriture et le fichier texte

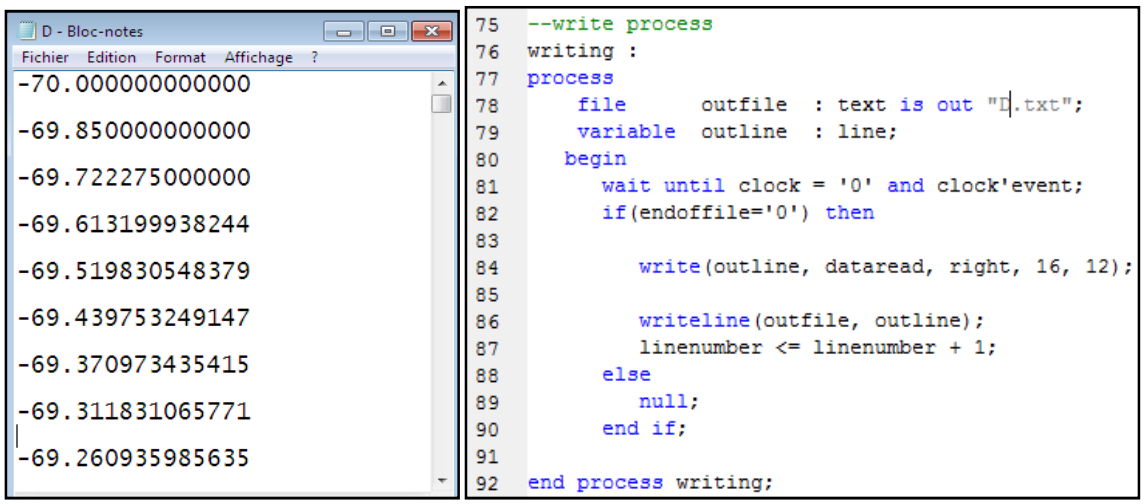


Figure (IV.12): le processus de writting, et le fichier (D.txt) de Phasic Bursting

IV.2.1.A.5. Mixed mode (E)

La figure suivante représente un aperçu sur la description VHDL de Mixed Mode (E) :

```
10
11  entity izhik_E is
12      Port ( clock : in  STD_LOGIC;
13             reset : in  STD_LOGIC;
14             izhikevich_out : inout  real
15
16             );
17  end izhik_E;
18
19  architecture Behavioral of izhik_E is
20
21  signal a: real:=0.02;
22  signal b: real:=0.2;
23  signal c: real:=-55.0;
24  signal d: real:=4.0;
25
26  signal v: real:=-70.0;
27  signal u: real:=-14.0;
28  signal vv: real:=0.0;
29
30  signal I1:real:=10.0;
31
```

Figure (IV.13): Un aperçu sur la description VHDL de Mixed Mode*(E)*

Le résultat de la simulation en Xilinx de Mixed Mode (D) est donné par la figure suivante :

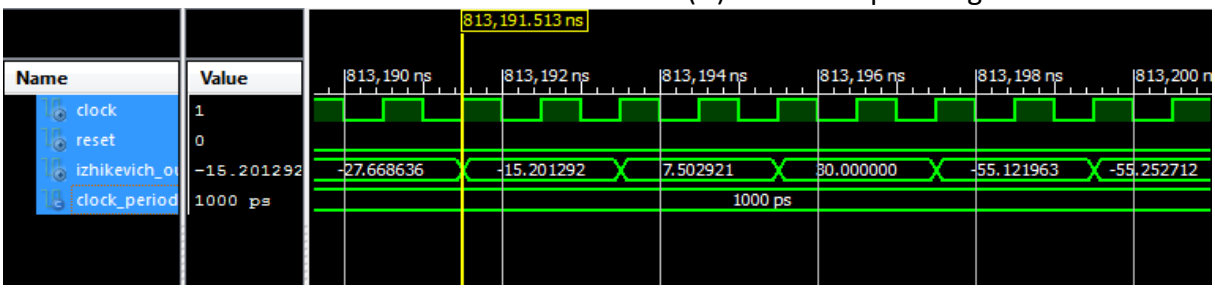


Figure (IV.14) : La simulation en ISIM de Mixed Mode (E)

La figure ci-dessous présente le processus d'écriture et le fichier texte

```
75  --write process
76  writing :
77  process
78      file      outfile  : text is out "E.txt";
79      variable  outline  : line;
80     begin
81        wait until clock = '0' and clock'event;
82        if(endoffile='0') then
83
84           write(outline, dataread, right, 16, 12);
85
86           writeline(outfile, outline);
87           linenumber <= linenumber + 1;
88        else
89           null;
90        end if;
91
92  end process writing;
```

E - Bloc-notes

Fichier Edition Format Affichage ?

```
-70.000000000000
-67.500000000000
-65.312500000000
-63.296523437500
-61.354472691635
-59.407311013884
-57.379768491201
-55.188325661731
```

Figure (IV.15): le processus de writting, et le fichier (F.txt) de Mixed Mode (E)

IV.2.1.A.6. Spike frequency adaptation (F)

La figure suivante représente un aperçu sur la description VHDL de Spike.frq.adapt (F) :

```
10
11  entity izhik_F is
12      Port ( clock : in  STD_LOGIC;
13             reset : in  STD_LOGIC;
14             izhikevich_out : inout  real
15
16             );
17  end izhik_F;
18
19  architecture Behavioral of izhik_F is
20
21  signal a: real:=0.01;
22  signal b: real:=0.2;
23  signal c: real:=-65.0;
24  signal d: real:=8.0;
25
26  signal v: real:=-70.0;
27  signal u: real:=-14.0;
28  signal vv: real:=0.0;
29
30  signal I1:real:=30.0;
31
```

Figure (IV.16): Un aperçu sur la description VHDL de Spike frequency adaptation (F)

Le résultat de la simulation en Xilinx de Spike frequency adaptation (F) est figuré ci-dessous :

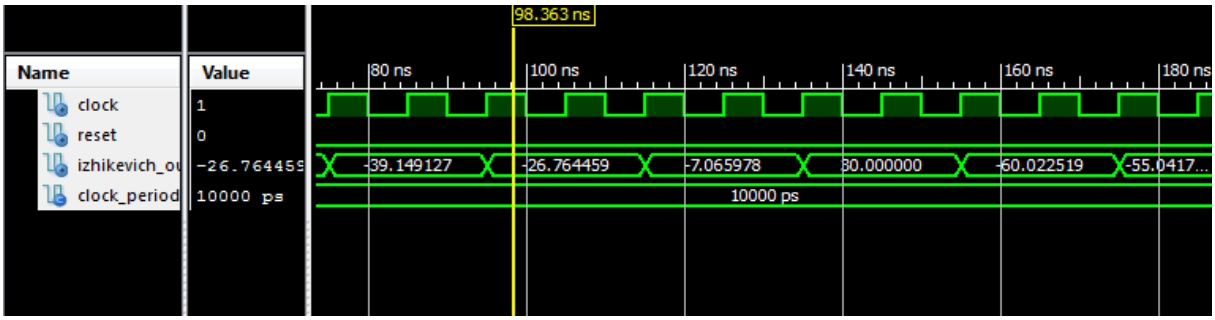


Figure (IV.17) : La simulation en ISIM de Spike frequency adaptation (F)

