ΕΘΝΙΚΟ ΜΕΤΣΟΒΕΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΚΑΙ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΜΙΚΡΟΥΠΟΛΟΓΙΣΤΩΝ ΚΑΙ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

SPICE-Compatible Verilog-AMS Model for Inferior Olive Neurons

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

Παπανικολάου Γεωργίου

Επιβλέπων: Δημήτριος Ι. Σούντρης
Αν. Καθηγητής Ε.Μ.Π.

Αθήνα, Αύγουστος 2015

ΕΘΝΙΚΟ ΜΕΤΣΟΒΕΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΚΑΙ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΜΙΚΡΟΥΠΟΛΟΓΙΣΤΩΝ ΚΑΙ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

SPICE-Compatible Verilog-AMS Model for Inferior Olive Neurons

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

Παπανικολάου Γεωργίου

Επιβλέπων: Δημήτριος Ι. Σούντρης
Αν. Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την ημερομηνία εξέτασης.

---	---	---
Δημήτριος Ι. Σούντρης	Κιαμάλ Πεκμεστζή	Γιώργος Ματσόπουλος
Αναπληρωτής Καθηγητής	Καθηγητής	Αναπληρωτής Καθηγητής

Αθήνα, Αύγουστος 2015

ΕΘΝΙΚΟ ΜΕΤΣΟΒΕΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΚΑΙ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΜΙΚΡΟΥΠΟΛΟΓΙΣΤΩΝ ΚΑΙ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Γεώργιος Παπανικολάου
Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Ηλ. Υπολογιστών

Contents

Acknowledgements

I am extremely grateful to have been part of the Microprocessors Laboratory and Digital Systems Lab team for the past 9 months, during which I conducted my graduate thesis project under the supervision of Prof. D.J. Soudris and Mr. D. Rodopoulos.

I would like to thank Prof. D.J. Soudris for all the advice and psychological support he provided me from the very beginning of this period. The discipline mentality and the frame of serious work he reinforced from the start were essential to my motivation for work.

Also, I would like to express my sincere gratitude to my supervisor and mentor, Mr. D. Rodopoulos, for his pleasant cooperation and professionalism. I really enjoyed the process of improving my technical skills under his wing, receiving the exact amount of feedback necessary to keep going.

My special thanks to G.Chatzikonstandis for providing his knowledge immediately and articulately whenever required.

Finally, I would like to thank my friends and family for being part of my journey as a graduate student in the School of Electrical and Computer Engineering of NTUA.

Περίληψη

Η μοντελοποίηση του εγκεφάλου είναι μια εφαρμογή που απασχολεί όλο και περισσότερο την επιστήμη τα τελευταία χρόνια και προσεγγίζεται από διάφορους κλάδους της Βϊοιατρικής όπως είναι η Ιατρική απεικόνιση και η λειτουργική μαγνητική τομογραφία (fMRI) αλλά και η πληροφορική υπερυψηλής απόδοσης (HPC). Τέτοιες τεχνολογίες επιτρέπουν τη συνεργασία σε μεγάλη κλίμακα, το διαμοιρασμό δεδομένων, την ανακατασκευή του εγκαφάλου σε διάφορες βιολογικές κλίμακες και την κατασκευή διαφόρων υπολογιστικών συστημάτων εμπνευσμένων από τον εγκέφαλο. Διάφορα μεγάλα εγχειρήματα [1, 2, 3] επικεντρώνονται στην κατανόηση της ανθρώπινης νοημοσύνης και συμπεριφοράς, αλλά και στην θεραπεία εγκεφαλικών παθήσεων. Πιο συγκεκριμένα, οι προσομοιώσεις στον ανθρώπινο εγκέφαλο θα μπορούσαν να εξηγήσουν πως δρούν τα διάφορα φάρμακα στον εγκέφαλο, ποιές είναι οι παρενέργειές τους και ίσως να βοηθήσουν στην ανακάλυψη νέων τρόπων θεραπείας.

Εν τω μεταξύ, η έννοια "neuromorphic computing" γίνεται όλο και πιο δημοφιλής. Η έννοια αυτή αναφέρεται στην κατασκευή ολοκληρωμένων κυκλωμάτων πολύ μεγάλης κλίμακας με φυσική αρχιτεκτονική και αρχές παρόμοιες εκείνης των νευρικών συστημάτων. Είναι εμφανές ότι υπάρχει μια διαρκής αναζήτηση της κατανόησης των εγκεφαλικών μηχανισμών και λειτουργιών.

Στη συγκεκριμένη διπλωματική εργασία θα εστιάσουμε σε αυτή την αναλογία μεταξύ ηλεκτρικών κυκλωμάτων και νευρικής λειτουργίας με τη χρήση ειδικών εργαλείων σχεδίασης ολοκληρωμένων κυκλωμάτων για την προσομοίωση νευρικών κυτάρων. Συγκεκριμένα, θα χρησιμοποιήσουμε έναν προσομοιωτή τύπου SPICE, ενισχυμένο με ένα VerilogA μοντέλο για το νευρικό κύταρο infoli.

Οι in-silico προσομοιώσεις απαιτούν την υιοθέτηση συγκεκριμένων νευρικών μοντέλων ώστε να επιτευχθεί η κατανόηση της εγκεφαλικής λειτουργίας. Δύο βασικές κατηγορίες νευρικών μοντέλων είναι τα υπολογιστικά και τα συμβατικά [10]:

• Συμβατικά μοντέλα: Μοντέλα που περιγράφονται με τη χρήση μαθηματικών εξισώσεων για να αναδείξουν την λειτουργιά των μηχανισμών χαμηλού επιπέδου στους οποίους οφείλεται και η ύπαρξη του εκάστοτε φαινομένου.

• Υπολογιστικά μοντέλα: Τα υπολογιστικά μοντέλα μπορούν να παρουσιαστούν ως μια πιο αφηρημένη περιγραφή της λειτουργιάς, ή ως μια πιο συμπαγή περιγραφή των αναπαραστάσεων και των αλγορίθμων που υιοθετήθηκαν για να περιγράψουν τη λειτουργία, μέχρι και ως μια περιγραφή του πως υλοποιήθηκαν αυτοί οι αλγόριθμοι και αναπαραστάσεις.

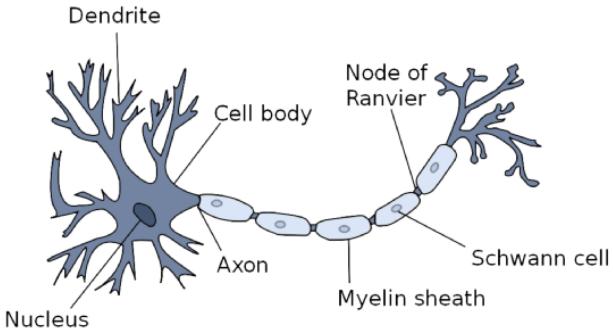
Εναλλακτική κατηγοριοποίηση των νευρικών μοντέλων μπορεί να είναι η παρακάτω:

• integrate-fire Μοντέλα: Περιγράφουν το δυναμικό μεμβράνης σε σχέση με το ρεύμα εισόδου που λαμβάνει και την κατάσταση των συνάψεων. Οι αλλαγές στην τάση μεμβράνης και την αγωγιμότητα δεν αποτελούν μέρος του προβλήματος, παρά μόνο η στοχαστική διαδικασία της ένεργοποίησης των δυναμικών δράσης αφού ξεπεραστεί ένα κατώφλι [11].

• Μοντέλα Izhikevich: Βιολογικώς έγκαιρα μοντέλα με υπολογιστική απλότητα [12], που θεωρεί τους νευρώνες ως ¨μαύρα κουτιά' που προκαλούν ή όχι εκρήξεις δυναμικού ενεργείας.

• Μοντέλα αγωγιμότητας: Είναι βασισμένα σε μια αντίστοιχη κυκλωματική αναπαράσταση μιας κυταρικής μεμβράνης όπως παρουσιάστηκε αρχικά από τους Hodgkin και Huxley [13].

Η δομή του νευρώνα που θα μας απασχολήσει στη συνέχεια φαίνεται στο Σχήμα ;;.



Σχήμα 1: Απεικόνιση της δομής ενός νευρώνα και των επιμέρους τμημάτων του [17]

Κάθε τμήμα του νευρώνα συμβάλλει με το δικό του ρόλο στη συνολική λειτουργία του νευρικού συστήματος. Τα διάφορα τμήματα του νευρώνα όπως απεικονίζονται στο Σχήμα 1 είναι τα εξής:

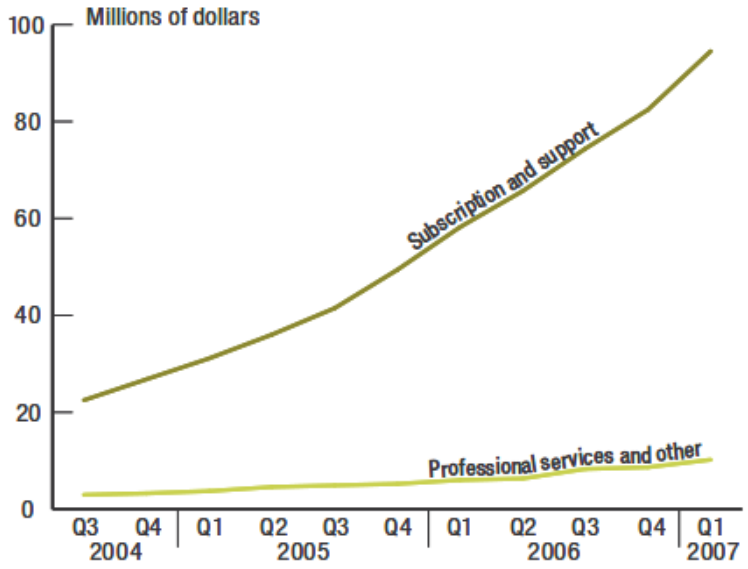
- Δενδρίτες: Οι δενδρίτες παίρνουν την είσοδο τους απο γειτονικά κύταρα μέσω των συνάψεων και στέλνουν την έξοδο στο σώμα [17].
- Σώμα: Μέσω ηλεκτροχημικών αντιδράσεων, το σώμα αθροίζει διεγερτικά ή ανασταλτικά σήματα από τα συναπτικά κομβία ή άλλους δενδρίτες. Αν το άθροισμα των σημάτων ξεπεράσει μια συγκεκριμένη τιμή (κατώφλι) τότε το σώμα ενεργοποιεί έναν ηλεκτρικό παλμό που μεταφέρεται σε γειτονικά κύταρα [17].
- Άξονας: Ο άξονας είναι μια προέκταση του σώματος που συμβάλει στη μετάδοση των παλμών σε άλλα νευρικά κύτταρα.
- Κύταρα σχηματιμσμού Μυελίνης: Αποτελούνται από τον Νευροάξονα (Schwann cells, απομονώνουν τον άξονα απο ηλεκτρική δραστηριότητα και οι Κόμβοι του Ranvier είναι κενά μεταξύ των απομονωτικών κυτάρων [17].

Στα πλαίσια αυτής της εργασίας, μεγάλη έμφαση δίνεται στον τομέα της αυτοματοποίησης ηλεκτρονικού σχεδιασμού (EDA). Τα ετήσια έσοδα τριών από τις μεγαλύτερες εταιρίες στον τομέα αυτό διακαιολογούν την επιρροή που έχει στο σχεδιασμό λογισμικού μοντελοποίησης κυκλωμάτων.

Τα έσοδα παρόχων λογισμικού-ως-υπηρεσίες (Software-as-a-Service) και εμπορικού ανοιχτού λογισμικού (Commercial Open Source) είναι συνήθως πολύ περισσότερο σταθερά ανά τετράμηνο σε σχέση με τα έσοδα από παροχή αδειών επ΄ αόριστον όπως φαίνεται στο Σχήμα ;;.

Ορισμένα εργαλεία που απευθύνονται στον τομέαν των ολοκληρωμένων κυκλωμάτων είναι εργαλεία γνωστά ως εφαρμογές τύπου SPICE. Το SPICE αναπτύχθηκε στο ερευνητικό εργαστήριο Ηλεκτρονικής του πανεπιστημίου του Berkley στην Καλιφόρνια από τον Laurence Nagel [26].

Από τους διάφορους προσομοιωτές τύπου SPICE που έχουν κατασκευαστεί (π.χ HPSICE, ModelSim,Spectre), εμείς θα ασχοληθούμε με το εργαλείο Spectre της Cadence, το οποίο παρέχει γρήγορη και αξιόπιστη προσομοίωση για αναλογικά, ραδιοσυχνοτήτων και μικτού σήματος κυκλώματα και λεπτομερείς αναλύσεις σε επίπεδο τρανζίστορ σε διάφορα πεδία [;].



Σχήμα 2: Έσοδα ανά τετράμηνο ενός αντιπροσωπευτικού παρόχου υπηρεσιών λογισμικού [25]

Η μετάβαση στον προσομοιωτή Spectre είναι μια σχετικά απλή διαδικασία που αποτελείται από τα εξής βήματα:

• Τη μεταφορά του Matlab κώδικα [8] σε ένα συμβατό μετον προσομοιωτή μοντέλο σε γλώσσα VerilogA.

• Την κατασκευή του αρχείου κυκλωματικής περιγραφής (netlist) για τον απλό νευρώνα, όπως φαίνεται παρακάτω:

Αρχείο infoli.scs – Αρχείο περιγραφής απλού νευρώνα

```
simulator lang=spectre
ahdl_include "./infoli.va"

singleneuron (in out) infoli gbar_K=36 gbar_Na=120 g_L=0.3 E_K=-12 E_Na=115
E_L=10.6 C=1

Iinput (in 0) vsource type=pwl file=file.dat

opt1 options saveahdlvars=all
opt2 options rawfmt=psfascii

infoli tran stop=0.1
```

Ο κώδικας που απαρτίζει το VerilogA αποτελείται από τρία μέρη:

- το αρχείο `infoli_define.va`, στο οποίο ορίζονται όλες οι παράμετροι και μεταβλητές.
- το αρχείο `infoli_assign.va`, στο οποίο γίνονται οι απαραίτητες αρχικοποιήσεις κατά το αρχικό στάδιο της προσομοίωσης.

- το αρχείο `infoli_function.va`, που υλοποιεί το ουσιώδες τμήμα της συνάρτησης σε κάθε βήμα της ανάλυσης στο χρόνο.

Τα τρία αυτά αρχείο συμπεριλαμβάνονται στον κώδικα `infoli.va`, ο οποίος καλείται μέσω του αρχείο κυκλωματικής περιγραφής.

Ο κώδικας `infoli.va` φαίνεται παρακάτω:

Ο κώδικας `infoli.va` για την περιγραφή του μοντέλου.

```
‘include "disciplines.h"
‘include "constants.h"

‘define INT_NOT_GIVEN -9999999

module infoli (in,out);

//node definitions
input in;
output out;
current in;
voltage out;

‘include "infoli_define.va"

analog begin

‘include "infoli_assign.va"
‘include "infoli_function.va"

end

endmodule
```

Οι διάφορες μεταβλητές και παράμετροι που χρησιμοποιούμε για την περιγραφή του μοντέλου infoli, καθώς και η χρήση τους, φαίνεται στους παρακάτω πίνακες 1 και 2:

Οι εξισώσεις που διέπουν το μοντέλο είναι παρακάτω:

$$I_{ion} = I_{in} - I_K - I_{Na} - I_L \tag{1}$$

όπου τα I_{Na}, I_K,I_L δίνονται από τις σχέσεις:

$$I_{Na} = m^3 * \overline{g_{Na}} * h * (V_{out}(t) - E_{Na}) \tag{2}$$
$$I_K = n^4 * \overline{g_K} * (V_{out}(t) - E_K) \tag{3}$$
$$I_L = g_L * (V_{out}(t) - E_L) \tag{4}$$

Πίνακας 1: Παράμετροι

Παράμετροι	Περιγραφή
$\overline{g_{Na}}$,$\overline{g_K}$	Τιμές ηλεκτρικών αγωγιμοτήτων που αναπαριστούν τα ιοντικά κανάλια που εξαρτώνται από το χρόνο και την τάση.
$\overline{g_L}$	Γραμμική αγωγιμότητα που αναπαριστά τα ρεύματα διαρροής.
E_L, E_{Na}, E_K	Πηγές έντασης οι τιμές των οποίων καθορίζονται απο τις συγκεντρώσεις των αντίστοιχων ιόντων.
C	χωρητικότητα μεμβράνης ανά μονάδα επιφάνειας

Πίνακας 2: Μεταβλητές

Μεταβλητές	Περιγραφή
n, m, h	Αδιάστατες ποσότητες μεταξύ 0 και 1 που σχετίζονται με την ενεργοποίηση των καναλιών καλίου, νατρίου και απενεργοποίηση καναλιού νατρίου αντίστοιχα.
a_i,b_i	Σταθερές ανάλογες της τάσης αλλά όχι του χρόνου για το ι-οστό κανάλι.
I_{Na},I_K,I_L,I_{ion}	I_{ion} είναι το συνολικό ρεύμα μέσα από τη μεμβράνη που υπολογίζεται από τη σχέση (1).

Η νέα τιμή της τάσης υπολογίζεται μέσω της προσέγγισης Euler πρώτης τάξης ώστε να παραχθεί μια καμπύλη που προσομοιάζει την εξέλιξη της τάσης στο χρόνο. Η τάση εξόδου σε κάθε βήμα της προσομοίωσης δίνεται από:

$$V_{out}(t+1) = V_{out}(t) + deltaT * \frac{I_{ion}}{C} \tag{5}$$

Παρομοίως, οι σταθερές ενεργοποίησης των καναλιών δίνονται από τις παρακάτω εξισώσεις:

$$n = n + deltaT * (a_n * (1-n) - b_n * n) \tag{6}$$
$$m = m + deltaT * (a_m * (1-m) - b_m * m) \tag{7}$$
$$h = h + deltaT * (a_h * (1-h) - b_h * h) \tag{8}$$

Ένα δίκτυο πολλών νευρώνων μπορεί να κατασκευαστεί με παρόμοια λογική. Σύμφωνα με τον κώδικα [9], πλέον λαμβάνουμε υπόψιν και τις διασυνδέσεις μεταξύ των διαφόρων τμημάτων του νευρώνα αυτού καθέαυτού καθώς και τα ρεύματα που οφείλονται στη διαφορά τάσης μεταξύ δενδριτών διαφορετικών νευρώνων.