La figure ci-dessous présente le processus d'écriture et le fichier texte

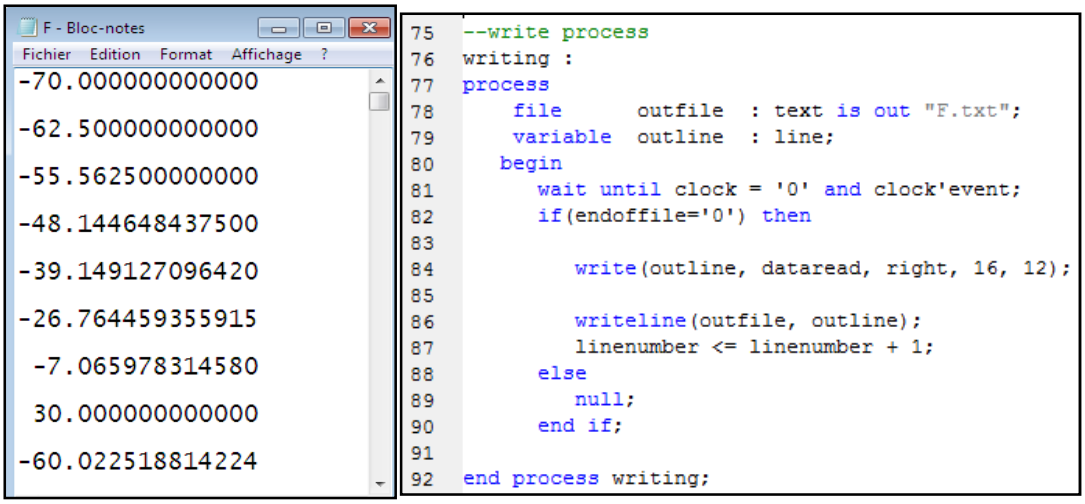


Figure (IV.18): le processus de writting, et le fichier (F.txt) de Spike frequency adaptation (F)

IV.2.1.A.7. Inhibition-Induced-Spiking (S)

```
entity izhik_S is
    Port ( clock : in  STD_LOGIC;
           reset : in  STD_LOGIC;
           izhikevich_out : inout  real

           );
end izhik_S;

architecture Behavioral of izhik_S is

signal a: real:=0.02;
signal b: real:=0.2;
signal c: real:=-55.0;
signal d: real:=4.0;

signal v: real:=-70.0;
signal u: real:=-14.0;
signal vv: real:=0.0;

signal I1:real;
```

Figure (IV.19): Un aperçu sur la description VHDL d'Inhibition-Induced Spiking(S)

Le résultat de la simulation en Xilinx deInhibition-Induced-Spiking (S)est figuré ci-dessous :

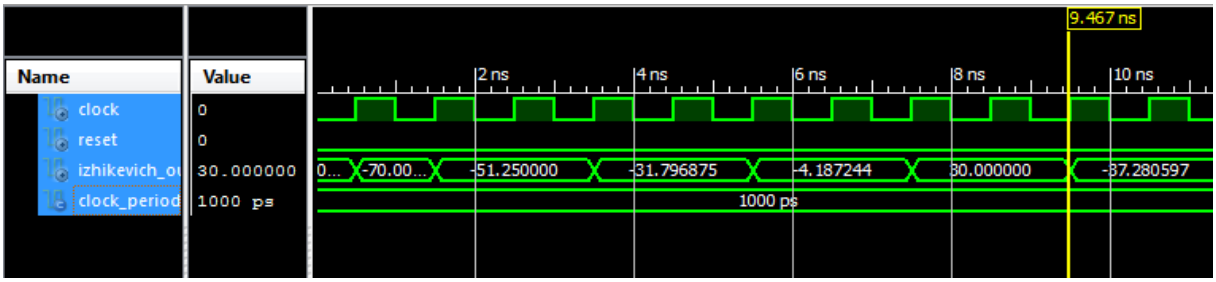


Figure (IV.20): La simulation en ISIM de -Inhibition-Induced Spiking (S)-

La figure ci-dessous présente le processus d'écriture et le fichier texte

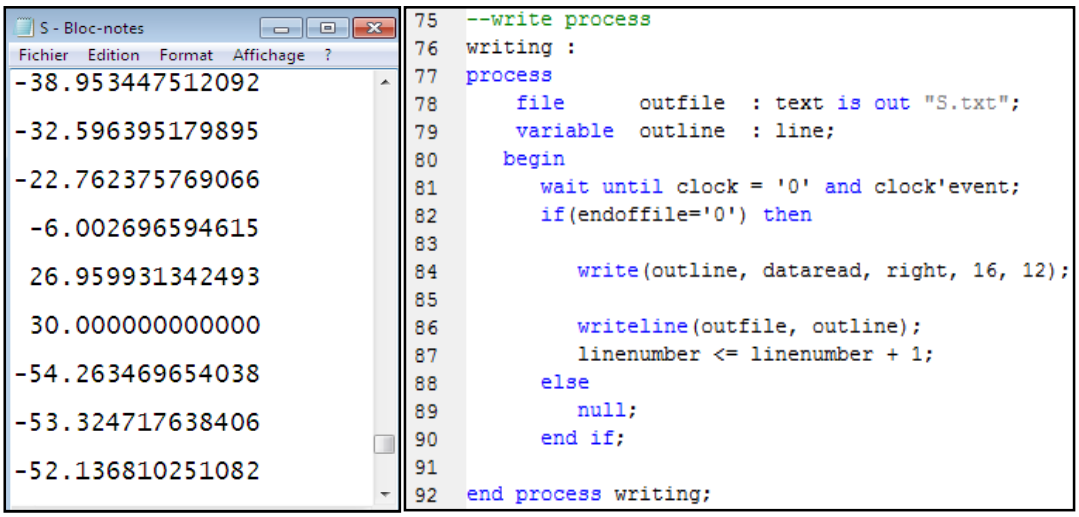


Figure (IV.21): le processus de writting, et le fichier (S.txt) d'Inhibition-Induced-Spiking (S)

IV.2.1.A.8. Inhibition-Induced-Bursting (T)

```
10
11  entity izhik_T is
12      Port ( clock : in  STD_LOGIC;
13             reset : in  STD_LOGIC;
14             izhikevich_out : inout  real
15
16             );
17  end izhik_T;
18
19  architecture Behavioral of izhik_T is
20
21  signal a: real:=-0.026;
22  signal b: real:=-1.0;
23  signal c: real:=-45.0;
24  signal d: real:=-2.0;
25
26  signal v: real:=-63.8;
27  signal u: real:=63.8;
28  signal vv: real:=0.0;
29
30  signal I1:real;
31
```

Figure (IV.22): Un aperçu sur la description VHDL d'Inhibition-Induced-Bursting(T)

Le résultat de la simulation en Xilinx deInhibition-Induced-Bursting(T) est figuré ci-dessous :

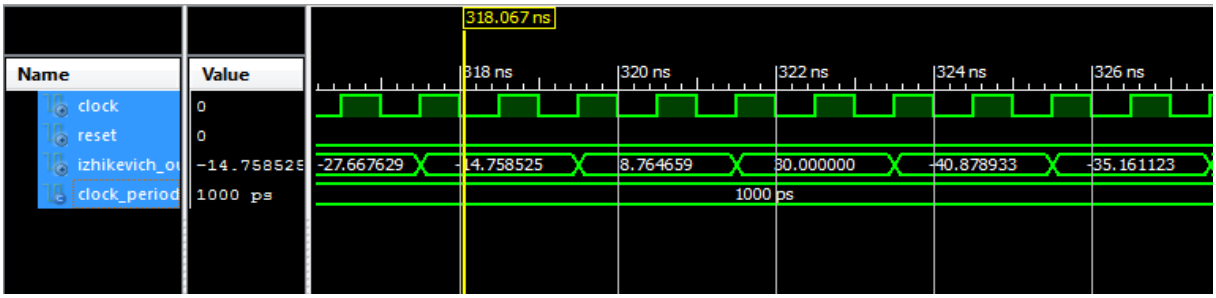


Figure (IV.23): La simulation en ISIM de -Inhibition-InducedBursting (T)

La figure ci-dessous présente le processus d'écriture et le fichier texte

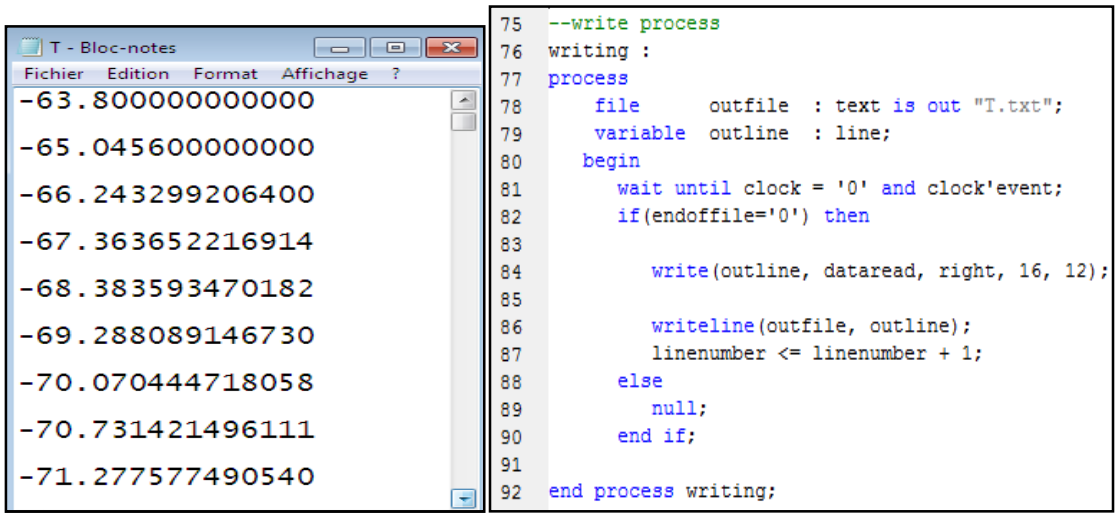


Figure (IV.24): le processus de writting, et fichier (T.txt) de: Inhibition-Induced-Bursting (T)

IV.2.1.B. la description de la version de type virgule fixe:

Nous avons présenté quelques résultats de la simulation de la version en type virgule fixe pour différentes valeurs des paramètres (a, b, c, d)

IV.2.1.B.1 Tonic Spiking (A):

La figure suivante représente un aperçu sur la description VHDL en type virgule fixe de Tonic Spiking (A) :

```
16  entity izhik_sfixed_X is
17      Port ( clock : in  STD_LOGIC;
18             reset : in  STD_LOGIC;
19             izhikevich_out_real : inout real;
20             izhikevich_out_real_unsg : inout real;
21             izhikevich_out : inout sfixed (7 downto -16)
22
23             );
24  end izhik_sfixed_X;
25
26  architecture Behavioral of izhik_sfixed_X is
27
28  signal a: sfixed (7 downto -16):= to_sfixed(0.02,7,-16);
29  signal b: sfixed (7 downto -16):= to_sfixed(0.2,7,-16);
30  signal c: sfixed (7 downto -16):= to_sfixed(-65.0,7,-16);
31  signal d: sfixed (7 downto -16):= to_sfixed(6.0,7,-16);
32
33  signal v: sfixed (7 downto -16):= to_sfixed(-70.0,7,-16);
34  signal u: sfixed (7 downto -16):= to_sfixed(-14.0,7,-16);
35  signal vv: sfixed (7 downto -16):= to_sfixed(-70.0,7,-16);
36
37  signal I1:sfixed (7 downto -16):= to_sfixed(14.0,7,-16);
```

Figure (IV.25): Un aperçu sur la description VHDL de Tonic Spiking (D)

La simulation en Xilinx (ISIM) de la description VHDL en type virgule fixe de Tonic Spiking est donnée par la figure suivante :

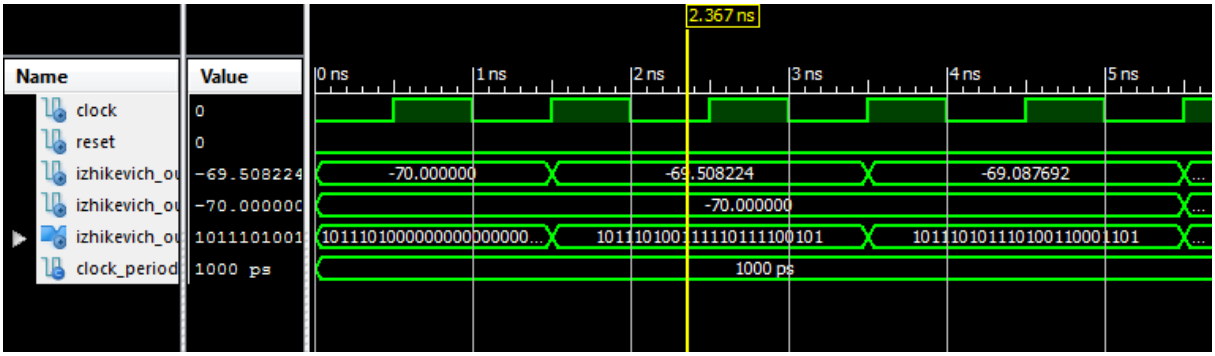
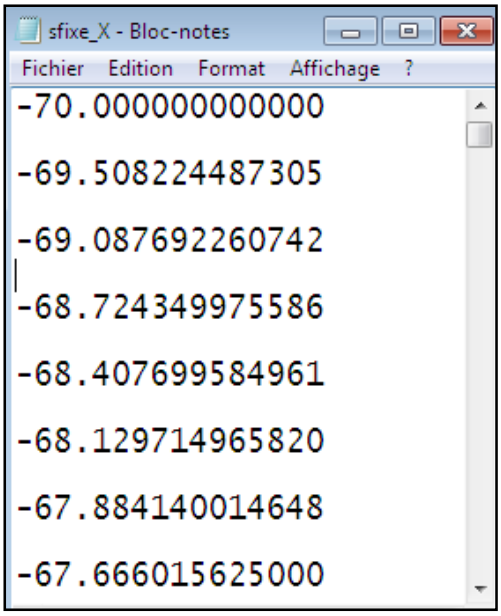