Η κυκλωματική περιγραφή ενός δικτύου τεσσάρων νευρώνων φαίνεται στο παρακάτω αρχείο:

Αρχείο κυκλωματικής περιγραφής δικτύου τεσσάρων νευρώνων

```
simulator lang=spectre
ahdl_include "./infoli.va"

singleneuron1 (in v_dend2 v_dend3 v_dend4 v_dend1) infoli parameters
singleneuron2 (in v_dend1 v_dend3 v_dend4 v_dend2) infoli parameters
singleneuron3 (in v_dend1 v_dend2 v_dend4 v_dend3) infoli parameters
singleneuron4 (in v_dend1 v_dend2 v_dend3 v_dend4) infoli parameters

Iinput (in 0) vsource type=pwl file=file.dat
```

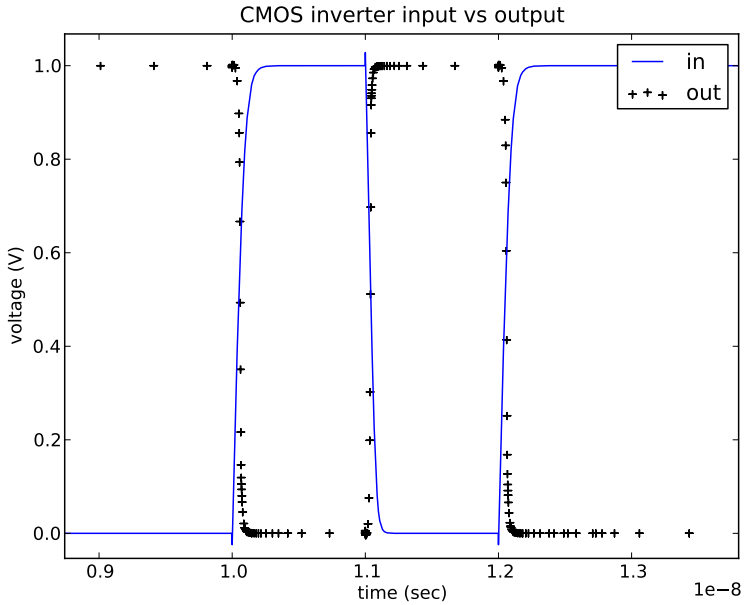
Συγκρίνοντας τους κώδικες MATLAB και VerilogA εντοπίζουμε ορισμένες διαφορές. Σχετικά με τον τρόπο μεταγλώτισης, ο κώδικας MATLAB χρησιμοποιεί τον Matlab Compiler, που κωδικοποιεί και πακετάρει τον κώδικα MATLAB, οποίος στη συνέχεια θα τρέξει με τη βοήθεια του Matlab Compiler Runtime.

Ο προσομοιωτής Spectre δημιουργεί έναν φάκελο μεταγλώτισης ahdl στην αρχή της προσομοίωσης.

Η βασική διαφορά μεταξύ Spectre και MATLAB, είναι το προσαρμοζόμενο βήμα της προσομοίωσης. Στο σχήμα ;; φαίνεται το προσαρμοζόμενο βήμα που χρησιμοποιεί το Spectre για την προσομοίωση ενός αντιστροφέα BSIM4.

Τα πλεονεκτήματα του προσαρμοζόμενου βήματος φαίνονται ήδη από μια απλή προσομοίωση για την επαλήθευση της λειτουργικότητας του VerilogA μοντέλου, στο Σχήμα ;;. Βλέπουμε οτι ο προσομοιωτής Spectre καταφέρνει μια συμπύκνωση στα χρονικά βήματα κατά 90.1%.

Ένα τυπικό αποτέλεσμα πολλών προσομοιώσεων φαίνεται στο Σχήμα ;;, όπου η ακρίβεια της προεπιλογής conservative είναι εμφανής.



Σχήμα 3: Απεικόνιση της χρήσης προσαρμοζόμενου βήματος για την προσομοίωση αντιστροφέα

Στο σχήμα ;; φαίνεται η ακρίβεια και συμπύκνωση χρονικών σημείων για 150 επαναλήψεις. Ο χρόνος εκτέλεσης είναι 100 ms με ρεύματικούς παλμούς εύρους 1 ms. Είναι εμφανές πως υπάρχουν ανεπιθύμητες περιπτώσεις οι οποίες επηρεάζουν αρνητικά την ακρίβεια της προσομοίωσης.

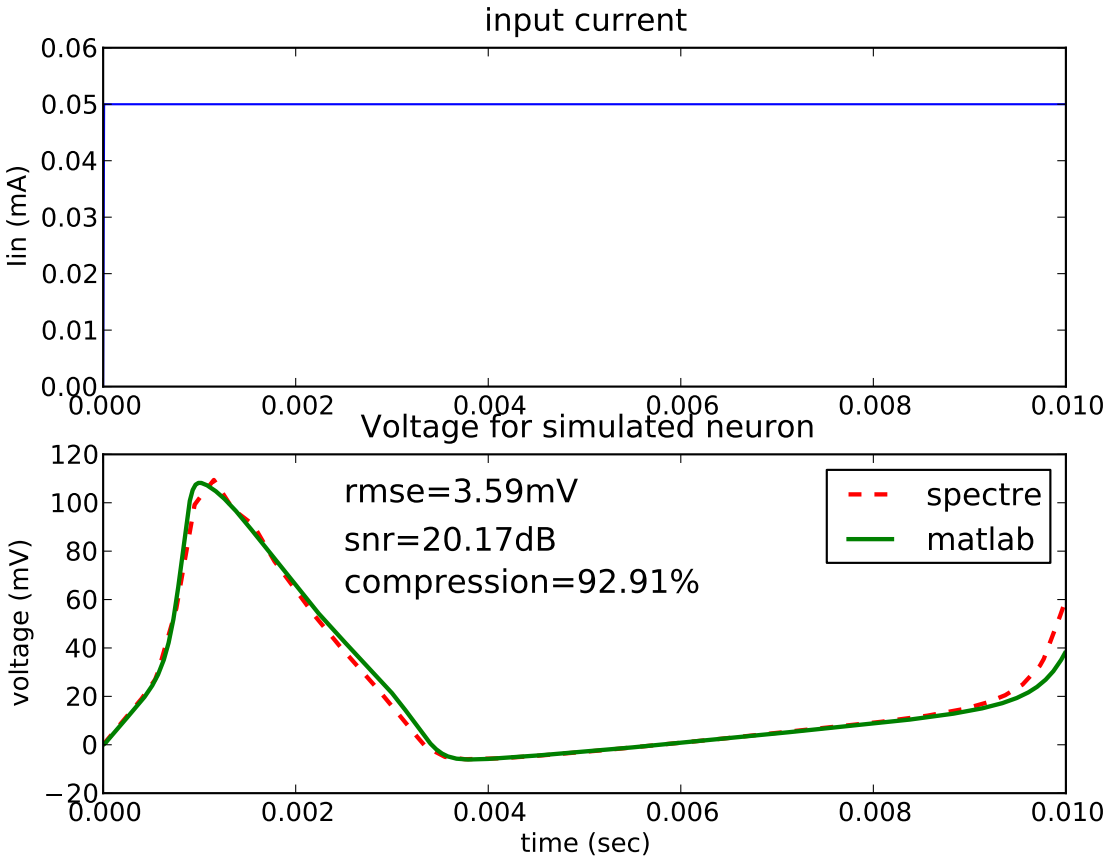
Οι δύο περιπτώσεις στο Σχήμα 7, που εντοπίστηκαν να οδηγούν σε χειροτέρευση του σηματοθορυβικού λόγου φαίνονται παρακάτω και οφείλονται είτε σε διαφορά φάσης μεταξύ των δύο καμπυλών MATLAB και Spectre, είτε σε ανακριβή ενεργοποίηση δυναμικού ενεργείας από τον προσομοιωτή Spectre.

Στη συνέχεια στο Σχήμα 8, μελετάμε τη συμπεριφορά του προσομοιωτή σπεστρε ανάλογα με την εφαρμογή διαφορετικών ρευμάτων στην είσοδο. Για σταθερή διάρκεια προσομοίωσης 400 ms, παρατηρούμε τη συμπεριφορά της ρίζας του μέσου τετραγωνικού σφάλματος (RMSE)και του σηματοθορυβικού λόγου (SNR)σε σχέση με το εύρος παλμού ρεύματος και τη μέση διάρκεια ανάμεσα στην ενεργοποίηση κάθε παλμού.

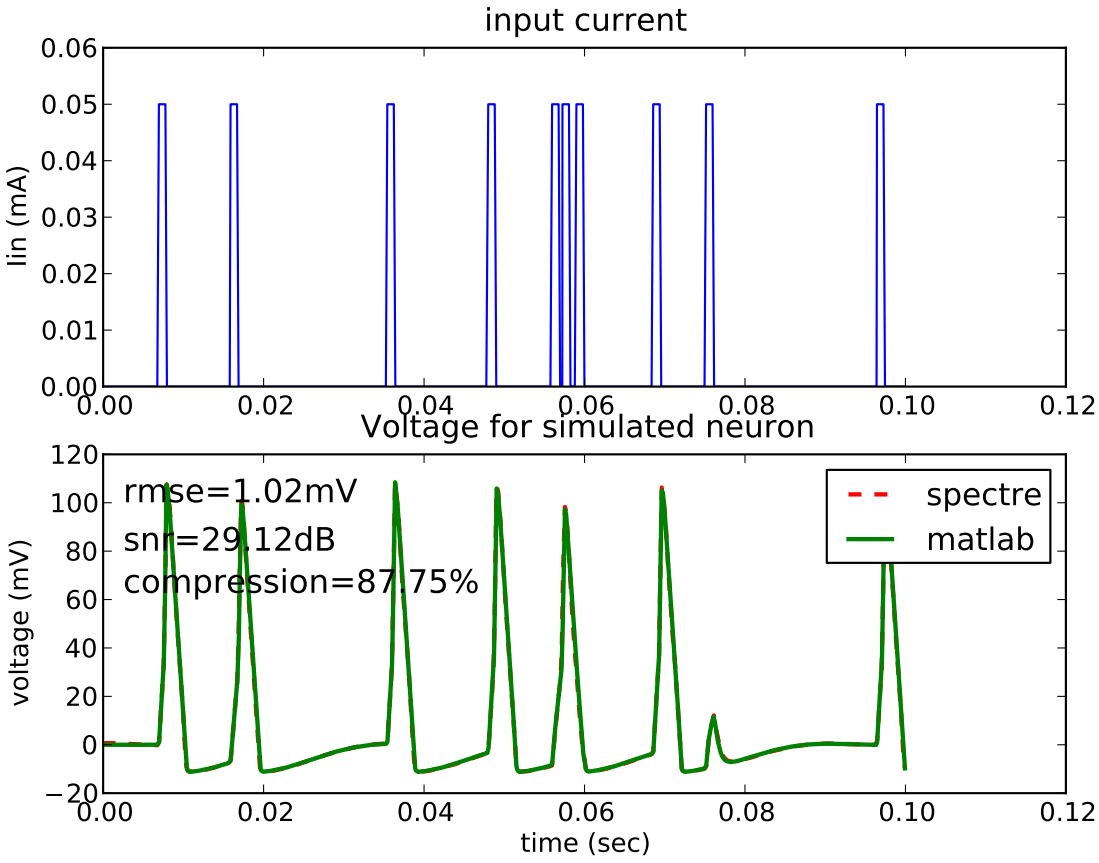
Αντίστοιχη διαδικασία ακολουθούμε και για την υλοποίηση των 4 διασυνδεδεμένων νευρώνων. Αρχικά δείχνουμε την εγκυρότητα της υλοποίησης για ένα δίκτυο 2 επί 2 στο Σχήμα 9.

Επίσης στο Σχήμα 10, απεικονίζεται πως ο συνδυασμός μεγάλης διάρκειας και μικρής μέσης διάρκειας μεταξύ παλμών ρεύματος, μπορεί να οδηγήσει σε σφάλματα στην Spectre υλοποίηση.

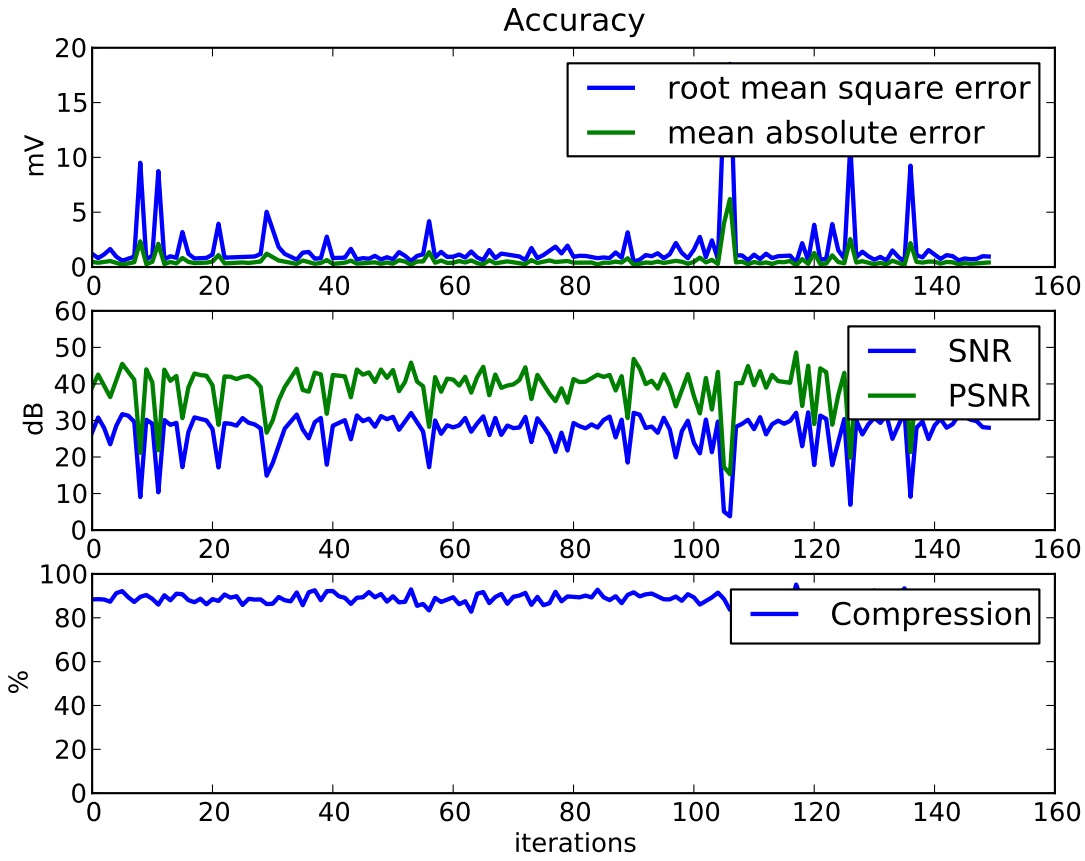
Τέλος, μπορούμε να δούμε πως η μεταβολή ορισμένων παραμέτρων όπως το maxstep μπορεί να συντελέσει στην αυξομείωση της ακρίβειας προσομοίωσης, καθιστώντας τελικώς τον προσομοιωτή Spectre ένα πολύ ευέλικτο εργαλείο για την προσομοίωση νευρώνων. Στο Σχήμα 11, βλέπουμε την επίδραση του maxstep στο σηματοθορυβικό λόγο για κάθε έναν από τους νευρώνες του δικτύου.



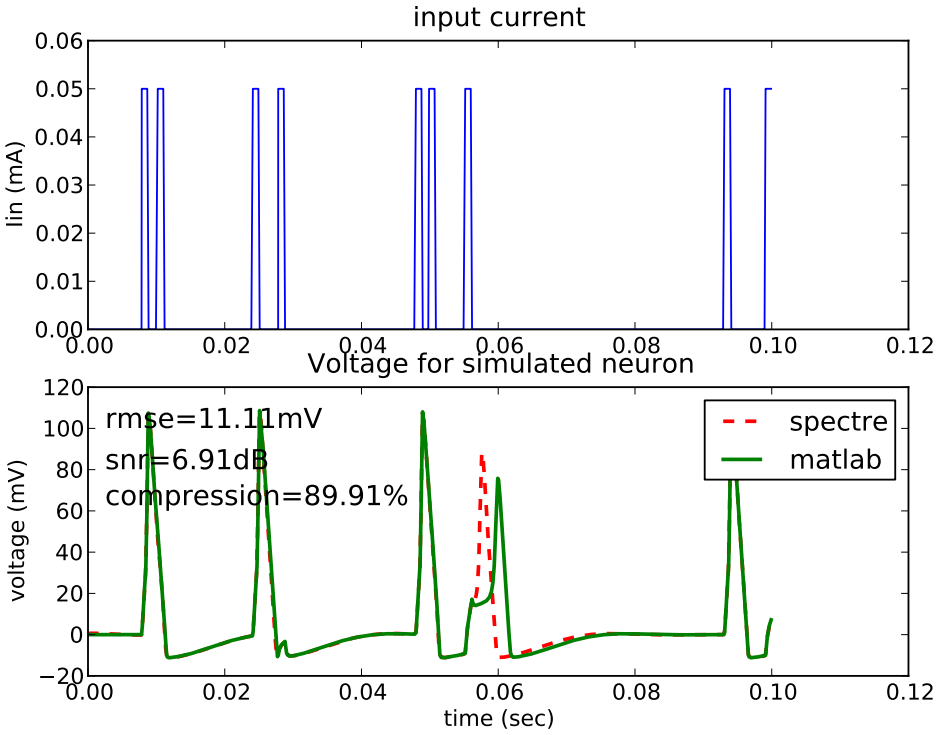
Σχήμα 4: Μια παλινδρόμηση της τάσης μεμβράνης συγκρίνοντας τα δύο εργαλεία



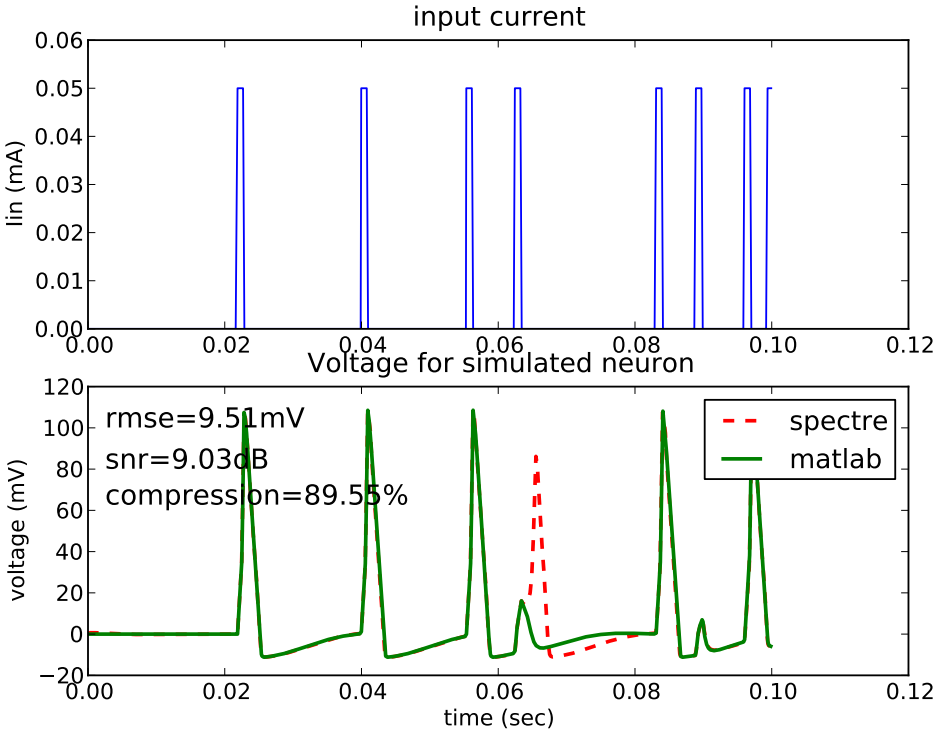
Σχήμα 5: Πολλαπλές ταλαντώσεις για τυχαία είσοδο