Figure (IV.26): La simulation en ISIM de Tonic Spiking (A) en virgule fixe

La figure ci-dessous présente le processus d'écriture et le fichier texte de Tonic Spiking (A) en type virgule fixe :




```
75  --write process
76  writing :
77  process
78      file      outfile  : text is out "sfixed_X.txt";
79      variable  outline  : line;
80    begin
81       wait until clock = '0' and clock'event;
82       if(endoffile='0') then
83
84          write(outline, dataread, right, 16, 12);
85
86          writeline(outfile, outline);
87          linenumber <= linenumber + 1;
88       else
89          null;
90       end if;
91
92  end process writing;
```

Figure (IV.27): le processus de writting, et fichier (T.txt) de Tonic Spiking (A)

IV.2.1.B.2 Phasic bursting (D):

La figure suivante représente un aperçu sur la description VHDL en type virgule fixe de Phasic bursting (D) :

```
16  entity izhik_sfixed is
17      Port ( clock : in  STD_LOGIC;
18             reset : in  STD_LOGIC;
19             izhikevich_out_real : inout real;
20             izhikevich_out_real_unsg : inout real;
21             izhikevich_out : inout sfixed (7 downto -16)
22
23             );
24  end izhik_sfixed;
25
26  architecture Behavioral of izhik_sfixed is
27
28  signal a: sfixed (7 downto -16):= to_sfixed(0.02,7,-16);
29  signal b: sfixed (7 downto -16):= to_sfixed(0.25,7,-16);
30  signal c: sfixed (7 downto -16):= to_sfixed(-55.0,7,-16);
31  signal d: sfixed (7 downto -16):= to_sfixed(0.05,7,-16);
32
33  signal v: sfixed (7 downto -16):= to_sfixed(-64.0,7,-16);
34  signal u: sfixed (7 downto -16):= to_sfixed(-16.0,7,-16);
35  signal vv: sfixed (7 downto -16):= to_sfixed(-64.0,7,-16);
36
37  signal I1:sfixed (7 downto -16):= to_sfixed(0.6,7,-16);
38
```

Figure (IV.28): Un aperçu sur la description VHDL de Phasic Bursting (D)

La simulation en Xilinx (ISIM) de la description VHDL en type virgule fixe de Phasic Bursting est donnée par la figure suivante :

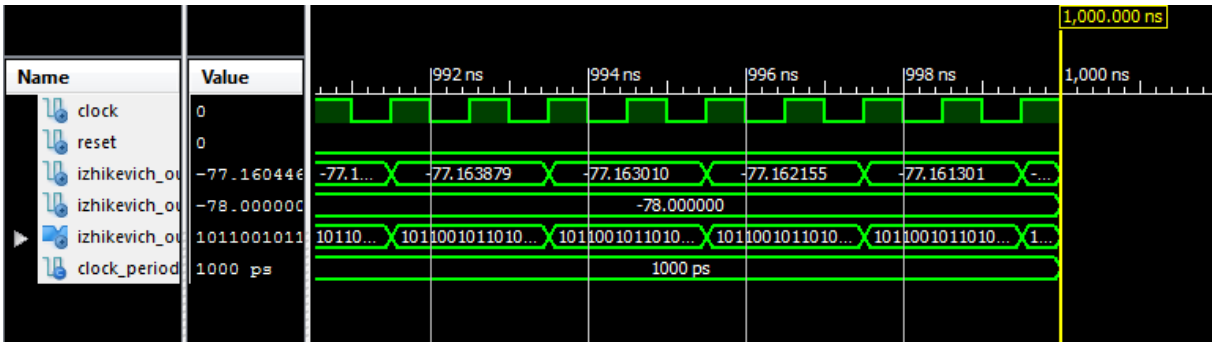


Figure (IV.29): La simulation en ISIM de Phasic Bursting (D) en virgule fixe

La figure ci-dessous présente le processus d'écriture et le fichier texte de Phasic Bursting en type virgule fixe :

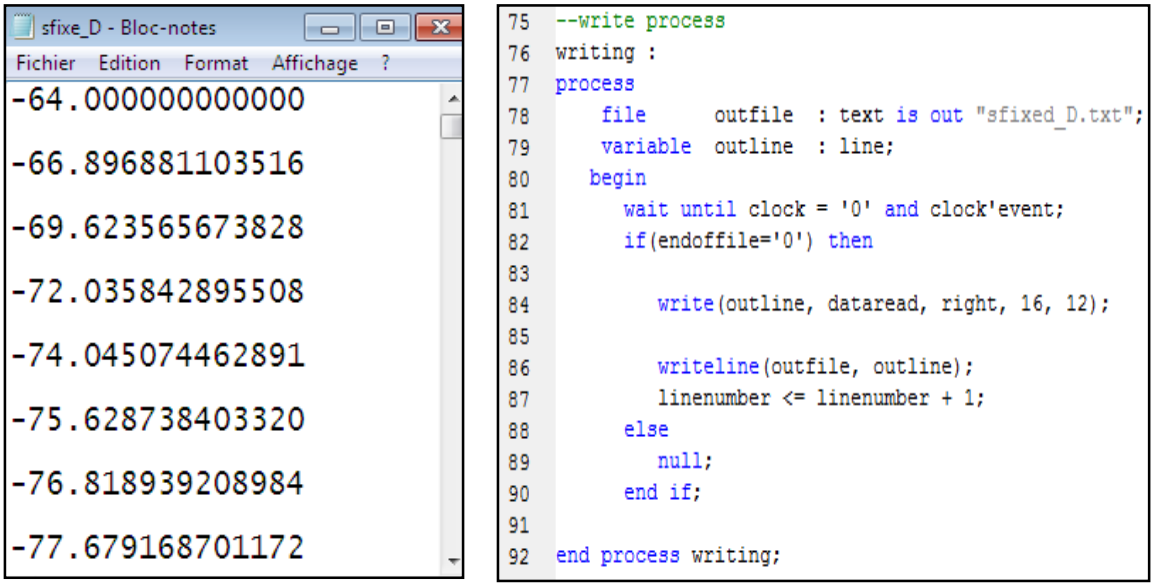


Figure (IV.30): le processus de writting, et fichier (T.txt) de Phasic Bursting (D)

IV.2.2. La représentation des résultats en MATLAB

Après que nous avons simulé la description VHDL en XIlinx (ISIM) et stocké les résultats dans des fichiers textes, nous avons tracé ces derniers en Matlab. Les figures suivantes montrent les graphes de notre simulation pour différentes valeurs des paramètres (a, b, c, d) du modèle Izhikevich.

IV.2.2.A. Les résultats de la simulation de version de type réel :

On va présenter les résultats obtenus de la simulation VHDL de la version de type réel, en variant les quatre paramètres : (a, b, c, et d) et le variable (I)

IV.2.2. A.1.Tonic spiking (A)

Nous avons dessiné les valeurs stockés dans le fichier texte (2.txt) de Tonic Spiking en Matlab. (Voir la figure suivante).

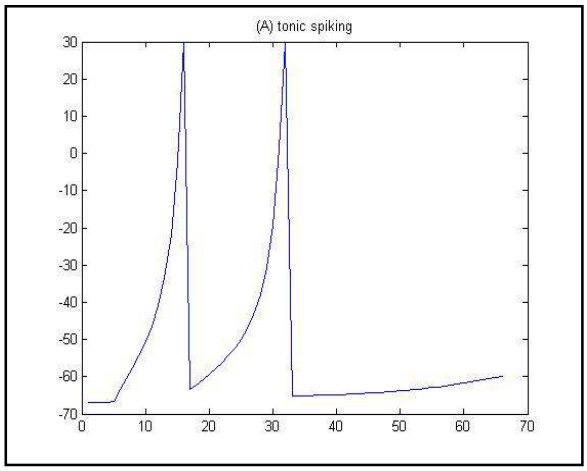


Figure (IV.31): le graphe de Tonic Spiking (A) de la simulation VHDL en MAtlab

IV.2.2.A.2.Phasic spiking (B)

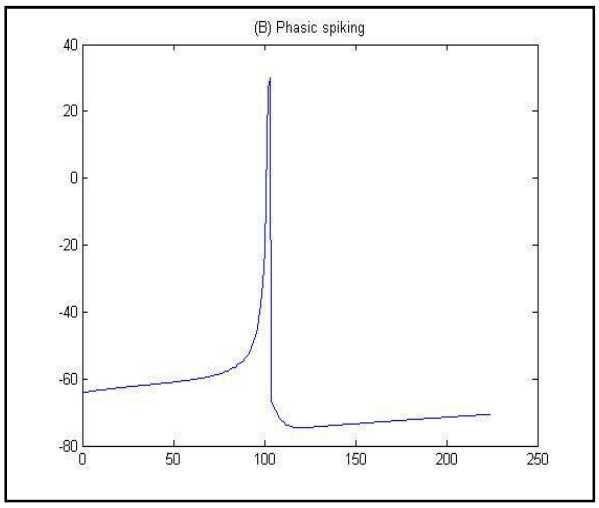


Figure (IV.32): le graphe de Phasic Spiking (B) de la simulation VHDL en Matlab

IV.2.2.A.3.Tonic Bursting (C)

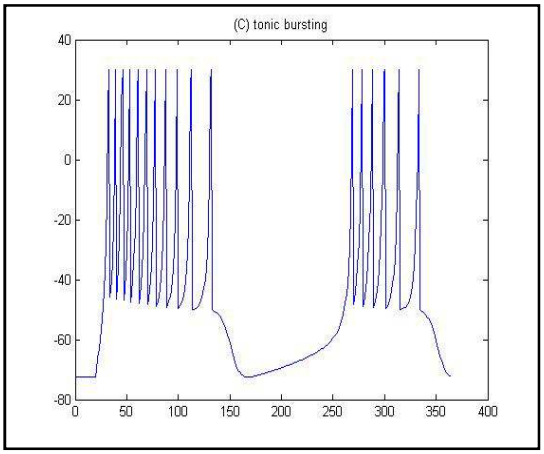


Figure (IV.33): le graphe de Tonic Bursting (C) de la simulation VHDL en Matlab

IV.2.2.A.4.Phasic Bursting (D)

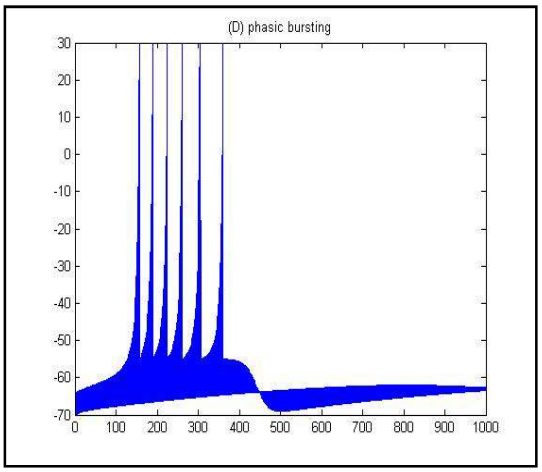


Figure (IV.34): le graphe de Phasic bursting (D) de la simulation VHDL en Matlab

IV.2.2.A.5.Mixed Mode (E)

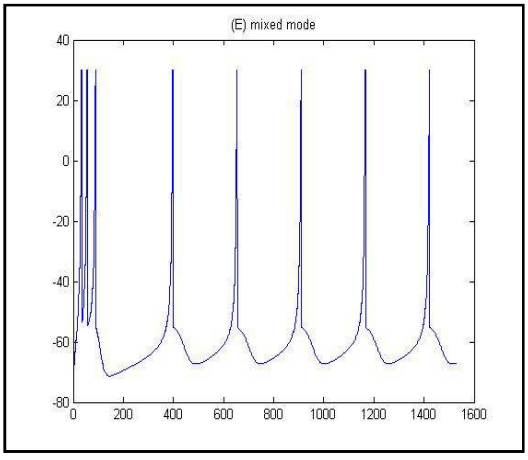


Figure (IV.35): le graphe de Mixed Mode (E) de la simulation VHDL en Matlab

IV.2.2.A.6.Spike frequency adaptation (F)

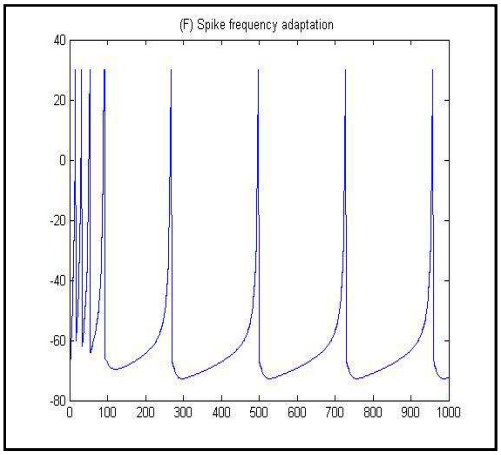


Figure (IV.36): le graphe Spike Frequency Adaptation de (F) de la simulation VHDL en Matlab

IV.2.2.A.7.Inhibition induced spiking (S)