Σχήμα 6: Ακρίβεια και συμπύκνωση χρονικών σημείων για 150 επαναλήψεις

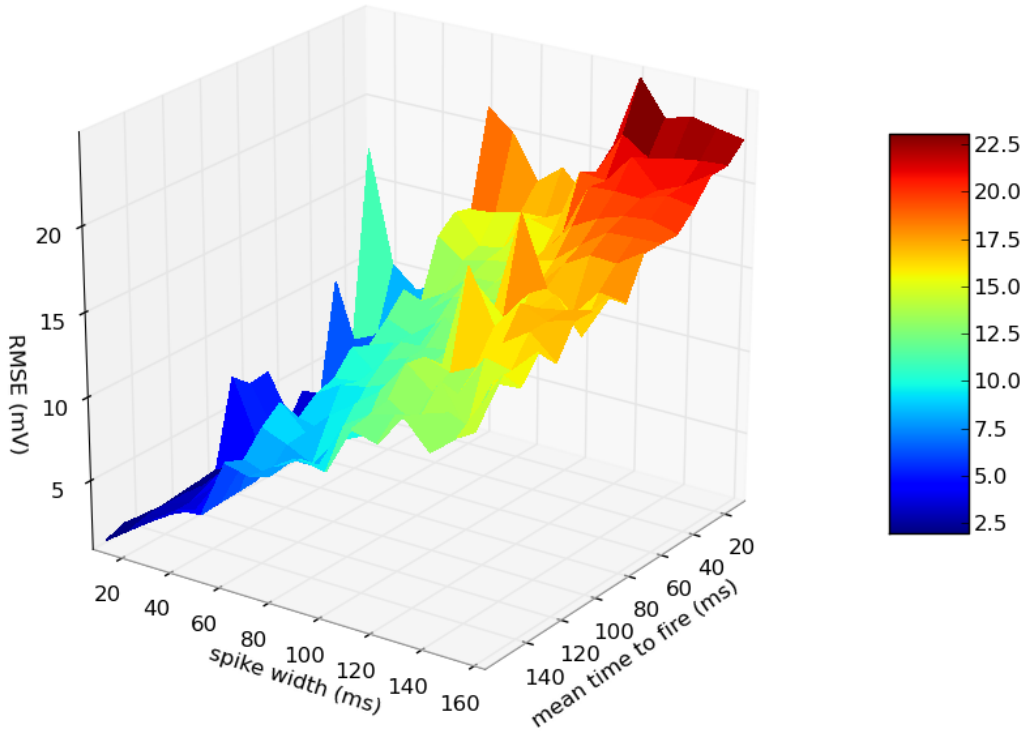


(α′) Διαφορά φάσης

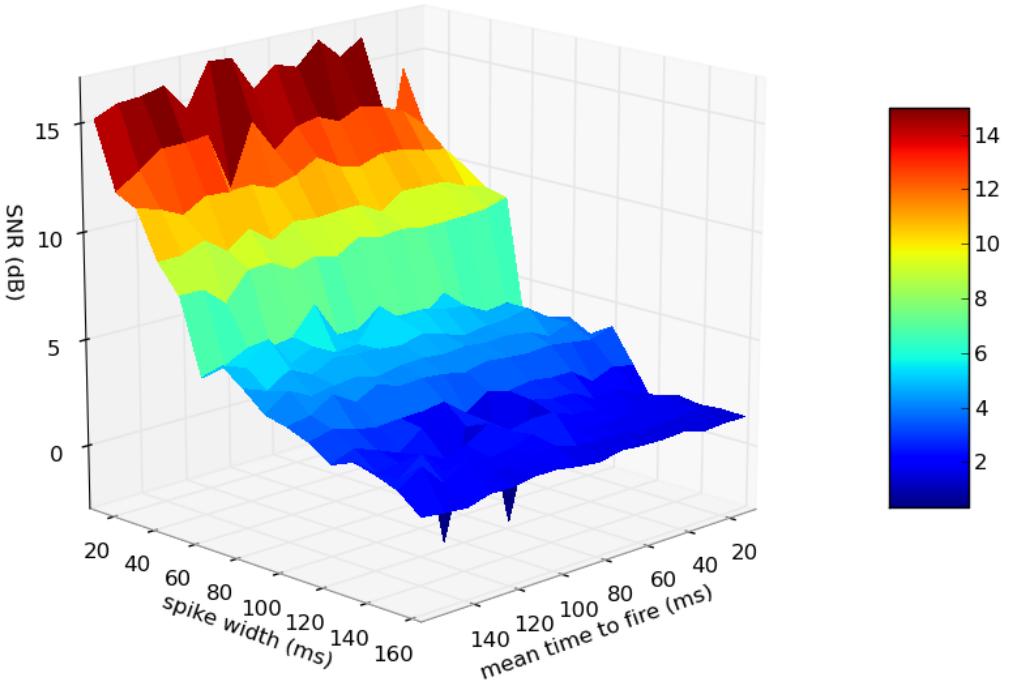


(β′) Ανακριβής ενεργοποίηση

Σχήμα 7: Περιπτώσεις που οδηγούν σε χειροτέρευση του σηματοθορυβικού λόγου

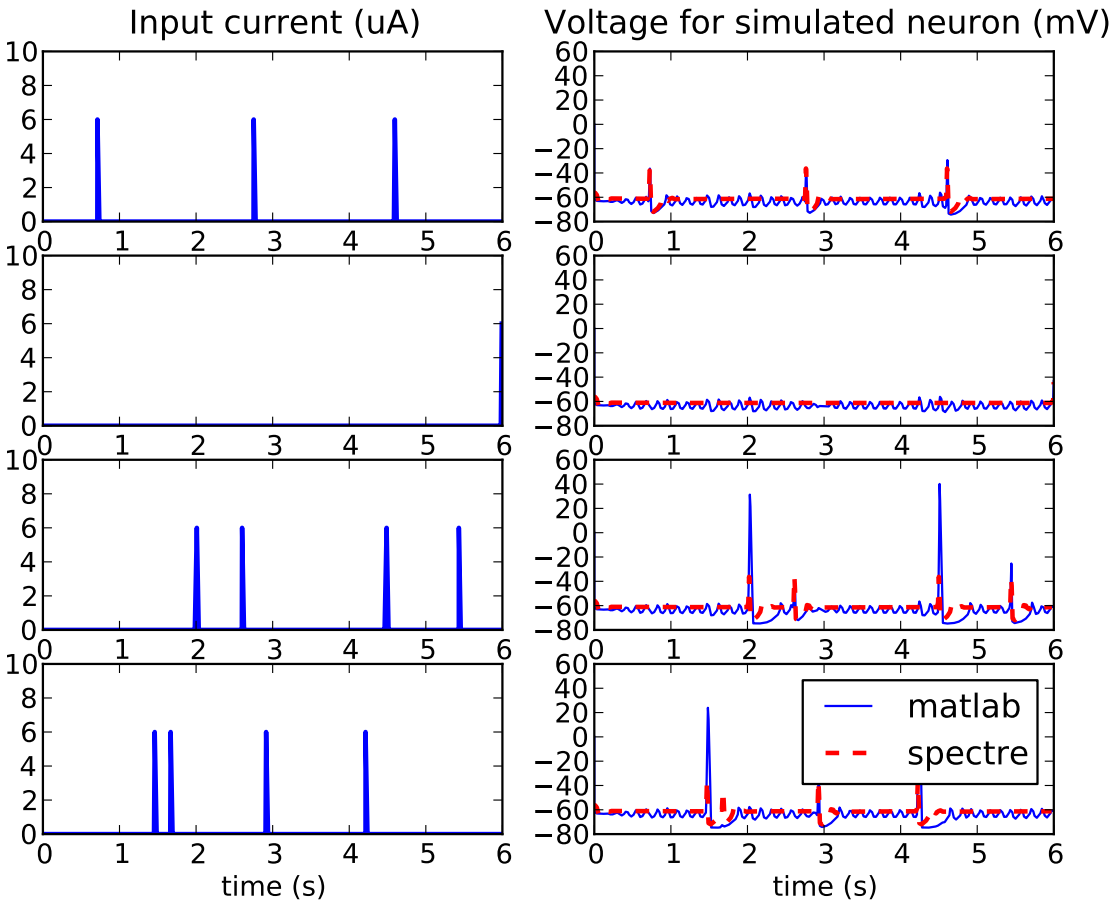


(α΄) ρίζα του μέσου τετραγωνικού σφάλματος

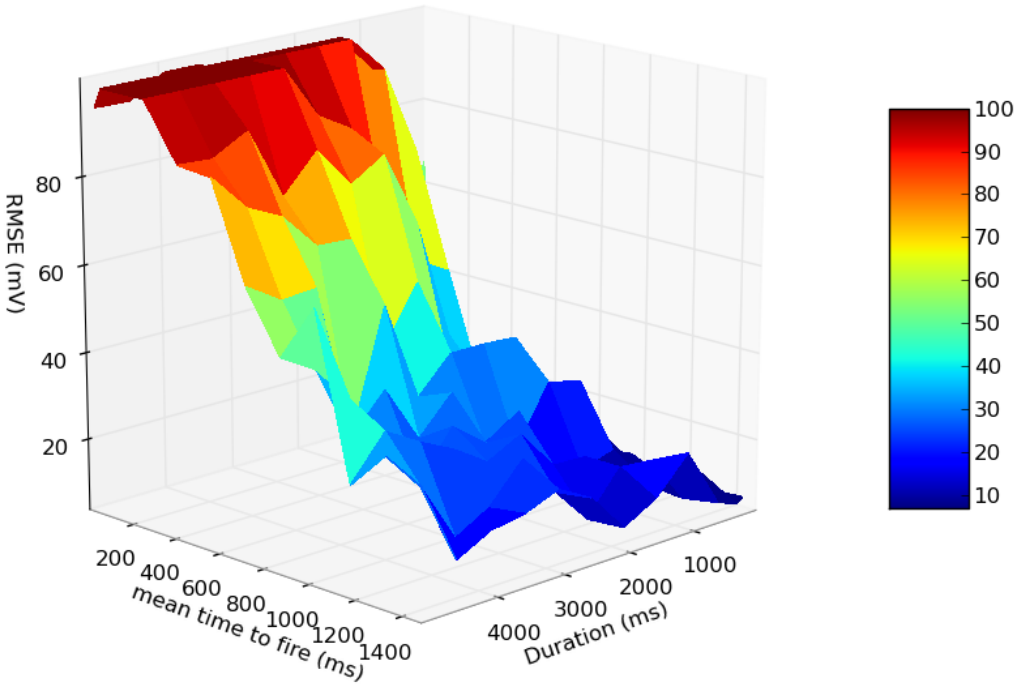


(β΄) σηματοθορυβικός λόγος

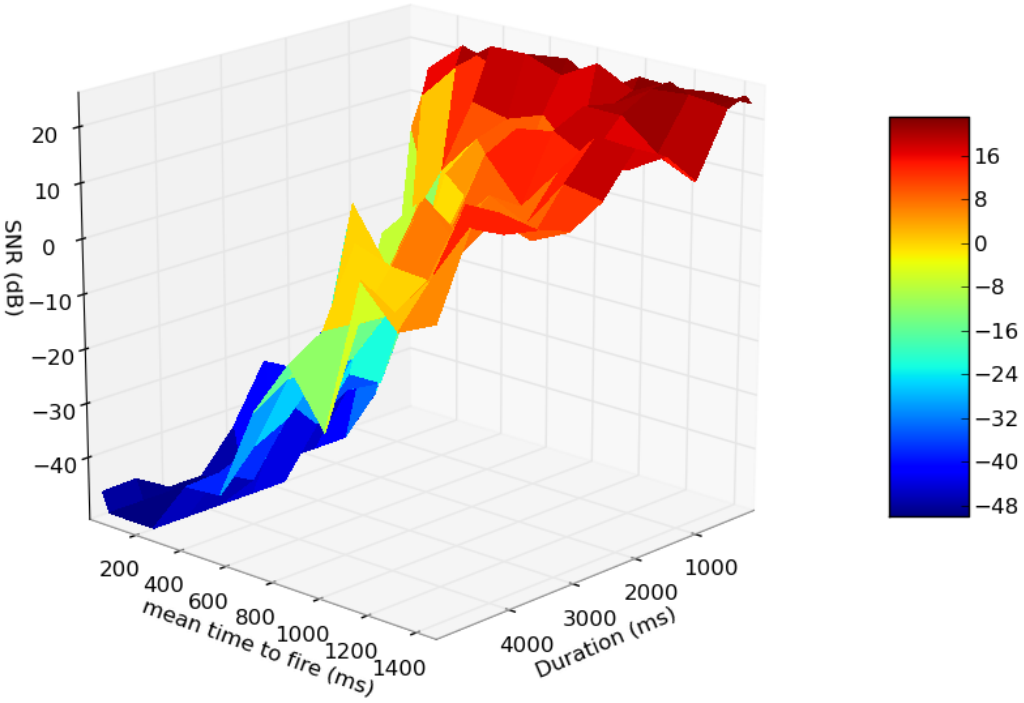
Σχήμα 8: συμπεριφορά της ρίζας του μέσου τετραγωνικού σφάλματος και του σηματοθορυβικού λόγου σε σχέση με το εύρος παλμού ρεύματος και τη μέση διάρκεια ανάμεσα στην ενεργοποίηση κάθε παλμού



Σχήμα 9: Τάση εξόδου για κάθε έναν από τους τέσσερεις νευρώνες

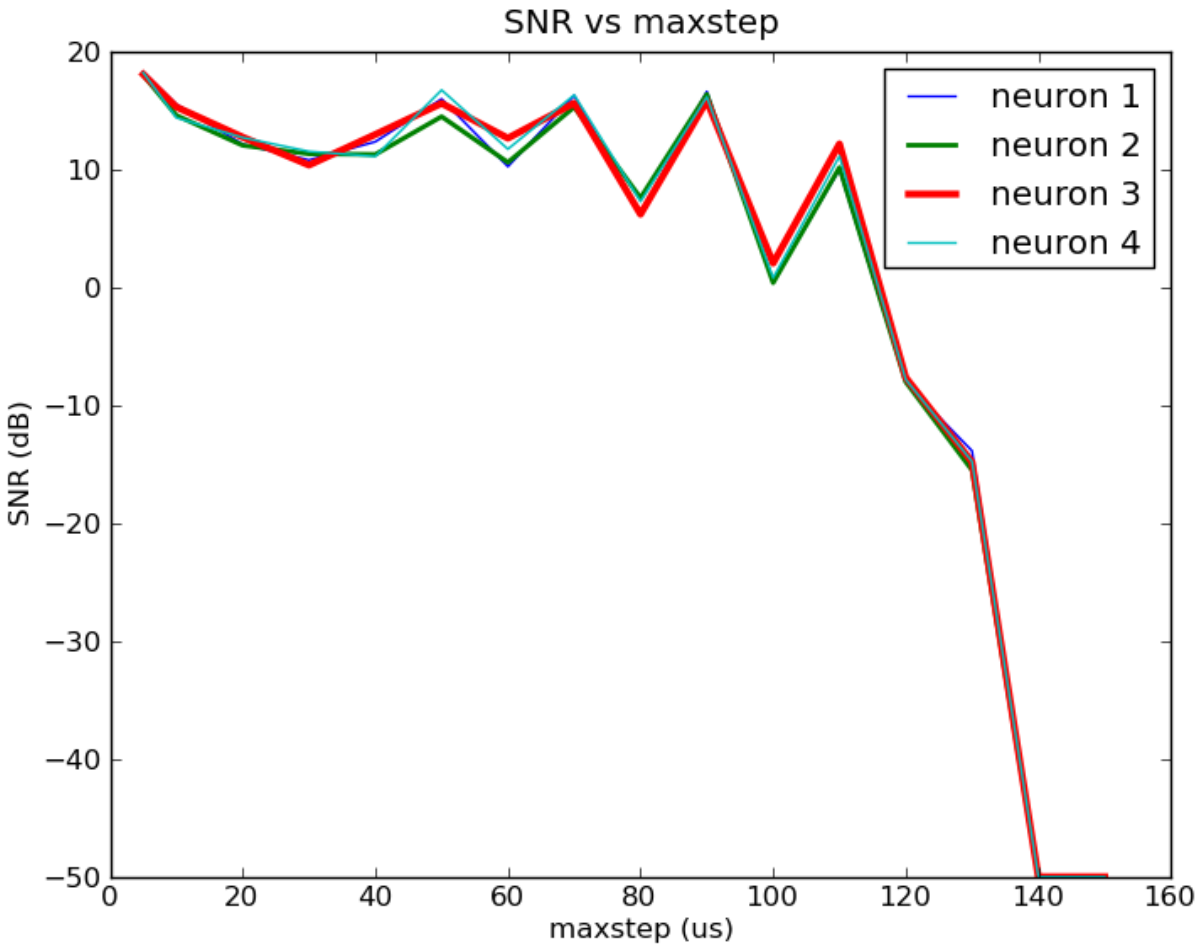


(α΄) Ρίζα του μέσου τετραγωνικού σφάλματος



(β΄) Σηματοθορυβικός λόγος

Σχήμα 10: Υλοποίηση πολλών νευρώνων: Συμπεριφορά της ρίζας του μέσου τετραγωνικού σφάλματος και του σηματοθορυβικού λόγου σε σχέση με τη διάρκεια της προσομοίωσης και τη μέση διάρκεια ανάμεσα στην ενεργοποίηση κάθε παλμού



Σχήμα 11: Σηματοθορυβικός λόγος καθενός από τους τέσσερεις νευρώνες σε σχέση με το μέγιστο βήμα προσομοίωσης

Abstract

The field of brain modeling is a very promising step towards the understanding of human brain functions and the discovery of new treatments for brain diseases. Appearing research challenges are being tackled by several scientific approaches including fMRI, Neuroimaging and High Performance Computing. Despite the numerous scientific advances towards the demystification of the human brain, there is always a constant need for higher optimization and standardization of the simulation procedures. Neuromorphic computing is a very interesting concept and a major step towards the better understanding of the brain, as it establishes the connection between designed neural systems and actual biological nervous systems. This analogy between electrical circuits and neural functionality is explored using traditional EDA tools that are prominent in the IC design domain.

The purpose of this thesis is to provide a detailed transient response of a inferior olivary nuclei (InfOli) model as a single neuron and as part of multi-neuron interconnection network, through a standard, multi-physics simulator for integrated circuits. The software of our choice is Spectre by Cadence. Starting from a Matlab implementation a compatible Verilog-A model is created, which is run in the Cadence Spectre simulator. The results from Spectre and the MATLAB simulator are compared to derive figures of merit as different input parameters and inter-connection schemes are explored and finally the accuracy limitations of the SPICE-like commercial tool are quantified.

Keywords: Electronic Design Automation (EDA), Inferior Olive (InfOli), Spike Width (SW), Mean time to fire (Mtf), Root Mean Squared Error (RMSE), Signal to Noise Ratio (SNR).

List of Figures

List of Tables

Chapter 1

Introduction

In order to coordinate all the functional needs of its organism, the brain has evolved into a complex combination of structures each of which has a specific role. Every structure however, consists of neural cells connected together in various ways forming neural networks. The full spectrum of neural behavior has not yet been fully documented and analysed, nor has it been explained how spectacular functions such as thought, stem from the interconnection of these tiny cells. The research challenges regarding the brain are numerous and are being tackled by a wide range of scientific approaches, while some approaches like functional Magnetic Resonance Imaging (fMRI) or neuroimaging analysis are characterized by the development and implementation of computational models in order to study fundamental brain structures and functions.

Brain modeling is also a popular application increasingly appearing in High Performance Computing (HPC) and multi-core processing systems, with modeling and visualization being the core components. Scientific observations are translated into mathematics, developing powerful algorithms to represent neuronal behavior realistically, and to make the best possible use of (super)computing power. These technologies will enable collaboration and data sharing in a large scale, with the purpose of achieving a many-scale brain reconstruction, matching clinical data to brain diseases, and developing brain-inspired computing systems [1]. Several large scale projects [2],[3],[1] focus on the understanding of the human cognition and behavior as well as the treatment of many brain related diseases. "The ultimate goals of brain simulation are to answer age-old questions about how we think, remember, learn and feel, to discover new treatments for the scourge of brain disease and to build new computer technologies that exploit what we have learned about the brain" [2].

In the meantime, the concept of neuromorphic computing is coming to light. This concept

describes the use of VLSI technology to mimic neurobiological architectures present in the nervous system. Neuromorphic analog, digital, mixed signal and software systems implement models of neural systems for perception, motor control etc. The physical architecture and principles of designed neural systems are based on those of biological nervous systems [4]. Whearas traditional Von-Neuman architecture chips make precise calculations reliably and are efficient for numerical problems, they require substancial amounts of power for more complex problems. On the other hand, neuromorphic architectures can detect and predict complex data patterns using relatively low electricity and are efficient for applications that are "rich in visual or auditory data and that require a machine to adjust its behavior as it interacts with the world" [5]. It is obvious that there is a constant pursuit towards the understanding of human brain mechanisms and functions.

This thesis attempts to amplify the analogy between electrical circuits and neural functionality by using tools for traditional Integrated Circuit design in order to perform transient simulations of networks of InfOli cells. More specifically, a commercial SPICE-like simulator is used, enhanced with a compatible InfOli model. The latter is implemented in the Verilog-A language [6], which is traditionally used to behaviourally model complex electrical/electronic phenomena [7]. The text of the current thesis is organized as follows:

Chapter 2 holds a state-of-the-art analysis regarding prior neuron models as well as an overview of EDA landscape. Related work is explored, with emphasis on alternative modeling and inter-connection schemes of the InfOli neurons. A summary of the EDA domain is provided as an attempt to find common ground between EDA and Neuron Modeling.

Chapter 3 introduces the reader to our Verilog-A implementation of a simplified single neuron InfOli model based on [8] and a more complex connection scheme of multiple neurons [9]. Characteristics of the Spectre simulator in relation to the Verilog-A model are also presented, and finally a comparison is made between the legacy MATLAB code and the Verilog-A code base with emphasis on the adaptive step characteristic of the EDA tool used.

Chapter 4 presents the verification for the single- and multi-neuron implementations. Different simulation approaches are discussed, and figures of merit that illustrate detailed com-

parison metrics for MATLAB and the Cadence Spectre simulator are presented.

This thesis concludes in Chapter 5 which summarizes results and respective observations. Remarks about the infOli implementation and the simulator are made, in addition to future work references.

Chapter 2
Related Work

2.1 Introduction

The current chapter illustrates the state of the art on neuron modeling and and gives a brief overview of the EDA landscape. Section 2.2 discusses different types of neuron models and presents prior art on neural connectivity and channel dynamics, whereas Section 2.3 focuses on the impact of the EDA technology and presents an overview of the industry's technical landscape and the juxtaposition between EDA and neuron modeling.

2.2 Neuron Models

2.2.1 Biological Aspects

Simulating biologically plausible models with the purpose of understanding the brain functionality is the main benefit of brain simulation even with the absense of in-vivo experiments. Some models can provide insight for a single cell's behaviour, while other, more complex models, can provide information on the network dynamics of bigger brain sections [3]. In-silico simulations can greatly accelerate brain experimentation and the understading of the biological mechanisms.

Two basic types of neuron modeling are the conventional and the computational models [10]:

• `Conventional models`: This type consists of the standard reductive modeling, which is usefull for describing the neural phenomena (*descriptive* models), without much regard to the

substrate, as well as providing explanations (*explanatory* models) of the physiological sources by considering the lower level mechanisms that generated them. Those mechanisms are described in terms of models that need to be expressed as a set of mathematical equations, which are capable of accurately capturing the involved phenomena.

• `Computational models`: The idea behind these models is that computations are the best way to interprit a specific brain task. The key aspects of computations are *representation*, *storage*, and *transformation* (*algorithmic manipulation*) planes, and are considered the way neural machinery can represent and store information of outside stimuli. Computational models can be further decomposed into the three planes described above, from an abstract description of the underlying task, through a more concrete description of the representations and algorithms adopted to satisfy the task, to a description of how these representations and algorithms are actually implemented.

Alternative classification of neuron modeling is possible, exposing Integrate-and-fire models, Firing-rate models and Conductance-based models [10]:

• `Integrate-and-Fire`: Describes the membrane potential of a neuron in terms of the synaptic inputs and the injected current that it receives. An action potential (spike) is generated when the membrane potential reaches a threshold, but the actual changes associated with the membrane voltage and conductances driving the action potential do not form part of the model. The synaptic inputs to the neuron are considered to be stochastic and are described as a temporally homogeneous Poisson process [11].

• `Izhikevich Models`: Biologically plausible model with extreme computational simplicity [12], that abstracts the phases of neuron spikes treating the neuron as a black box.

• `Conductance models`: Conductance-based models are based on an equivalent circuit representation of a cell membrane as first put forth by Hodgkin and Huxley [13].

The InfOli Model which is within the current thesis, falls under the *Conventional, Conductance-based model* category, as full biological information is exposed during the simulation, due to the full modelling of the underlying mechanisms. The Inferior Olive (InfOli) neural cells are of major importance for the development of motor skills, space perception and cognitive

tasks. Extensive research has been placed to reveal their functionality [9, 14, 15].

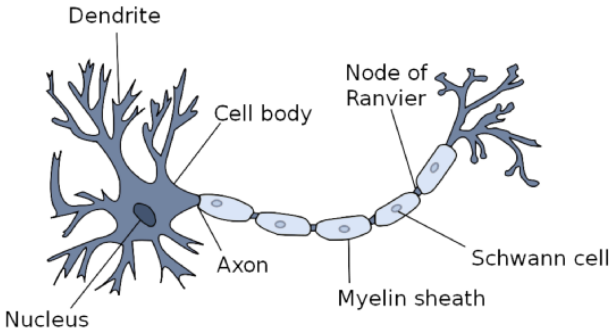


Figure 2.1: Illustration of the neuron's structure and its compartments [16]

Biological neurons (Figure 2.1) are divided into several compartments that have their own unique characteristics and a different biological role [17]:

- **Dendrites**. The dendrites of a neuron receive input current from Deep-Cerebellar-Nuclei (DCN) cells as well as other neighbouring InfOli cells and forward their output signal to the soma compartment [17]. The interfaces with neighbouring InfOli cells are called gap-junctions or electrical synapses.

- **Soma** is the neuron cell's body. Through electrochemical reactions, the cell nucleus (soma) receives a plenty of activating (stimulating) and inhibiting (diminishing) signals by synapses or dendrites, and accumulates these signals. As soon as the accumulated signal exceeds a certain value (threshold value), the cell nucleus of the neuron activates an electrical pulse which then is transmitted to nearby neurons [17].

- **Axon**.The pulse is transferred to other neurons by means of the axon. The axon is a long, slender extension of the soma and it transmits the electrochemical output signal to other brain cells, Purkinje-Cells, through the Climbing Fiber (CF).

- **Myelin sheath** that consists of **Schwann cells**, insulate the axon very well from electrical activity and **Nodes of Ranvier** are gaps which appear where one insulate cell ends and the next one begins [17].

Several prior art neuron implementations fall under the category of *Conductance-based* models. In [18], a neuromophic fully digitally programmable network of 4 biophysical Hodgkin-

Huxley neurons and conductance-based synapses is proposed (Figure 2.2) that accurately models the detailed rate-based kinetics of membrane channels in the neural and synaptic dynamics.

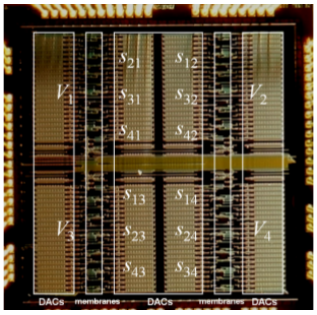


Figure 2.2: The analog VLSI chip NeuroDyn [18]

Furhter related work [19], involves an effecient digital circuit implementation that captures the quantized stochastic nature of K^+ ion channels of a Morris-Lecar (ML) *Conductance-based* neuron model.

Artificial Neural Networks are another subsection of neural analysis that tries to mimic a biological nervous system. ANN's are described by sets of adaptive weights, (i.e. numerical parameters that are tuned by a learning algorithm) and the capability of approximating non-linear functions of their inputs. Specifically perceptrons, are multilayer networks without recurrence and with fixed input and output layers [16].

Many other High Performance Computing networks have been implemented paving the way of a new era for cortical simulations. In [20], the simulations, which incorporate phenomenological spiking neurons, individual learning synapses, axonal delays, and dynamic synaptic channels, reach 1.6 billion neurons and 8.87 trillion synapses, exceeding the scale of the cat cortex.

2.3 EDA Technology

In the context of this thesis, much emphasis is given on EDA tools. The increasing amounts of CPU time and system memory as a result of the higher integration level of transistors

develops the need for exploring "High Performance computing techniques on the field of EDA applications, such as transient IC simulations" [21].

The annual data revenues (5-year stock prices) of three major NASDAQ listed EDA companies [22, 23, 24] justify the impact of this booming industry on creating modeling software tools all the way up from transistors to logic gates, IP blocks and processors. With a focus on developing new technologies from a customer's perspective and by adjusting execution and service delivery on customers' needs, EDA software has experienced an evolving pricing equation that reflects the new means of delivering software functionality. Moving from the established practice of selling perpetual licenses and time-based licesences for packaged software, to newer approaches that include software-as-a-service and commercial open source (COS), new pricing structures result in the adoption of new business models, research developments and sales practices.

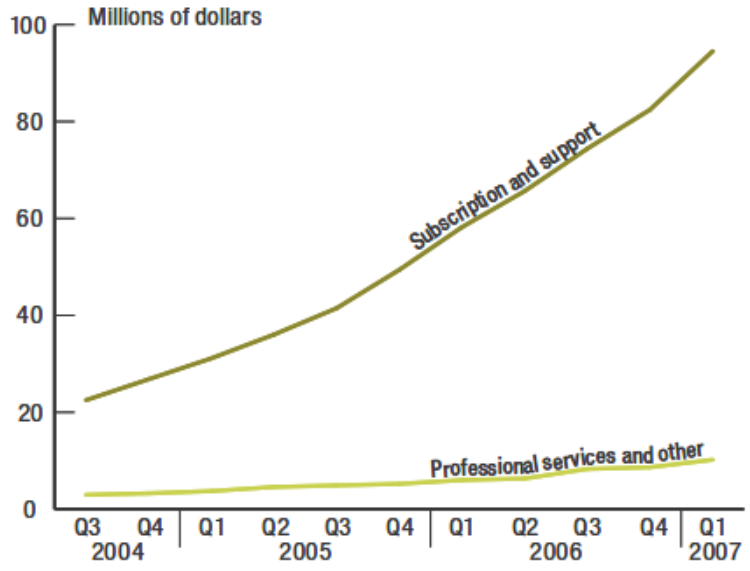


Figure 2.3: Reported Quarterly Revenue of a Representative Vendor of Software-as-a-Service [25]

Revenue reported by software-as-a-service and COS providers (Figure 2.3), is usually much less volatile quarter to quarter than the license fee revenue of vendors that use a perpetual model [25].

Several produced tools directed at the IC domain, are software tools known as SPICE applications. SPICE was developed at the Electronics Research Laboratory of the University

of California, Berkeley by Laurence Nagel [26]. It combines operating point solutions, transient analysis, and various small-signal analyses with the circuit elements and device models needed to successfully simulate many circuits and paved the way for many other circuit simulation programs.

"The increasing need for accurate and efficient circuit simulation has prompted the development of many circuit simulation programs as well as the advancement of the associated numerical methods" [27]. Several SPICE-like simulators have been produced, some of which are "HSPICE" by Synopsys, "ModelSim - Nanometer IC Design" by Mentor Graphics and "Spectre" by Cadence.

2.4 EDA and neuron modeling - Motivation

There is a similar trend between brain modeling complexity and the number of transistors in processors. As indicated in Figure 2.4, although brain modeling complexity is increasing among species with larger and more complex neural networks, so does the number of transistors in modern processors, a fact that makes the solutions to brain modeling problems more attainable.

In an attempt to find common ground between EDA and neuron modeling, we stumble upon the concept of modeling abstraction [29]. In traditional EDA, the Y-chart (Figure 2.5) is often introduced and it features the available abstraction levels of integrated circuit modeling.

Starting off at the bottom of the structural representation, the **technology CAD** level is where the modeling of diffusion and ion implantation happens and solid state physics and simulations reveal how the material works under different conditions. Then, in the **circuit level**, which is the main abstraction level under the scope of this thesis, Kirchoff's laws and similar rules are used to solve the transient, steady state or frequency response. Higher abstraction levels include the **register transfer level (RTL)** and the **system level** which is the highest in the abstraction scale.

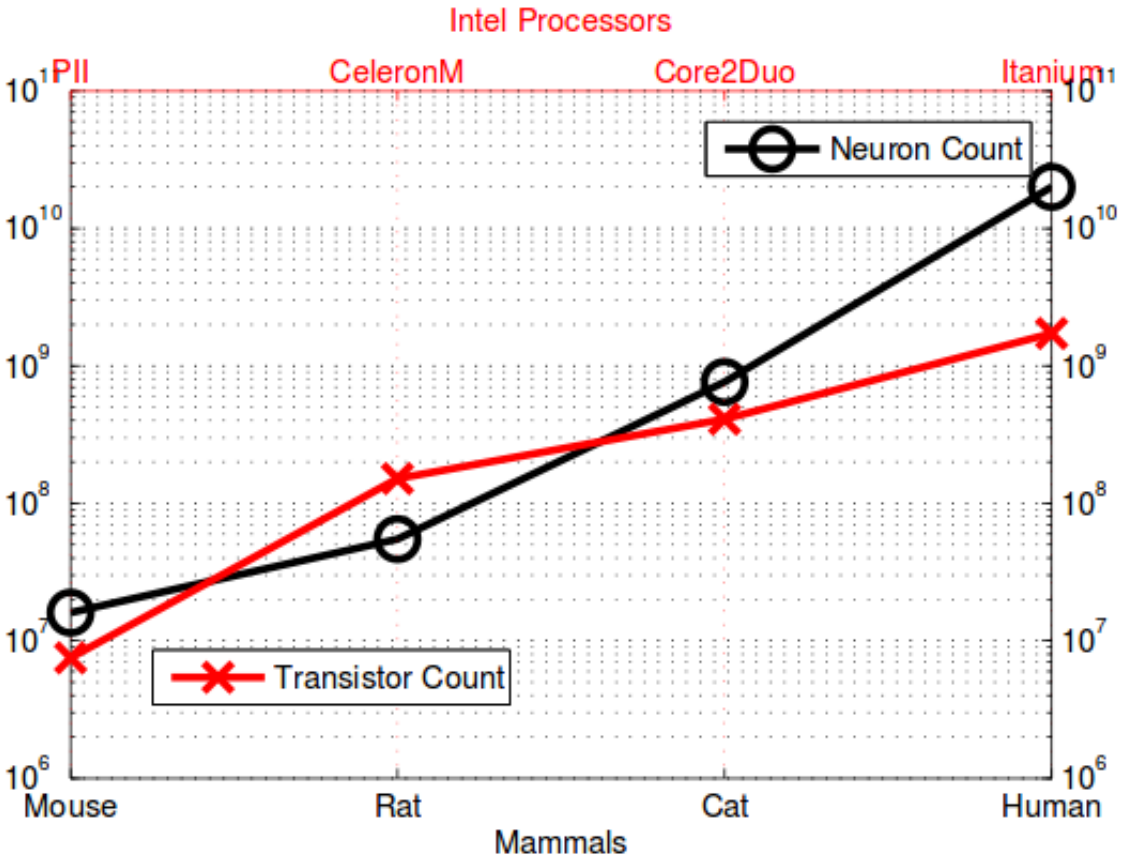


Figure 2.4: transistor vs neuron count [20, 28]

As far as the neuron modeling context is concerned, however, the concept of modeling abstractions is fragmented across coursework in a more vague manner. Some related work drifts temporarily from the single-neuron level to discuss neuronal dynamics in terms of an averaged population activity, like for example in the cortical tissue, which contains about 10^5 neurons. This spatial abstraction introduces concepts such as the average firing rate $r_i(t)$,$r_e(t)$ across excitatory-inhibitory populations and over "spikes" [31]. The average activity of neurons describes the dynamics of the network in terms of these averaged quantities. A similar concept of the average coupling strenght of two neurons being seperated by a distance, appears on [17].

There are many promising large-scale parallel applications focusing on neuron modeling [20]. However, the neuron modeling domain as well as other other areas related to the design and implementation of neuromorphic systems lack the levels of standardization and optimazation that EDA community provides. The motivation for this thesis lies in the prospect of EDA tools being used as a standard system for simulating asynchronous neural networks. While

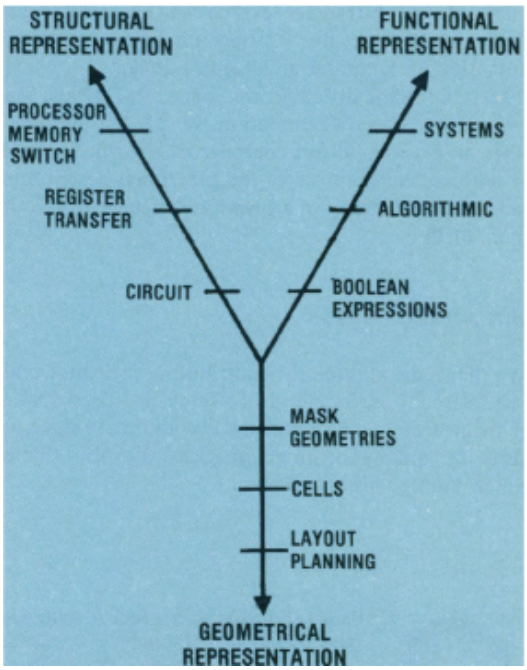


Figure 2.5: Integrated circuit abstractions for modeling and implementation [30]

modeling abstraction is a concept established in EDA for over 30 years, it has yet to be systematically introduced in neuron modeling as well.