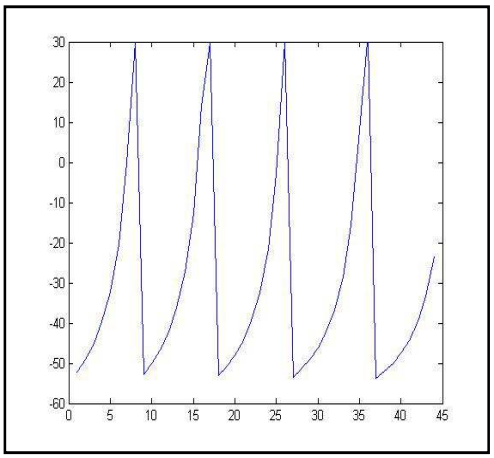


Figure (IV.37): le graphe de Inihibition induced spiking (S) de la simulation VHDL

IV.2.2.A.8.Inhibition induced bursting (T)

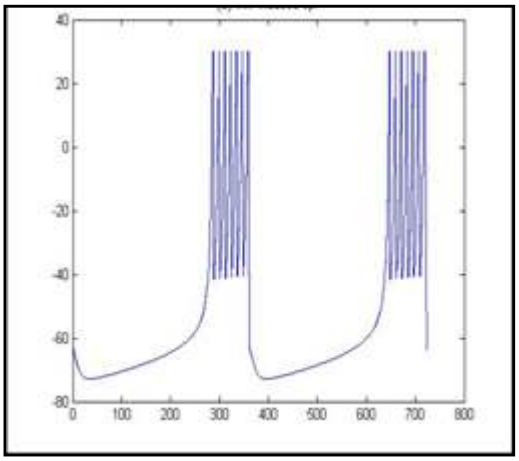


Figure (IV.38): le graphe de .inhibition induced bursting (T) de la simulation VHDL

IV.2.2.B. Les résultats de la simulation de la version de type virgule fixe :

Dans cette partie, nous avons présenté quelques résultats obtenus de la simulation de la méthode « virgule fixe »

IV.2.2.B.1. Tonic Spiking (A)

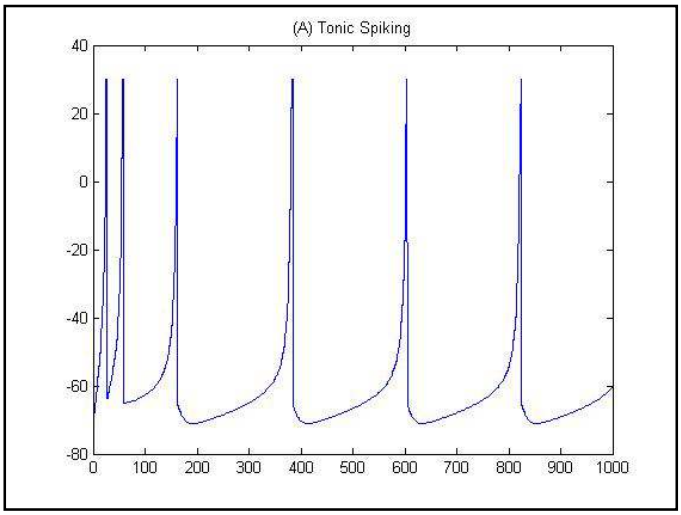


Figure (IV.39): le graphe de Tonic Spiking (A) en type « virgule fixe» en Matlab

IV.2.2.B.2.Phasic Bursting (D)

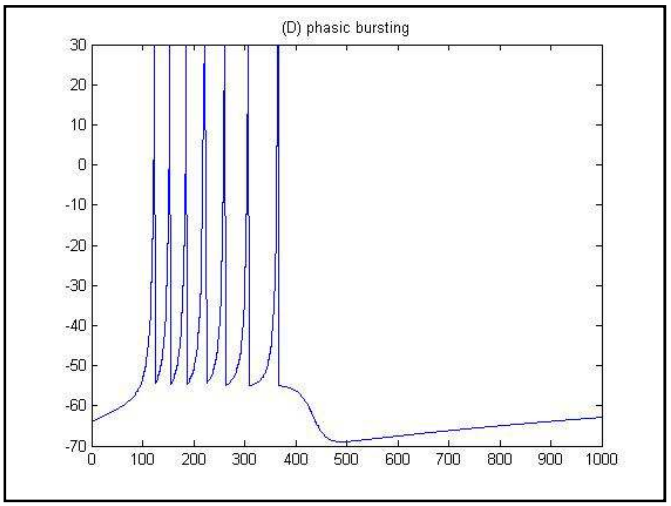


Figure (IV.40): le graphe de Phasic Bursting (D) en type « virgule fixe» en Matlab

En comparant les résultats de la méthode « réels » avec les résultats d'**Eugene.Izhikevich** en Matlab, nous avons constaté qu'elle était correcte et les résultats sont proches de celles d'**Eugene.Izhikevich** en Matlab.

Et pour les résultats de la méthode « virgule fixe » sont plus précises de celles de la méthode « réel » et plus proches aux résultats de M.**Eugene.Izhikevich**