The main focus of this thesis is the lower abstraction level, where mathematical equations describe the neuron's mechanisms and the involved phenomena are simulated in order to capture a detailed transient response of neuron cells. The SPICE-like simulator will simulate a compact Hodgkin-Huxley neuron model [8] and by exploiting the adaptive step sizing, it will produce a computationally efficient waveform representation of neuron spikes.

The libraries created for neuron timing can later be used in a higher, transaction-level modeling abstraction where neurons are perceived as finite state machines with a known temporal response. The modeling phase will be accelerated with event-driven simulations. Similar event driven simulations for neurons already exist under the scope of the Neural Simulation Tool (NEST) and the combination of MPI and openMP [32]. NEST is best suited for models that focus on the dynamics, size, and structure of neural systems rather than on the detailed morphological and biophysical properties of individual neurons [33].

Fully synthesized asynchronous implementations of digital neurons that mimic the event-driven nature of biological nervous system are already out to practise [34], establishing a

link between neuron networks and asychronous logic design. Such prior art is a major step towards the one-to-one correspondence with EDA design and the ultimate purpose of benefiting from the optimization and standardization that EDA offers.

Chapter 3

Spectre Implementation

3.1 Introduction

The purpose of this chapter is to describe the simulation tool where the inferior olive neuron model has been implemented through the Verilog-A language. A comparison is also made between MATLAB and Verilog-A code bases with an emphasis on the adaptive step sizing and finally a multi-neuron implementation is discussed with suggestions for future work.

3.2 Single Neuron Implementation

Given the Matlab function of the Hodgkin and Huxley model [8] for a single neuron, a flowchart is created describing the different stages of the process. Namely, the Hodgkin and Huxley model describes the individual neuron and how both ion conductances and voltage changes affect the action potential and the conductances of different ions. The equations used in the MATLAB code were derived straight from the original Hodgkin and Huxley paper [13]. In our case of a single neuron with one input and one output, the process is described by the flowchart in Figure 3.1.

Making the transition to the Spectre Circuit Simulator is a straightforward process that requires two steps. Firstly, a spectre-compatible InfOli model based on the given MATLAB code is implemented in the Verilog-A language, which is traditionally used to behaviorally model complex electrical/electronic phenomena [7].

Secondly, a netlist is created to be run by the Spectre Simulator as described in Figure 3.2. The circuit description file for the single neuron model is provided below :

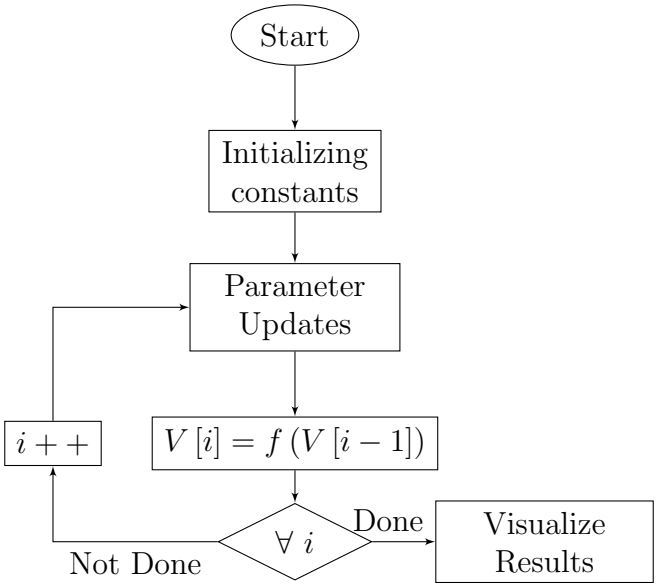


Figure 3.1: Flowchart of the Matlab function that describes the Hodgkin Huxley model

The infoli.scs file – The netlist of a single neuron

```
simulator lang=spectre
ahdl_include "./infoli.va"

singleneuron (in out) infoli gbar_K=36 gbar_Na=120 g_L=0.3 E_K=-12 E_Na=115
E_L=10.6 C=1

Iinput (in 0) vsource type=pwl file=file.dat

opt1 options saveahdlvars=all
opt2 options rawfmt=psfascii

infoli tran stop=0.1
```

The main section in the sample netlist consists of the following instance statements:

- the singleneuron component, which is an "expanded" model derived from the original infoli module and is connected to the input node `in` and the output node `out`. The singleneuron component receives the parameter values for the ion conductances, voltage equlibrium potentials and the membrane capacitance. The infoli module is the master name of the component that identifies its type and is defined in the `infoli.va` file included with the command `ahdl_include "./infoli.va"`. The `infoli.va` file should be provided at the current directory.

- the Iinput component connected to the input node `in` and the ground node 0, and is a piecewise linear voltage source derived from a raw file called `file.dat`. Even though

this component is defined as a voltage source or `vsource`, later on we will use the potential of the `in` node to access the current flowing between nodes 0 and `in`.

The code that describes the Verilog-A model is divided in three parts:

- `infoli_define.va` file, in which every variable and parameter is defined.
- `infoli_assign.va` file, in which the necessary initial values of certain variables are assigned during the initial step event of the simulation.
- `infoli_function.va` file, where the main body of the InfOli function is placed and is executed at every time step of the transient simulation.

These three files are held together in the `infoli.va` code, which should be provided in the netlist.

The `infoli.va` code used is shown below:

infoli.va code – The Verilog-A description for an InfOli model

```
‘include "disciplines.h"
‘include "constants.h"

‘define INT_NOT_GIVEN -9999999

module infoli (in,out);

//node definitions
input in;
output out;
current in;
voltage out;

‘include "infoli_define.va"

analog begin

‘include "infoli_assign.va"
‘include "infoli_function.va"

end

endmodule
```

Table 3.1: Parameters Usage

Parameters	Description
$\bar{g}_{Na}$,$\bar{g}_K$	Maximal values of electrical conductances representing Voltage-gated ion channels that depend on both voltage and time.
$\bar{g}_L$	Linear conuctance that represents leaked currents.
E_L, E_{Na}, E_K	Voltage sources whose voltages are determined by the ratio of the intra- and extracellular concentrations of the ionic species of interest.
C	membrane capacitance per unit area

Table 3.2: Variable Usage

Variables	Description
n, m, h	Dimensionless quantities between 0 and 1 that are associated with potassium channel activation, sodium channel activation, and sodium channel inactivation, respectively.
a_i,b_i	Rate constants for the i-th ion channel, which depend on voltage but not time.
I_{Na},I_K,I_L,I_{ion}	I_{ion} is the total current through the membrane calculated by equation (3.1).

As specified in the singleneuron instance statement `singleneuron (in out) infoli` in the infoli.scs netlist, `infoli` is the name of the subcircuit being called; `singleneuron` is the unique name of the subcircuit call; and *in*, *out* are the connecting nodes to the subcircuit call [35]. However, when the singleneuron instance statement is called from the netlist, the Spectre simulator substitutes these connecting node names in the singleneuron call for the connecting nodes in the infoli module definition. The main body of the Verilog-A definition is placed inside the *analog block* which is denoted by the keyword `analog`.

The `infoli_define.va` code includes all the variables and parameters that are being used. In contrast to variables, default values must be assigned to parameters during definition. In our case the default values characterised by `‘INT_NOT_GIVEN` are very large negative numbers as indicated in the `infoli.va` code, but their values are being edited in the netlist level. The different parameters and variables used in our infoli model are presented in Table 3.1 and Table 3.2.

In the `infoli_assign.va` code, the n, m, h and a_i,b_i variables are being initialized according to the Hodgin-Huxley equations [13] and the output membrane voltage is set equal to the zero baseline voltage. These initializations take place only during the first step of the simulation, that is, the DC operating point phase. This phase is called initial step *event* and is denoted by `@(initial_step)`.

The variables of the model need to be updated at every step of the transient simulation. This occurs in the `infoli_function.va` code file. The I_{ion} current is calculated based on the updated values by:

$$I_{ion} = I_{in} - I_K - I_{Na} - I_L \tag{3.1}$$

where I_{in} is the input current flowing between nodes 0 and `in`, while I_{Na}, I_K and I_L are given by the equations:

$$I_{Na} = m^3 * \bar{g}_{Na} * h * (V_{out}(t) - E_{Na}) \tag{3.2}$$

$$I_K = n^4 * \bar{g}_K * (V_{out}(t) - E_K) \tag{3.3}$$

$$I_L = \bar{g}_L * (V_{out}(t) - E_L) \tag{3.4}$$

The new value of the voltage is calculated using Euler's first order approximation in order to generate a curve that most closely approximates the time course of the voltage. The output voltage at every step of the simulation is given by:

$$V_{out}(t+1) = V_{out}(t) + \delta T * \frac{I_{ion}}{C} \tag{3.5}$$

Similarly, the channel activation constants are given in the following equations:

$$n = n + \delta T * (a_n * (1 - n) - b_n * n) \tag{3.6}$$

$$m = m + \delta T * (a_m * (1 - m) - b_m * m) \tag{3.7}$$

$$h = h + \delta T * (a_h * (1 - h) - b_h * h) \tag{3.8}$$

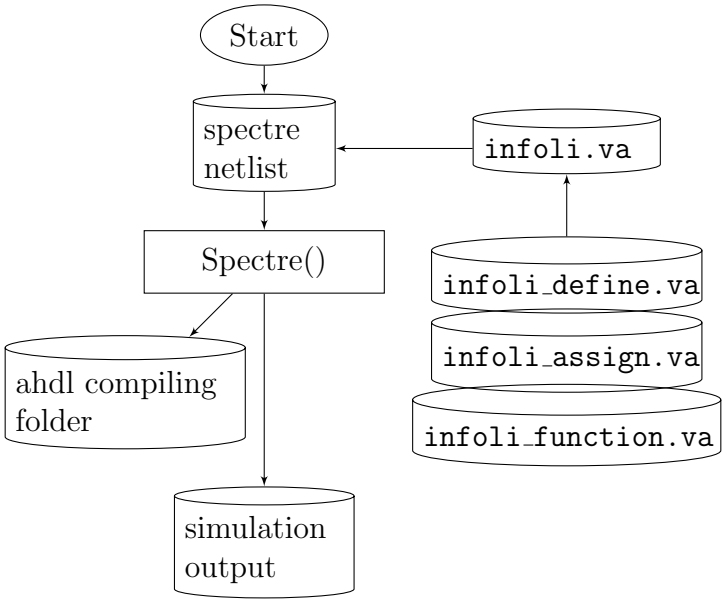


Figure 3.2: Spectre simulation flowchart. Spectre compiles the Verilog-A code for the infoli model, which included in the netlist, and produces the output and the ahdl compiling folder

The flowchart in Figure 3.2 indicates the process in which the netlist is compiled and run by the simulator and the results are produced.

Access Functions

Whereas in Matlab the input current and the output voltage of the membrane are simply defined by two vectors I and V, Verilog-A takes advantage of the access functions `I()` and `V()`, which are defined by the electrical discipline contained within the `disciplines.vams` file [36]. The arguments of the access functions refer to the internal nodes `in` and `out` of the InfOli module. In our case, `I(in)` accesses the current flowing between the `in` node and the implicit ground node. Similarly, `V(out)` accesses the potential between the `out` node and the ground node. Equation 3.5 is a branch contribution which defines the relationship between the module ports `out` and `in`.

Verilog-A Functions

Calculating the output voltage `V(out)` in equation 3.5 requires the knowledge of each step's duration, δT. As we will discuss in section 3.3, this value is not a constant number which

can be known from the beginning. For that purpose we take advantage of a specific built-in function, one of the many that Verilog-A currently supports. So, the value of δT in equation 3.5 is the unique duration of each step of the transient simulation calculated with the use of the `$abstime` built-in function [36], which in transient analysis returns the absolute simulation time in seconds. More specifically, δT is the difference in seconds between the returning values of two consecutive `$abstime` calls.

3.3 Multi neuron implementation

A multi-neuron network can be implemented based on the matlab code [9]. In comparison to the single neuron implementation, the connections between the different compartments of the infOli neuron model are now taken into consideration as in Figure 3.3.

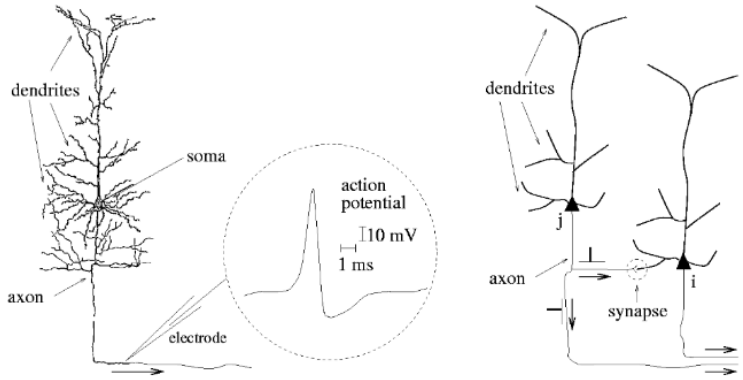


Figure 3.3: Typical structure of a neuron cell showing the communication between the axon and the dendrites of the neighbouring cell [17]

The MATLAB computational model works in a synchronous way, where in every step all neurons concurrently produce the output based on their previous state and the current inputs. The current input for each cell is the combination of the input current from the Deep Cerebellar Nuclei (DCN) and a neighbouring current computed as a function of the difference bewteen the dendritic voltages of two neighbouring cells times the conductance of their gap juction.

In the context of this thesis, 4 neurons are interconnected by twelve synapses as shown in

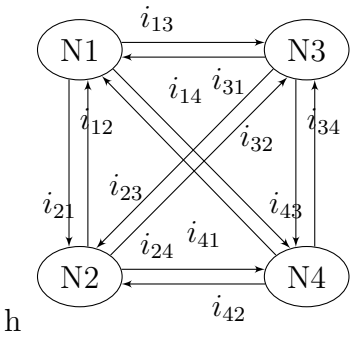


Figure 3.4: Connection scheme for 4 infOli neurons as described in our multi-neuron implementaion Verilog-A model

Figure 3.4, where the currents i_{ij} are given by the equation

$$i_{ij} = \left(V^i_{dend} - V^j_{dend}\right) * C^{ij}_{gap} \tag{3.9}$$

and C^{ij}_{gap} are the conductances of the gap junctions between each pair of neurons.

The connection scheme is implemented in Verilog-A by appropriately configuring the netlist. Four `singleneuron` components of the same infoli model are instantiated, where each component has 4 input nodes and 1 output node. When two single-neuron components connect to a node, they can either affect or be affected by this node. This connection takes the form of either the potential at the node, or the flow onto the node through the ports of the components. For example, the `singleneuron1` component affects the V^1_{dend} output node potential, which then contributes to the neighbouring input currents of the other three components as an input node potential, according to equation 3.9.

Part of the netlist instance statements for the 4-neuron interconnection scheme is provided below:

The infoli.scs file – The netlist of a 4-neuron interconnection scheme

```
simulator lang=spectre
ahdl_include "./infoli.va"

singleneuron1 (in v_dend2 v_dend3 v_dend4 v_dend1) infoli parameters
singleneuron2 (in v_dend1 v_dend3 v_dend4 v_dend2) infoli parameters
singleneuron3 (in v_dend1 v_dend2 v_dend4 v_dend3) infoli parameters
singleneuron4 (in v_dend1 v_dend2 v_dend3 v_dend4) infoli parameters

Iinput (in 0) vsource type=pwl file=file.dat
```

This connection scheme can be extended for an arbitary number of neurons with the construction of the netlist being automated by a seperate script. The size of the neuron network as well as the conductances of the gap junctions between neurons that define the interconnection scheme should be given as inputs by the user.

Verilog-A allows definitions to contain repeated elements defined using vectors of nodes. This can be used to implement devices such as the infoli module that have multiple inputs or outputs. The Verilog-A specification allows the size of vectored nodes to be specified as a parameter that can be assigned as a pre-processor constant (`‘define n 16`) or as a compile-time parameter [36].

In such cases the netlist instance statement for the i^{th} `singleneuron` component becomes:

- `singleneuron_i (in v_in[n] vout_i) infoli`

and the deifinition of the nodes as inputs in the `infoli.va` code becomes:

- `input [0:n] v_in ;`

This thesis however does not focus on the scaling of the InfOli modeling on the Cadence Spectre, rather on the accuracy that the simulator provides. As a result, the fully parametrized netlist production is left as future work.

3.4 Comparing code bases

3.4.1 Compiling

The MATLAB code is run using the Matlab Compiler, which encrypts and archives the MATLAB code (remains as .m code), and packages it in an executable wrapper. This is delivered to the end user along with the MATLAB Compiler Runtime (MCR). When the executable is run, it dearchives and decrypts the MATLAB code, and runs it against the MCR instead of MATLAB. The infOli function therefore runs exactly the same as it does within MATLAB.

On the other hand, the Spectre simulator creates an ahdl compiling folder (inv.ahdlSimDB) for the behavioral source code `infoli.va`, at the beginning of the simulaiton. Once the installed compiled interface for the verilog-A code is created, every other simulation will only require a check of whether the existing shared object for the `infoli` module is up to date.

3.4.2 Adaptive step size

The main difference between the Matlab time-based implementation and the Spectre Circuit Simulator is that the latter introduces an adaptive time step during the transient simulation. This event-based characteristic is implemented through Verilog-A with the use of the `if(analysis("tran"))` statement. More specifically, in the Matlab code, the parameter updates and the calculation of the new voltage values using Euler's first order approximation is nested within a for loop with a predefined by the user number of steps and step size which can be obtained as the difference between two simulation time points. In Verilog-A, the same process is embedded in the `if(analysis("tran"))` statement. Figure 3.5 shows how the Spectre simulator uses adaptive step to simulate the output of a typical CMOS inverter. More steps are required during rapid signal changes while a constant signal level is simulated by only a few steps. In that sense, the simulator runs and stores the minimum amount of activity that is required for accuracy to be guranteed. The output file size of

neuron simulation is kept therefore under reasonable control.

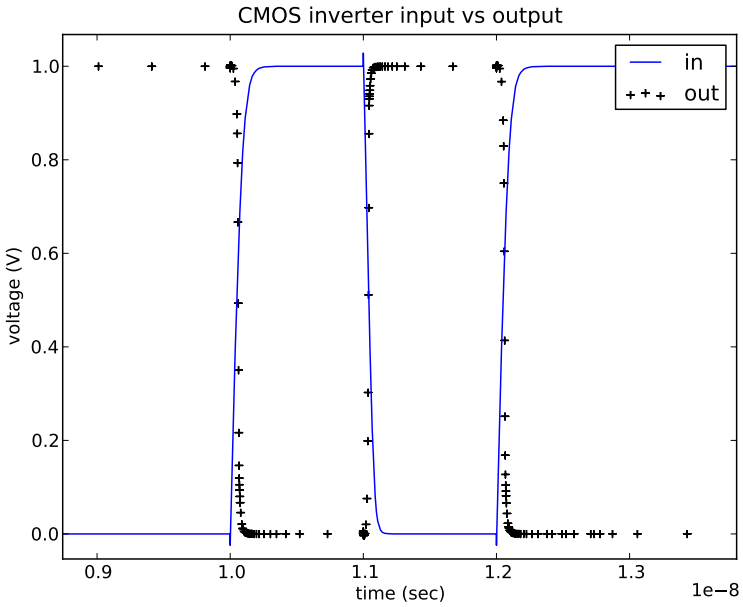


Figure 3.5: Illustrating the adaptive step characteristic of the Spectre simulator through simulating a typical CMOS Inverter

In most cases, the purpose of this adaptive stepsize control is to achieve some predetermined accuracy in the solution with minimum computational effort. The implementation of adaptive step size control requires that the stepping algorithm sends information about its performance and an estimate of its truncation error. Although the calculation of this information will add to the computational overhead, the trade-off will be received probably in smaller execution time and definately smaller output size [37].

The fundamental task of the adaptive step sizing algorithm is to take the largest step size consistent with specified accuracy constraints. Only when this is accomplished does the simulation benefit in terms of speed vs. accuracy [37]. In that sense, the Spectre simulator skips unnecessary steps where variations in the signal are smaller than a specific predetermined accuracy criterion and increases the number of steps where variations are large.

3.4.3 Spectre transient analysis parameters

Transient analysis parameters can be adjusted in several ways in terms of the error tolerance, the integration method or the maximum speed, in order to meet the accuracy needs. Usually, the performance and accuracy is improved by adjusting the `reltol` or `errpreset` parameters [35].

The `errpreset` option can set a group of parameters that control transient analysis accuracy. The spectre simulator uses three different errpreset values:

◇ `Liberal`: Suitable for digital circuits or analog circuits with short time constants
◇ `Moderate`: Suitable for approximating the accuracy of a SPICE2 style simulator
◇ `Conservative`: More appropriate for sensitive analog cicuits and high accuracy requirements

Based on the documentation [35], manual adjustments can be achieved by changing the values of specific parameters like `reltol` or `maxstep`. If, for example, more accuracy is required than that provided by `conservative` preset, the error tolerance can be tightened by setting `reltol` to a smaller value. The differences in the parameter values for the three `errpreset` settings are shown in Table 3.3.

Speed and accuracy can also be adjusted by the method parameter. The Spectre simulator uses three different integration methods: the backward-Euler method [38], the trapezoidal rule, and the second-order Gear method [39]. Several combinations are also permited between these methods [35].

Table 3.3: Differences in errpreset settings [35]

`errpreset`	liberal	moderate	conservative
`maxstep`	T/10	T/50	T/100
`reltol`	10e-03	1e-03	100e-06
`lteratio`	3.5	3.5	10
`relref`	sigglobal	sigglobal	alllocal
`method`	gear2	traponly	gear2only

3.4.4 Simulations wrappers

The whole simulation process for both single- and multi-neuron implementations is orchestrated by a Python script (version 2.7.6), as described in Figure 3.6. The input current source for both MATLAB and the Spectre simulator is derived from the a raw file that contains current spikes of 5 μA at random time points.

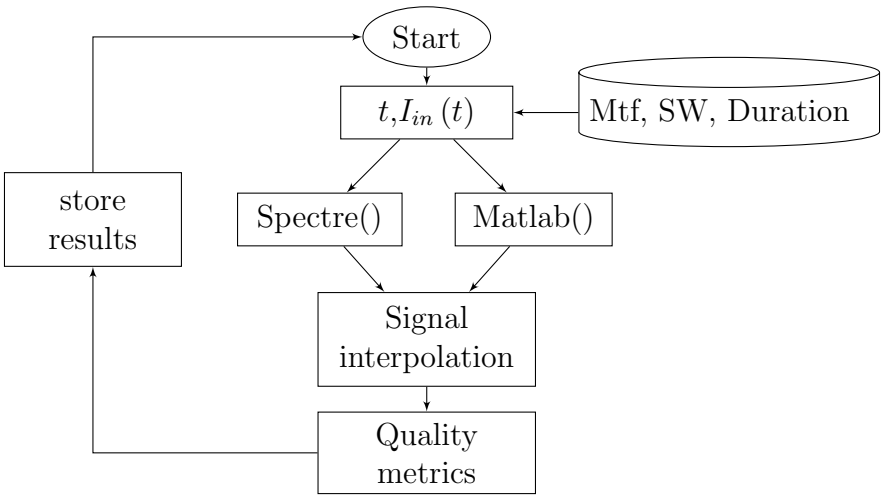


Figure 3.6: Python script flowchart

In order to create the input current file, a current input generator is used [40], that produces spikes with specific width, mean time between firing and duration, specified by the user. The InfOli neuron module function is run simultaneously both by Matlab and the Spectre simulator in parallel processes. Results are then stored in list structures and compared in terms of quality metrics.

Chapter 4
Simulation Results

4.1 Introduction

This chapter discusses the process of the single and multi neuron verification. Section 4.2, focuses on the singe neuron verification, the types of simulations and their purpose, the different settings used and the results produced by applying different input patterns. A similar discussion is made in section 4.3 about the multi neuron verification with a quick reference to future exploration.

4.2 Single-Neuron verification

4.2.1 One Oscillation - Unit step current pulse

After the Verilog-A implementation for the single-neuron model is complete, a simple simulation is run to verify the validity of the model compared to the existing Matlab impemention. The netlist parameters for the spectre simulation are shown in table 4.1.

Table 4.1: Membrane Voltage Oscillation netlist parameters

analysis type	transient
`duration`	7 ms
`input current`	5 μA pulse
`errprest`	moderate (default)

The resulting membrane potential for both Matlab and spectre is depicted in Fig 4.1. Even though normally the membrane potential rests at -65 mV and climbs up to 50mV in the

depolarization phase, for simplicity reasons we set the resting potential at 0 mV. The four regions of interest in the action potential of the cell membrane can be identified in this single oscillation [16]:

• Region I - Resting State

The output remains steady at the resting potential. Gradually, the excitatory post synaptic potential starts traveling down dendrites, summing up in the cell body (in the axon hillock).

• Region II - Depolarization Phase

When a threshold is crossed, the sodium (`Na`) channels open, the potasium (`K`) ion channels close and the membrane voltage begins to rise.

• Region III - Repolarization Phase

The sodium ion channels begin to close, the potasium ion channels, which have already began to open, dominate, and the potasium flow current out of the cell pertakes the sodium current coming into the cell. The cell membrane voltage begins to drop back down.

• Region IV - Refractory period

Excessive hyperpolarization occurs past 0mV and and the membrane voltage climbs back up to resting potential.

Right from the start, the benefits of the adaptive step in the simulation are obvious compared to Matlab's fixed step. The spectre simulator compresses time points by 92.91 % and with satisfying accuracy.

Table 4.2: Membrane Voltage Oscillation results - Moderate vs Conservative

`errpreset`	moderate	conservative
`compression`	92.91 %	53.5%
`SNR`	25.95 dB	24.34 dB
`RMSE`	2.13 mV	2.61 mV

Results from the voltage membrane oscillation simulation for both moderate and conserva-

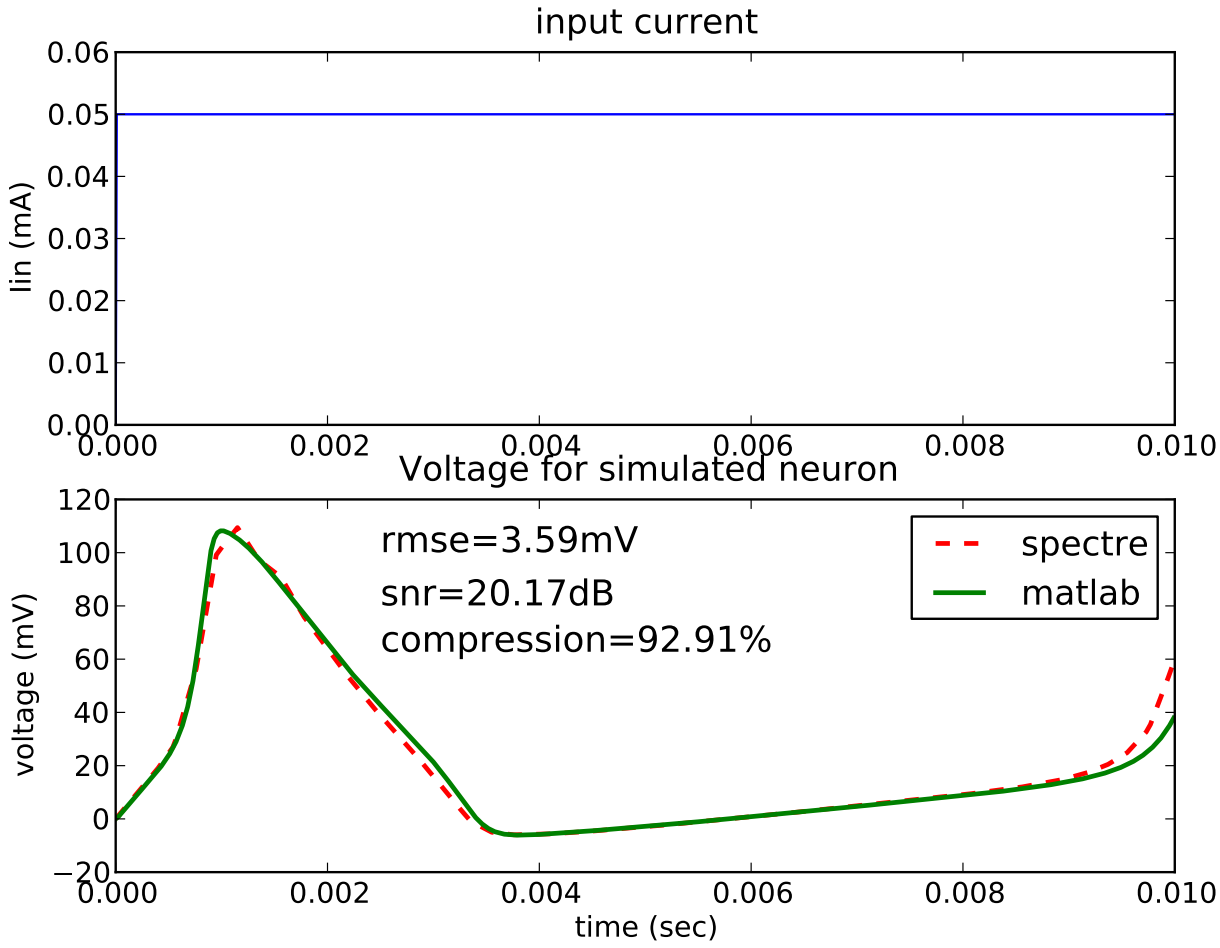


Figure 4.1: Matlab vs Spectre voltage oscillation, errpreset = moderate

tive errpreset are shown in Table 4.2. Due to the tighter `maxstep` setting, the time-point compression fell to 53.5%, in comparison to the moderate preset. For short simulation periods, the `moderate` setting leads to substancial compression and very satisfying accuracy compared to the `conservative` setting.

While experimenting with longer execution times however, the spectre simulator leads to convergence failures, due to the loose relative tolerance parameter, `reltol`, and the large `maxstep` value, which are accompanied with the moderate `errprest` parameter. The `reltol` parameter determines how well the circuit conserves charge and how accurately the Spectre simulator computes circuit dynamics and steady-state or equilibrium points, while `maxstep` is the largest time step permitted. As a result, $Tolerance_{NR}$, which is equal to $abstol + reltol * Ref$, does not properly bound the amount by which Kirchhoff's Current Law is

not satisfied as well as the allowable difference in computed values in the last two Newton-Raphson (NR) iterations of the simulation [35].

Table 4.3: Summary of the simulation parameters' usage [35, 41]

`parameter`	**Usage**
`maxstep`	Determines the simulation's maximum step.
`reltol`	Sets the maximum relative tolerance for values computed in the last two iterations. The default for reltol is 0.001.
`relref`	Determines how the relative error is treated
`lteratio`	Local truncation error. Sets the limit on numerical integration error in the simulator.
`abstol`	Sets absolute tolerance for differences in the computed values of voltages and currents in the last two iterations. These parameter values are added to the tolerances specified by reltol.
$Tolerance_{NR}$	A convergence criterion that bounds the amount by which Kirchhoff's Current Law is not satisfied as well as the allowable difference in computed values in the last two Newton-Raphson (NR) iterations of the simulation.
$Tolerance_{LTE}$	The allowable difference at any time step between the computed solution and a predicted solution derived from a polynomial extrapolation of the solutions from the previous few time steps.

In order to fix this convergence problem, the `errpreset` is set to `conservative` and consequently the `reltol` and `maxstep` parameters are tightened, creating more strict convergence criteria and increasing accuracy. Tightening `reltol`, however, also diminishes the allowable local truncation error:

$$Tolerance_{LTE} = Tolerance_{NR} * lteratio$$

.

The `lteratio` parameter is increased to compensate for the tightening of reltol, so that the

simulation time is not increased because of the decrease in the time step [35]. The usage of these parameters is summarized in Table 4.3.

4.2.2 Multiple Oscillations

The next phase of the verification is to oberve how both simulators handle the absolute refractory period of the membrane action potential, that is, the amount of time that has to go by before we can excite an identical action potential with the same spark. For that purpose the random current input generator is introduced, that produces current spikes with specific SW, Mtf and duration.

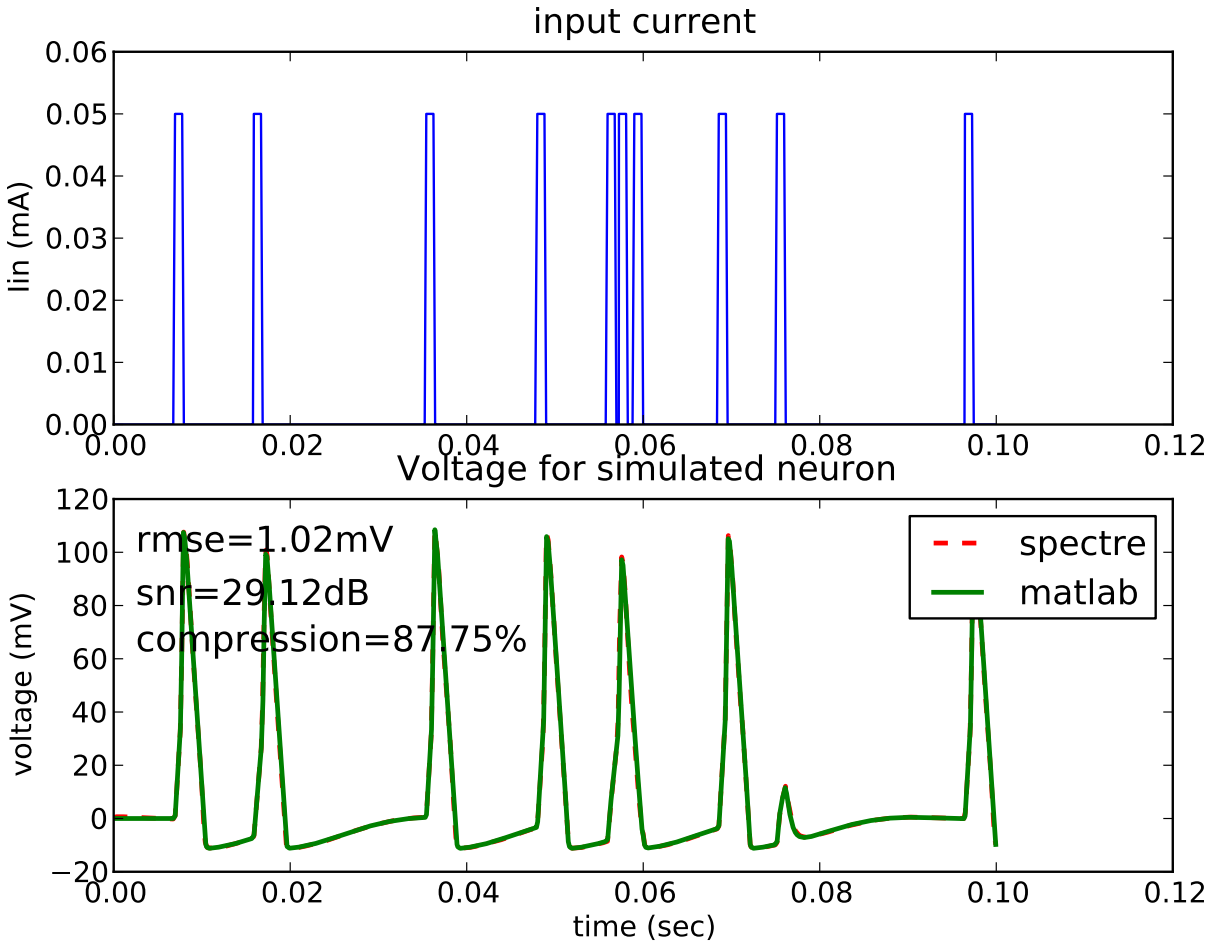


Figure 4.2: Multiple osciallations for random input, errpreset=conservative

A typical waveform is illustrated in Figure 4.2, where the accuracy obtained by the conser-

vative errpreset is obvious. As expected, two sufficiently consecutive current spikes will not cause two action potential excitations if the time between their firing is smaller than the absolute refractory period. This effect is given rise due to the fact that sodium channels enter an *inactivated* state, "in which they cannot be made to open regardless of the membrane potential" [42].

Figure 4.3, illustrates the accuracy and compression of the Spectre simulator in comparison to Matlab for 150 iterations. The simulation's execution time is 100 ms and the current spikes are of 1 ms width and a 10 ms mean time between each spike. It can be clearly seen that there are some outlier cases that produce a significant error, reducing the signal-to-noise ratio.