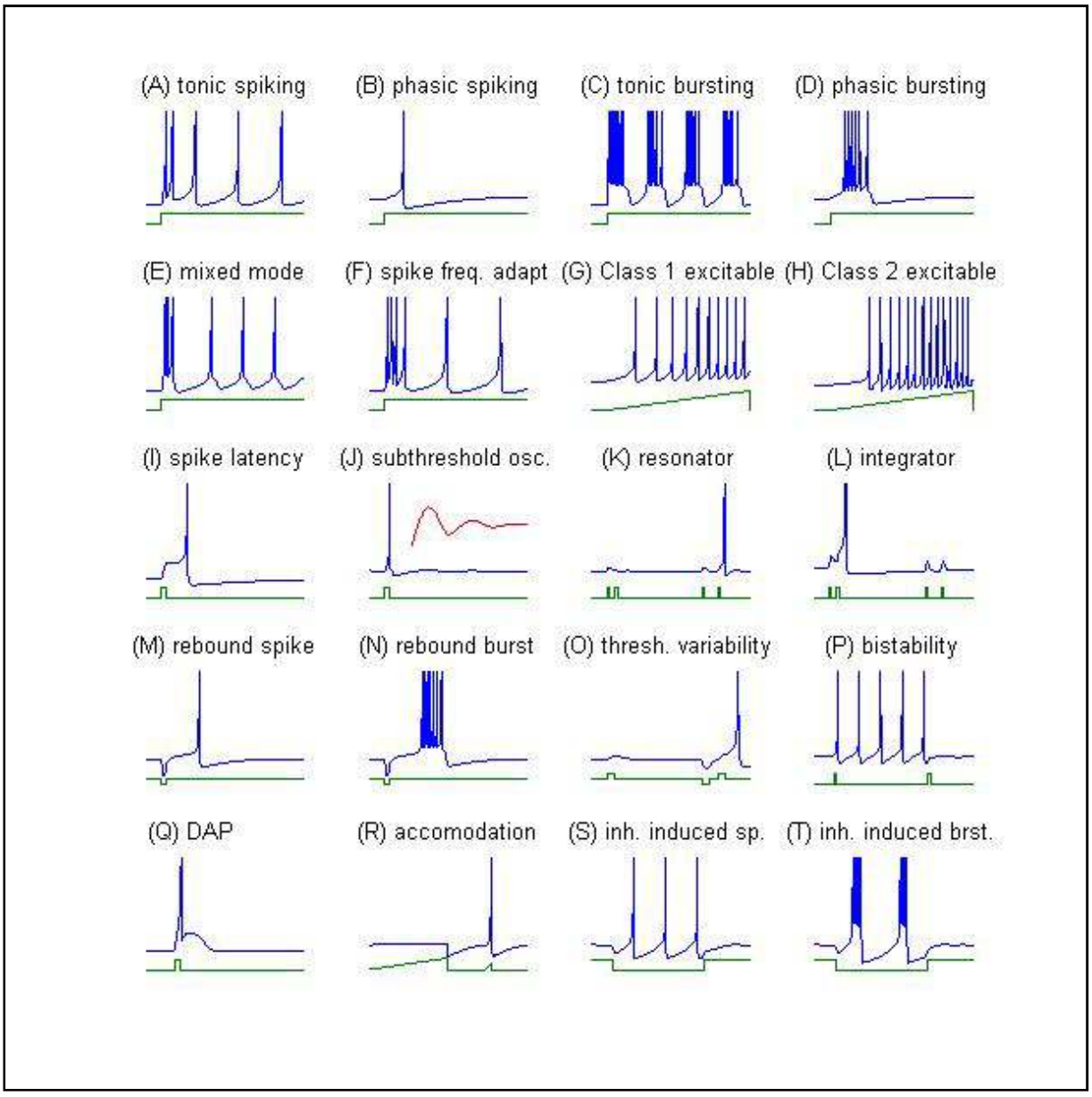


Figure (IV.41): le graphe de tous les spiking neurones d'Eugene.Izhikevich en Matlab

IV.3. Comparaison de la version « réel » et la version « virgule fixe » par rapport à Matlab :

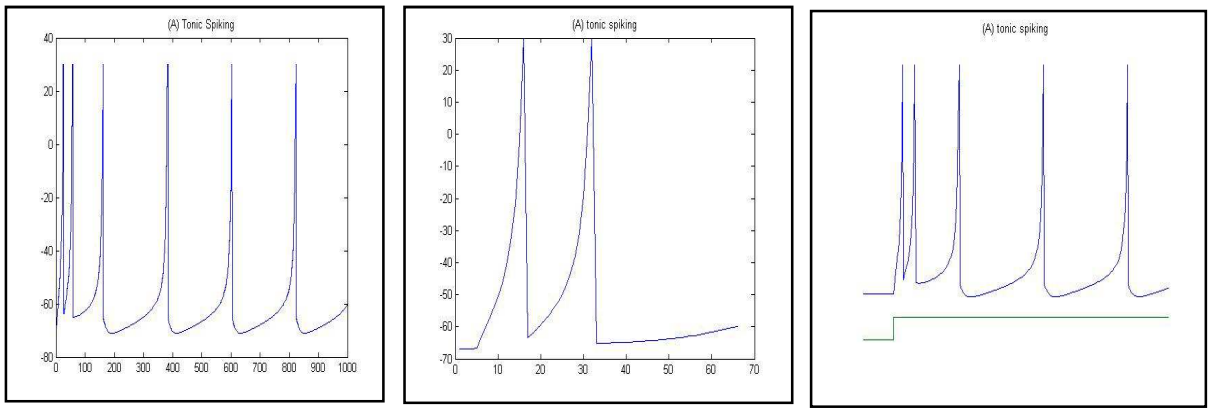
Nous avons présenté quelques exemples des résultats de la version « réelé et virgule fixe » et les comparé avec celle de Matlab

IV.3.1. Tonic Spiking

(a): Tonic Spiking en utilisant la méthode des nombres en virgule fixe en VHDL

(b): Tonic Spiking en utilisant la méthode des nombres réels en VHDL

(c): Tonic Spiking de M.**Eugene.Izhikevich** en Matlab



(a) (b) (c)

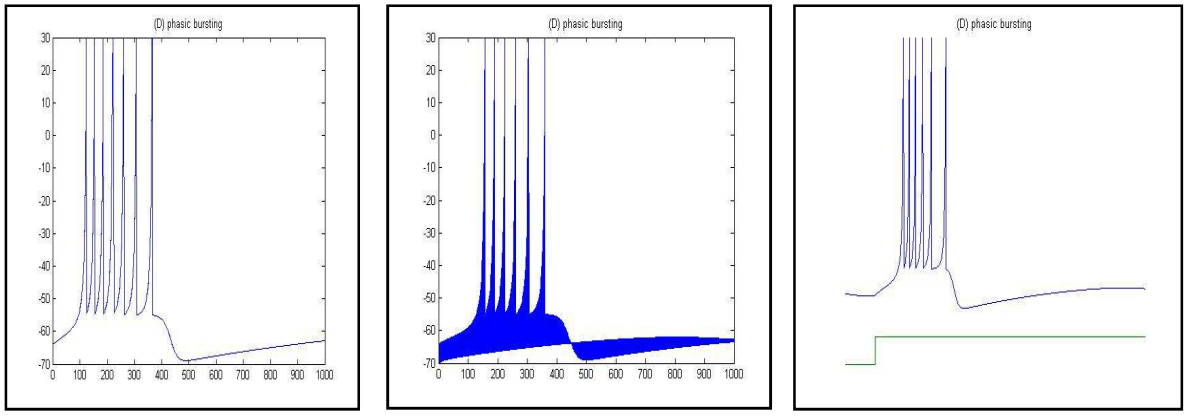
Figure (IV.42) : Comparaison de Tonic Spiking entre « réel », « virgule fixe » et « Matlab »

IV.3.2. Phasic Bursting

(43.a): Phasic Bursting en utilisant la méthode des nombres en virgule fixe en VHDL

(43.b): Phasic Bursting en utilisant la méthode des nombres réels en VHDL

(43.c): Phasic Bursting de M.**Eugene.Izhikevich** en Matlab



(a) (b) (c)

Figure (IV.43) : Comparaison de Phasic Bursting entre « réel », « virgule fixe » et « Matlab »

En comparant les deux méthodes (réels et virgule fixe) par rapport à celle de Matlab, nous avons constaté que les deux versions nous donnent des meilleurs résultats, mais la version utilisant les nombres en virgule fixe était plus précise que celle utilisant les nombres réels.

Les résultats de la simulation VHDL en utilisant les nombres réels sont corrects, mais la description de ce type est faite uniquement pour la simulation et pas pour l'implantation sur circuit FPGA.

Par contre la description VHDL en utilisant les nombres à virgule fixe est implantable sur le circuit FPGA, et ses résultats sont plus proches de celles de M.**Eugene.Izhikevich** en Matlab et plus précises.

IV.4.Conclusion

Dans ce dernier chapitre, nous avons présenté les résultats de description VHDL et de la simulation de notre modèle Izhikevich, en montrant tous les résultats et les graphes de toutes les réponses neuronales des deux versions (réel, et virgule fixe) et les comparé avec Matlab.