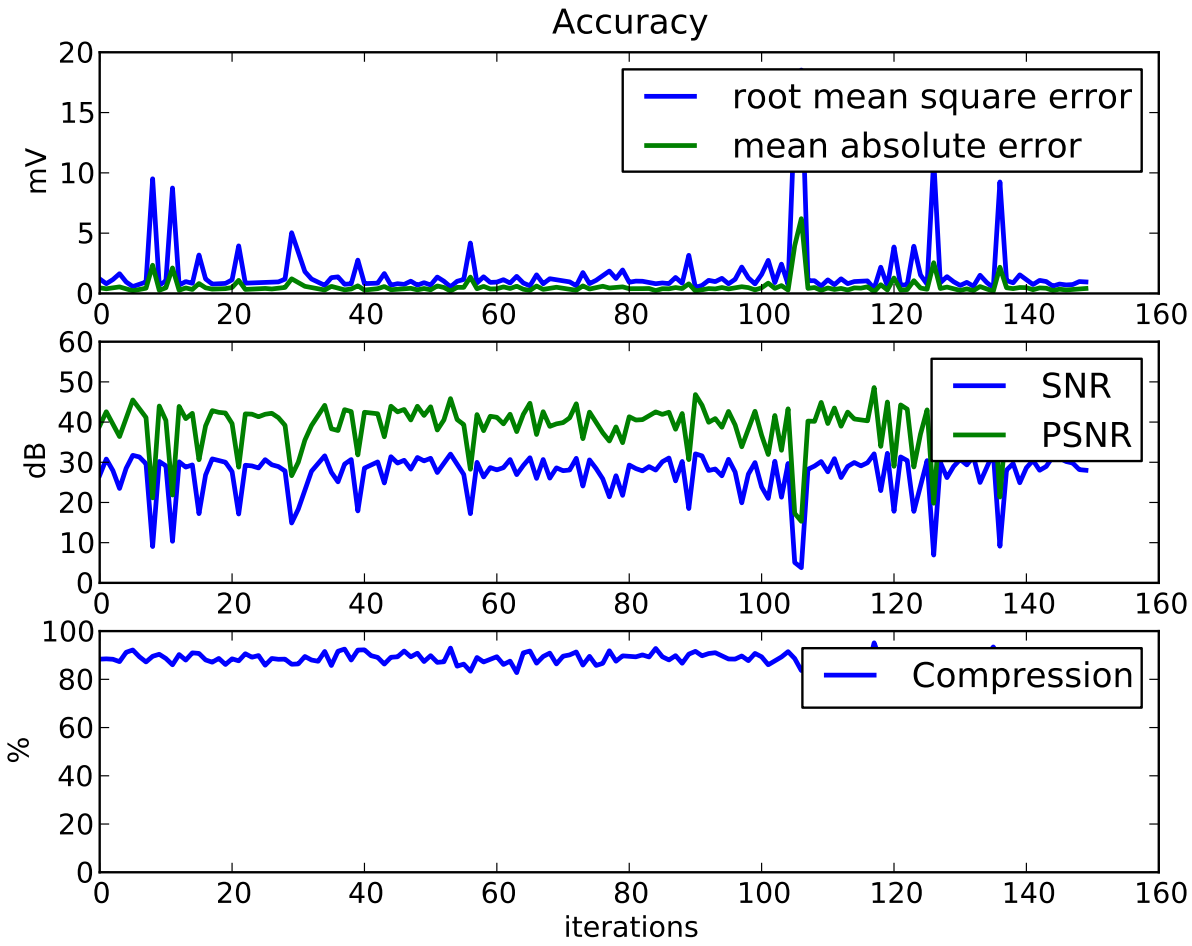


Figure 4.3: Accuracy and compression for 150 iterations

Two observed reasons that cause the error spikes or SNR drops are: (i) A slight delay of the

Matlab signal compared to the signal produced by the Spectre simulation, and (ii) an action potential excitation caused in the latter, while the Matlab signal remained below threshold. These two outlier cases are shown in Figure 4.4 (a),(b).

• **Phase shift**: The delayed matlab signal in Figure 4.5, is a product of random phase shifts of the MATLAB signal compared to the output of the Spectre simulator. For large current spike widths, the small delays caused by the phase shifts accumulate, and the delay between the signals increases causing a decrease in the SNR between both signals.

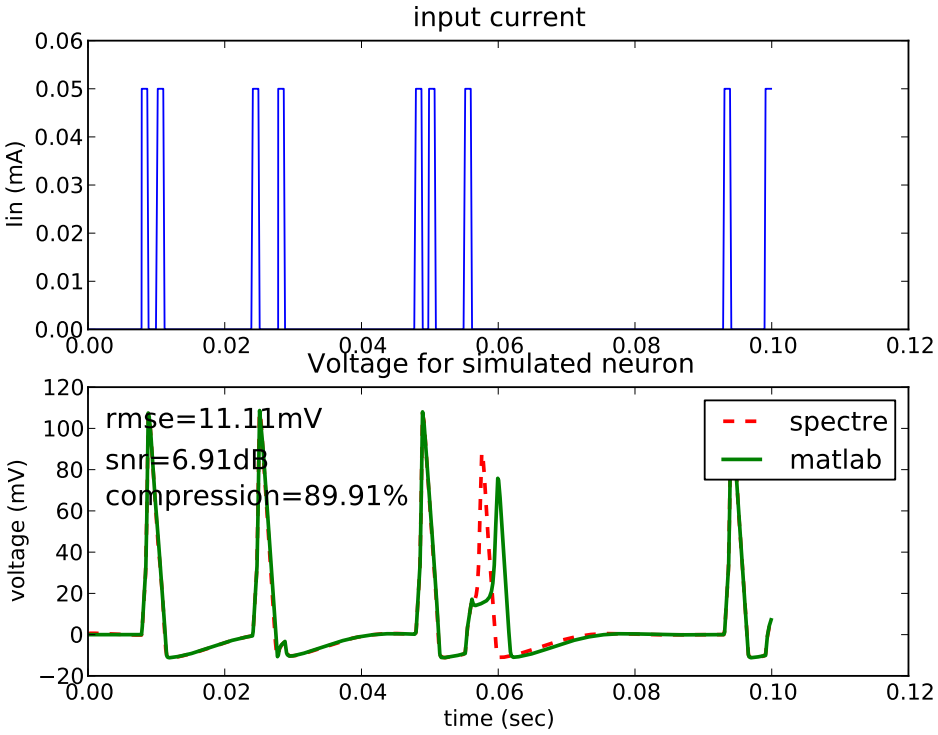
From a biological point of view, the delay is justified by the desensitization of the neuron to a constant stimulus ("accomodation" characteristic) [13], as predicted by the Hodgin-Huxley equation. What happens is that the same initial stimulus takes a longer peirod of time to raise the membrane voltage to the threshold, and as a result the action potential excitation will happen less often as a constant post synaptic current is applied after long periods. This is not to be confused, however, with the reason that causes a delay in the firing of action potentials in our simulation, which is merely a stohastic effect caused by the simulator.

• **Inaccurate excitations**: Innacurate excitations (figure 4.4 (a)) are caused when current spikes are applied in specific time points when the neuron is still in its inactivation period and no stimuli can cause a further excitation of the neuron. However each simulator interprets the inactivation period differently and therefore there are cases where a current spike at a specific time point would cause the spectre signal to rise and the matlab signal not to rise.

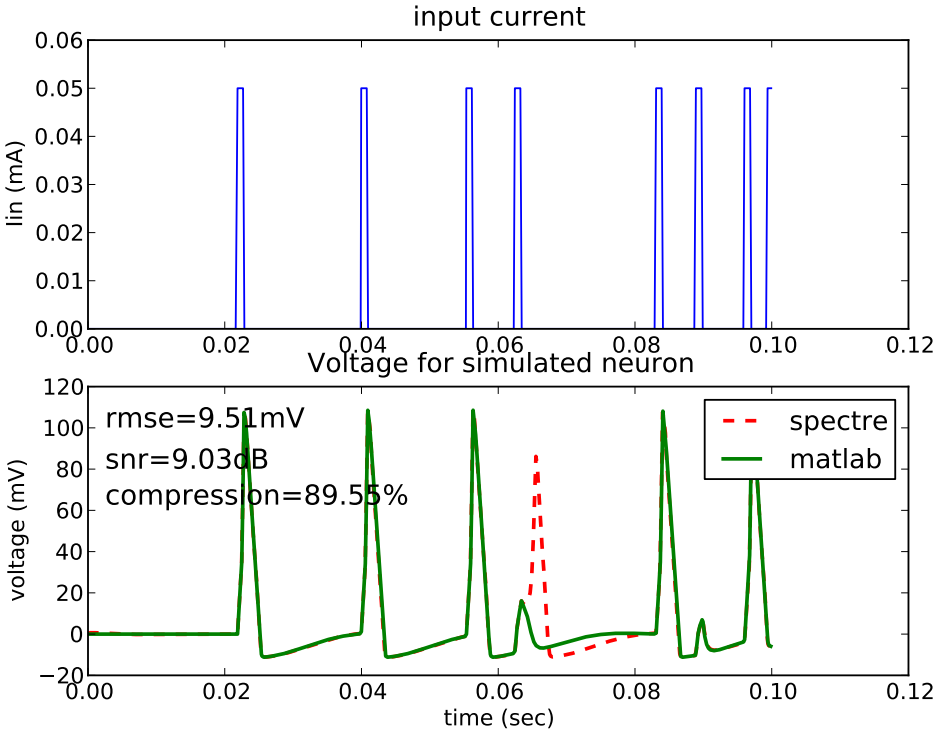
This effect is rare compared to the phase shifts, since it is improbable for the current spikes to always be fired with the same critical distance between them. On the other hand, the Matlab signal delay case is more common but produces a smaller error spike (Figure 4.3).

4.2.3 Input Parameter Sweeps

In this subsection we study the behaviour of the Spectre simulator when different input current schemes are applied by the current generator. The goal is to explain how the loss of accuracy and the convergence failures are related to the changes in the Mtf, SW and Duration. Each (Mtf,SW) and (Duration,Mtf) coordinate couple is the average of 20 Monte



(a) Phase shift



(b) Inaccurate excitation

Figure 4.4: Outlier cases that cause loss in the SNR of the Spectre implementation

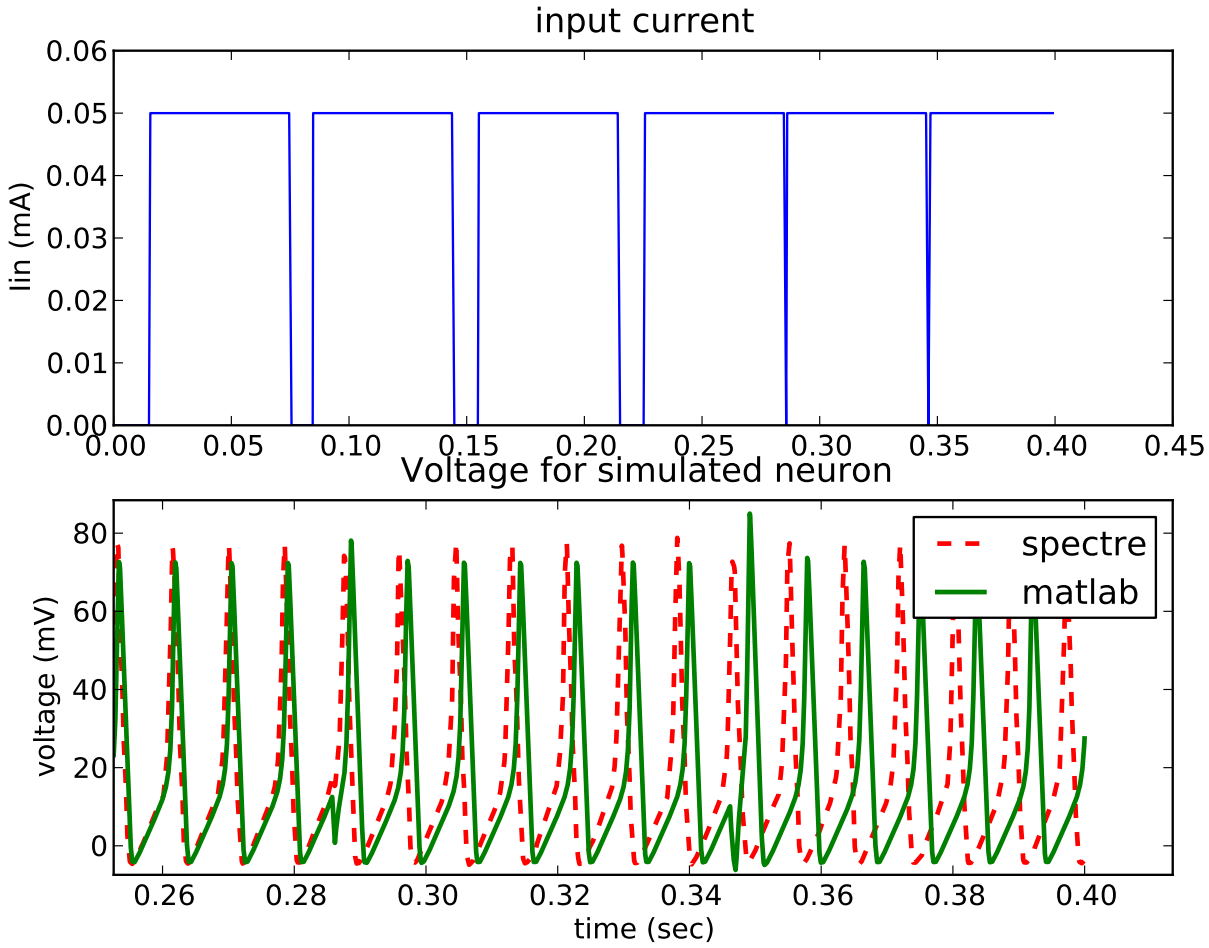
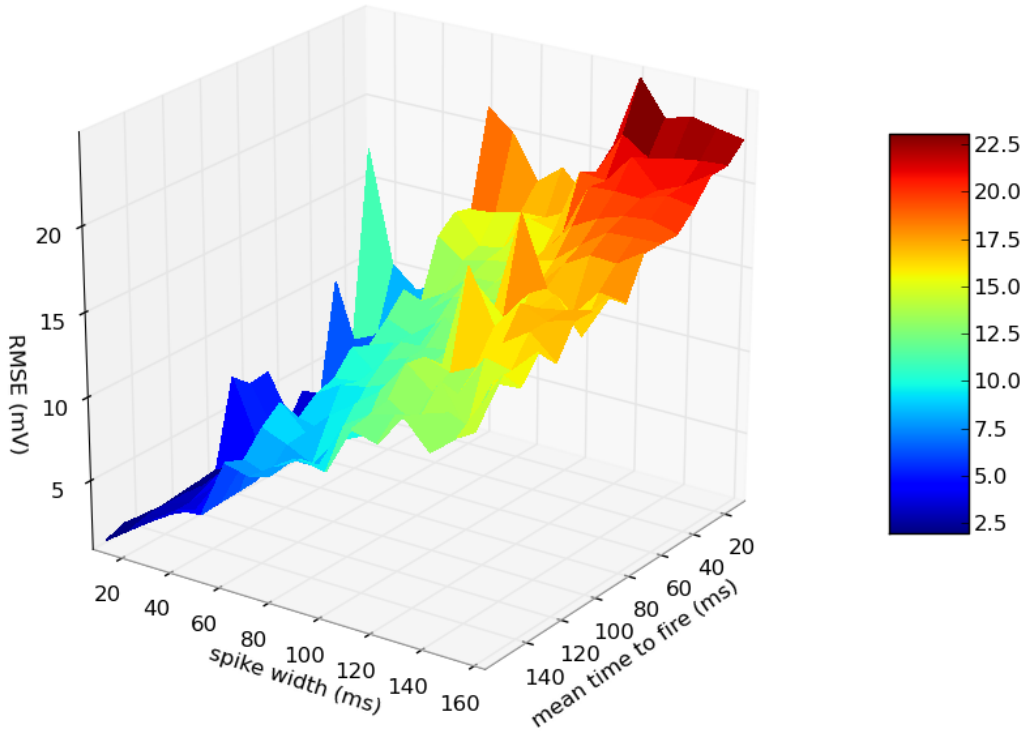


Figure 4.5: illustration of the delay caused by consecutive phase shifts of the MATLAB signal

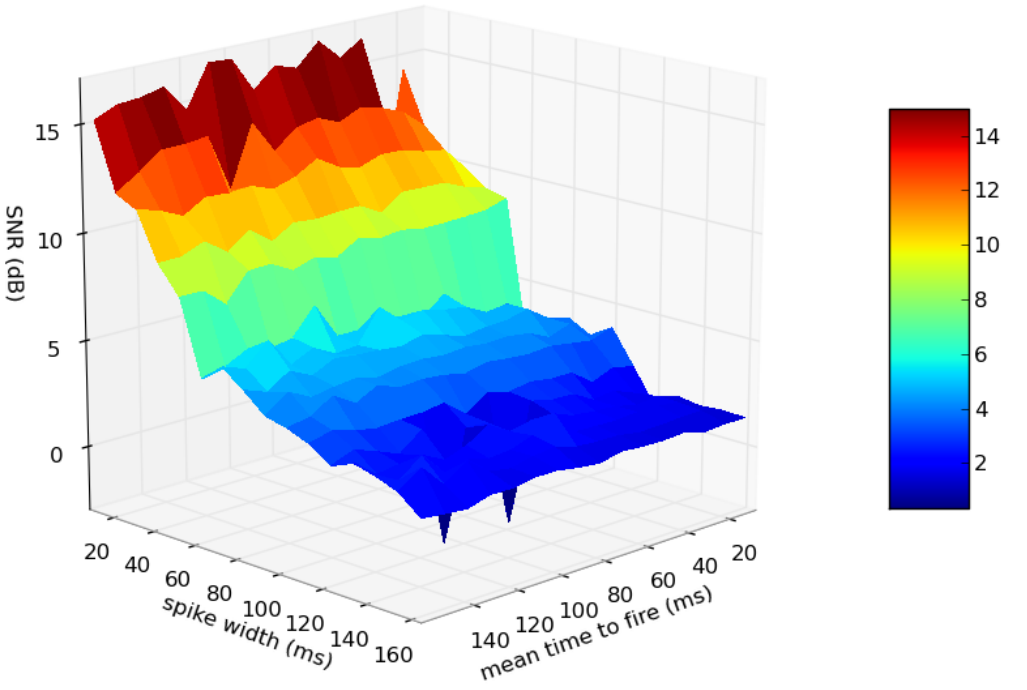
Carlo iterations.

For a constant duration of 400 ms, a 3D surface plot is created, that illustrates the SNR and RMSE between MATLAB and the Spectre simulator in terms of the Mtf and SW sweeps.