Une description vhdl aboutit à un circuit et non à un simple programme, c'est pour ça nous avons fait la comparaison entre les résultats de description VHDL des nombres réels et des nombres à virgule fixe, en prend toujours en référence les réponses standards de **Eugene. Izhikevich**.

Conclusion Générale et Perspectives

Conclusion générale

Les réseaux de neurones impulsionnels ont prouvé leur effecacité grace à leur puissance d'apprentissage et ceci en donnant les meilleurs résultats d'évaluation comparés aux réseaux de neurones classiques. Mais l'emploi des algorithmes consommateurs de ressources mémoires dans ces réseaux nécessitent un temps très important qui est précieux dans le domaine médical où un traitement des données en temps réel est souvent requis.

L'implantation de ces algorithmes sur circuit permet d'augmenter la vitesse de calcul (x1000, x10000, et plus).

Pour cela, nous avons proposé dans le cadre de ce mémoire l'utilisation des circuits programmables FPGA qui apportent une grande flexibilité d'emploi et qui permettent une réduction de temps en assurant un fonctionnement parallèle tout en donnant une dynamique de reconfiguration, ce qui est avantageux par rapport aux autres circuits.

Nous avons implémenté, avec succès le modèle du neurone impulsionnel Izhikevich en VHDL en utilisant deux versions (réel et virgule fixe). Nous avons présenté les résultats des simulations que nous avons obtenus à partir de la description VHDL pour différentes valeurs des paramètres (a, b, c, d) du modèle Izhikevich. En comparant les deux méthodes, nous avons constaté que la version utilisant les nombres représentés en virgule fixe était plus précise que celle utilisant les nombres réels. Cela s'explique par le fait que les nombres réels, contrairement aux nombres représentés en virgule fixe, ne sont pas synthétisables ; on peut les simuler mais pas les implémenter sur circuit.

Nous avons ensuite stocké les données obtenues sous xilinx dans des fichiers, que nous avons transmis à MATLAB pour tracer les courbes de réponses du modèle. Nous avons pu ainsi valider notre implémentation VHDL en comparant les courbes tracées à partir des résultats obtenus sous xilinx, avec celles obtenues directement sous MATLAB.

Nous avons vu dans cette thèse que l'implémentation d'un réseau de neurones impulsionnel sur FPGA est une tâche très complexe, car elle exige une connaissance et une maitrise très avancées des technologies relatives à ce type de circuit et de leurs environnements de développements. Malgré ces inconvénients, il reste très intéressant d'exploiter ce type de circuit car il nous offre des performances qu'on les trouve pas dans les autres circuits. On peut les résumer à leur puissance de calcul parallèle qui nous permet d'économiser le temps, en plus de leur flexibilité de configuration et reconfiguration, ils possèdent une gestion dynamique de connectivité à leur environnement, ce qui permet non seulement d'implémenter les différents algorithmes adéquats à des applications très pointues, mais tout en gardant une grande flexibilité de communication à des environnements très variés en conservant toujours l'efficacité énergétique et les performances nécessaires.

Perspectives

Les résultats très encourageants obtenus nous poussent donc à pour suivre le traitement de l'information par les réseaux de neurones impulsionnels, qui sera utilisée dans le développement de modèles de traitement de l'information dans le domaine de la vision artificielle.

Notre travail constitue aussi le début de la création d'une TOOL BOX pour la conception de réseaux de neurones sous vhdl. Le langage de description VHDL étant standard, cette tool-box pourrait être utilisée par différentes marques de circuits FPGA (XILINX, ALTERA..). D'autres types de neurones pourront également être ajoutés à la bibliothèque.

Dans notre projet nous avons fait une description VHDL d'un réseau de neurones impulsionnels (modèle Izhikevich), nous pouvons plus-tard faire une description en langage **Verilog**

Abréviations

Abréviations

PLD	**P**rogrammable **L**ogic
EPLD	**E**rasable **P**rogrammable **L**ogic **D**evice = circuit logique programmable et effaçable
ASIC	**A**pplication-**S**pecific **I**ntegrated **C**ircuit = circuit intégré propre à une application
FPGA	**F**ield **P**rogrammable **G**ate **A**rrays
CAO	**C**onception **A**ssistée par **O**rdinateur
LCA	**L**ogic **C**ell **A**rray
CLB	**B**locs **L**ogiques **P**rogrammables
IOB (BES)	**B**locs d'**E**ntrées/**S**orties
SRAM	**S**tatic **R**andom **A**ccess **M**emory
VHSIC	**V**ery **H**igh **S**peed **I**ntegrated **C**ircuit
VHDL	**V**hsic **H**ardware **D**escription **L**anguage

Références Bibliographiques

Bibliographie

Livres:

- C.ALEXANDRE. « Circuits logiques programmables ». Editeur Ellipses. (1998)
- Eugene M. Izhikevich. « Dynamical Systems in Neuroscience: The Geometry of Excitability and Bursting». The Mit Press. London, England. (2007)
- Pierre Borne. «Les réseaux de neurones: Présentation et applications». Méthodes et pratiques de l'ingénieur. (2007)

Thèses:

- Anthony Mouraud. « Approche distribuée pour la simulation événementielle de réseaux de neurones impulsionnels : Application au contrôle des saccades oculaires». Thèse de Doctorat. l'Université des Antilles et de la Guyane. (2009)
- Javier Gómez Casado. «Implementation of an artificial neuron in VHDL». Espagne (2008)
- Lzeboudjen.N, Farah*.A, Titri.S, et Boumeridja.H. «Digital Implementation of Artificial Neural Networks: From VHDL Description to FPGA Implementation». Ecole Nationale Polytechnique, El Harrach, Algiers- Algeria. (1998)
- MARC-ANDRÉ CANTIN. «IMPLANTATIONS DU RÉSEAU DE NEURONES-FUZZY ART». Mémoire présenté comme exigence partielle de La Maîtrise En Informatique. L'Université du Québec à Montréal. (1998)
- Romain Brette. «Modèles Impulsionnels de Réseaux de Neurones Biologiques». Thèse de Doctorat. Ecole Doctorale à Paris, France. (2003)
- Serge Raoul DZONDE NAOUSSI. « Implantation des réseaux neuromimetiques sur cible FPGA- application à l'intégration d'un système de filtrage actif »
- Zahi Ait Ouali. « Application des FPGA à la commande d'un moteur asynchrone »

Articles:

- Eugene M. Izhikevich. «Which Model to Use for Cortical Spiking Neurons» IEEE TRANSACTIONS ON NEURAL NETWORKS, VOL.15, NO.5, SEPTEMBER. (2004)
- Eugene M. Izhikevich. «Simple Model of Spiking Neurons» IEEE TRANSACTIONS ON NEURAL NETWORKS, VOL. 14, NO. 6, NOVEMBER. (2003)
- GUELLAL Amar. « Les circuits FPGA : description et applications ». (2010)

Sites internets:

- http://www.izhikevich.org/
- http://opensourcebrain.org/projects/izhikevichmodel/wiki
- http://senselab.med.yale.edu/ModelDB/ShowModel.cshtml?model=39948