It is obvious from Figure 4.6 (a) and (b), that when the SW value increases the error becomes greater and therefore the SNR decreases, as descussed in previous section. Moreover, it is clear from Figure 4.6 (a), that a shorter mean time between current input firing produces a greater error. This is a result of inaccurate excitations occuring as described in the previous subsection. Some spikes along the surface are attributed to convergence failures, in which cases, very large error values and very small SNR values contribute to the respective average. Cases of convergence failures are very rare for a 400 ms execution time, but happen more

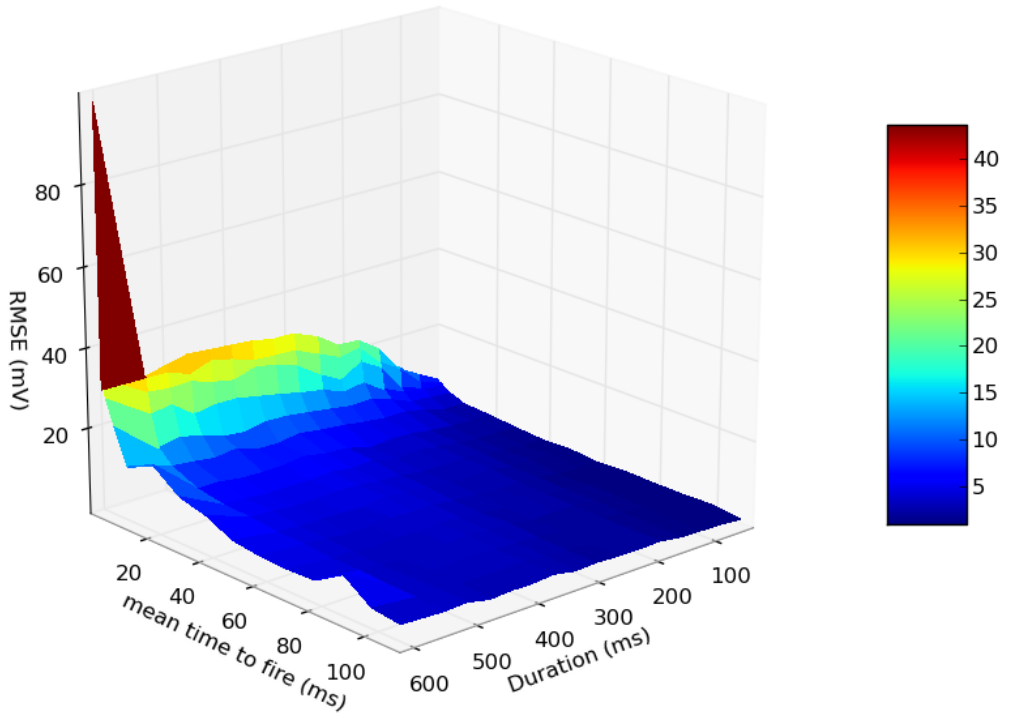


(a) RMSE (mV)

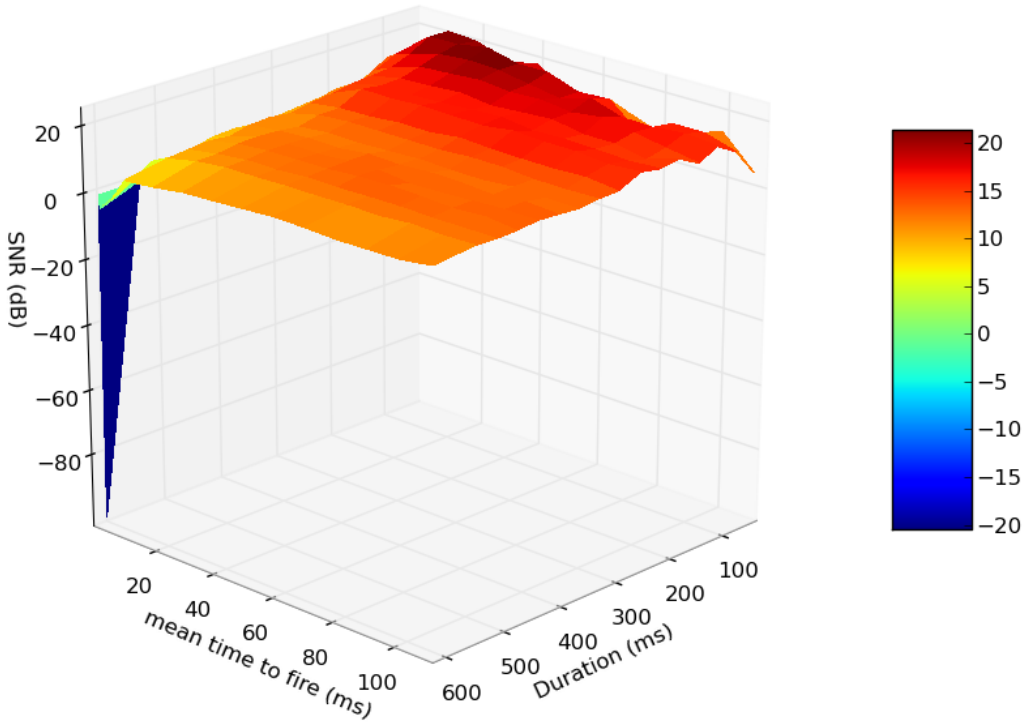


(b) SNR (dB)

Figure 4.6: Single neuron verification: Mtf and SW vs RMSE and SNR for a constant duration of 400 ms



(a) RMSE (mV)



(b) SNR (dB)

Figure 4.7: Single neuron verification: Mtf and Duration vs RMSE and SNR for a constant spike width of 20 ms

often when the execution time increases as we will see next.

The connection between total simulation time and convergence failure cases is illustraded in Figure 4.7 (a) and (b). With a constant spike width of 20 ms that produces the best SNR, convergence problems are becoming clearly evident for execution times longer than 600 ms, regardless of the Mtf.

However, a connection between the Mtf and the quality of the output signal can be observed in Figure 4.7 (a), where substancially short periods between current spikes cause the RMSE to increase and the SNR to decrease.

Even though the output signal is quite insensitive to Mtf for a smaller execution time (i.e 100 ms or less), it is clear that a combination of small Mtf and long execution time causes convergence failures.

4.3 Multi-neuron verification

Again, a simple simulation is run, illustrated in Figure 4.8 to verify the validity of the multi-neuron model compared to the existing Matlab impementation [9]. The current generator produces spikes of 25 ms width and a mean time of 1.5 s between each firing for a period of 6 s. The results of our initial verification are shown in Table 4.4.

The Spectre simulator accurately triggers the voltage spikes. It seems however, that it does not accurately interpret the subthreshold oscillations as can be seen in Figure 4.9.

Table 4.4: Membrane Voltage Oscillation results for each of the 4 neurons of the network. The current generator produces spikes of 25 ms width, mean time of 1.5 s and 6 s duration.

`settings`	maxtep=100usec, lteratio=10, relref=alllocal			
	$neuron_{11}$	$neuron_{12}$	$neuron_{21}$	$neuron_{22}$
`compression`	49.69 %			
`elapsed time`	matlab: 84.84488 s		spectre: 4.088 s	
`SNR`	24.06 dB	24.49 dB	15.97 dB	16.46 dB
`RMSE`	5.44 mV	5.14 mV	13.85 mV	13.07 mV

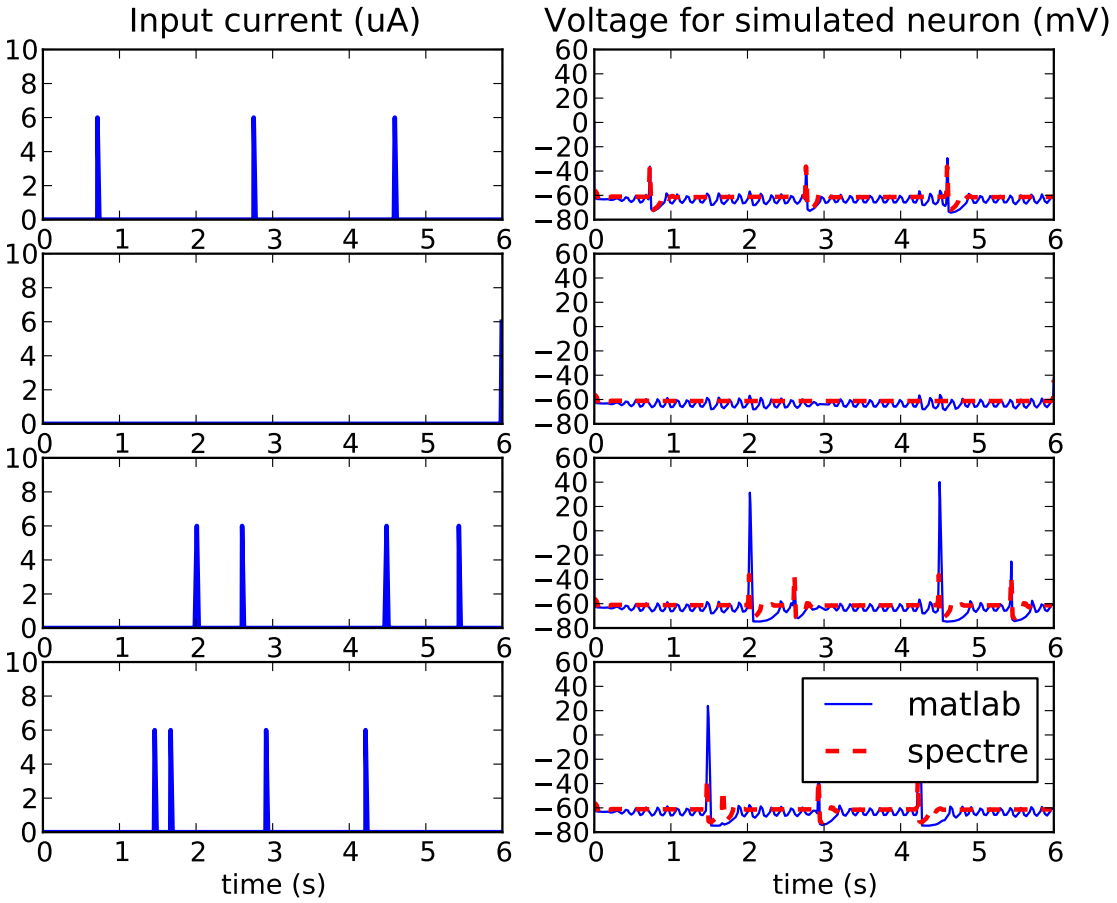


Figure 4.8: Output voltage for each simulated neuron of the 2x2 connection scheme

As in subsection 4.2.3, we explore the behaviour of the Spectre simulator when different input current schemes are applied to our multi-neuron implementation. For a constant spike width of 25 ms, we present the average SNR and RMSE of our multi-neuron implemenation relative to the changes in duration and mean time to fire. Each coordinate is the mean of 20 iterations and of all 4 neurons. Once again in Figure 4.10, we can see how the spectre simulator cannot handle the combination of small mean time to fire and large durations, whereas the SNR increases as the Mtf increases, causing a subsequent drop in the RMSE. For longer duration simulations, the matlab code [9] deviates from normal solutions, sending the output to inf and eventually leading to nan values for the RMSE and SNR. These cases are dismissed by repeating the iteration each time matlab does not lead to accepted output values.

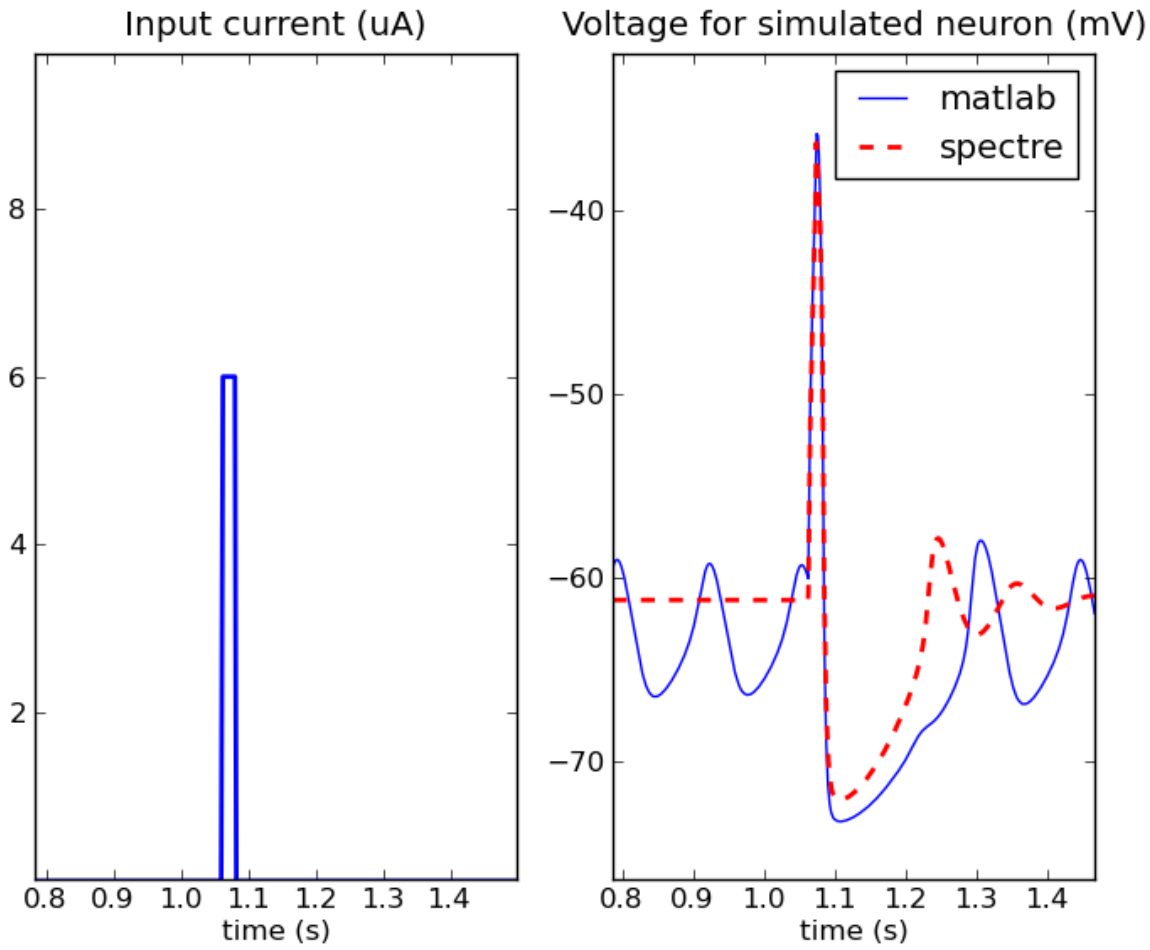
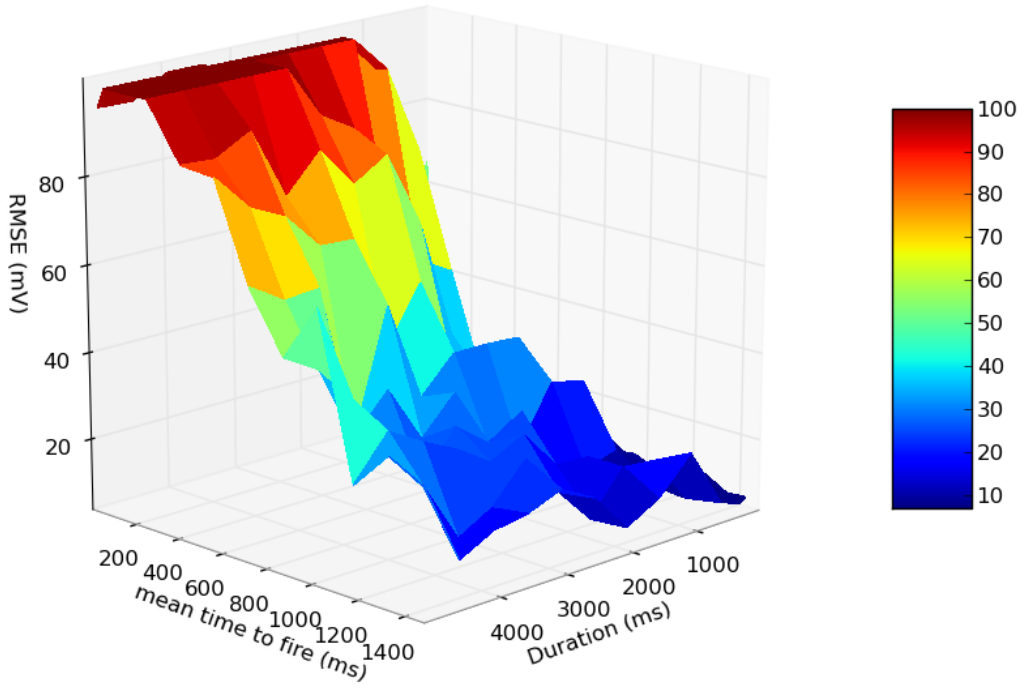


Figure 4.9: Matlab vs Spectre signal in the subthreshold oscillation level

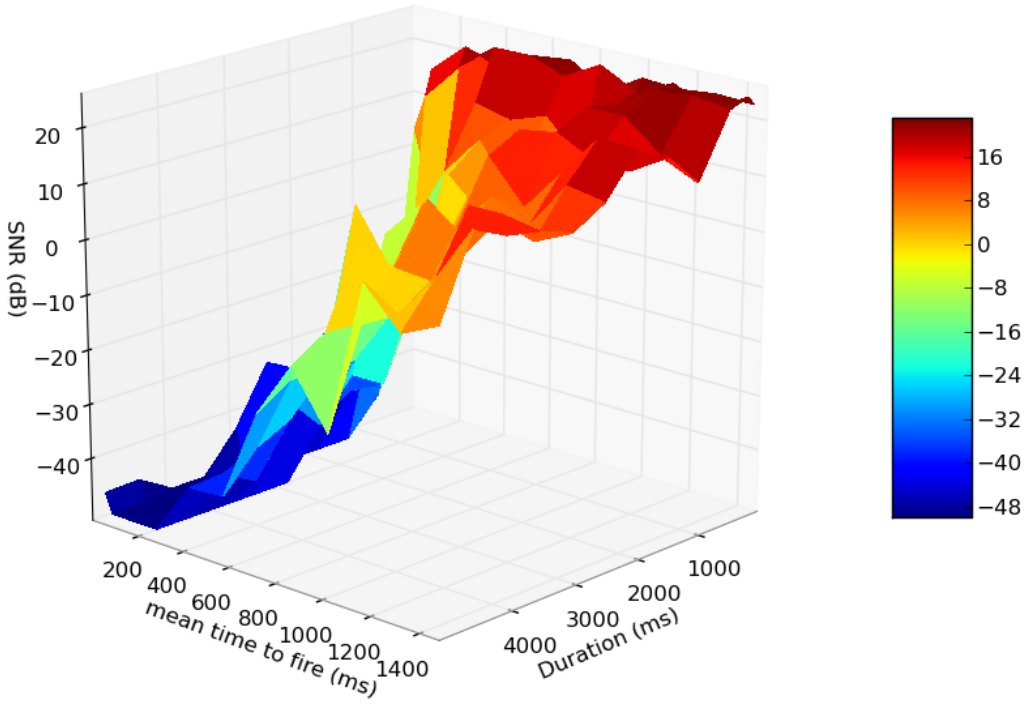
A key aspect of using EDA for neuron simulations is the optimization possibilities provided by the tool at hand. For future work reference we provide Figure 4.11 that illustrates the variabilty in the SNR of each neuron of our network, that is caused by the adjustment of the `maxstep` parameter.

It is clear that a decrease in the maxstep, causes an increase in the SNR value, as the variations in the signal are satisfied by more frequent simulation steps. However, even with twice the step of the Matlab implementation (that generates a 50 % compression of time points), the Spectre simulator shows significant accuracy. The input current spikes for each neuron are of 20 ms width and 1.5 s mean time between each firing for a period of 4 s.

The previous exploration is extended in a 3D graph to show how changes in the maxstep



(a) RMSE (mV)



(b) SNR (dB)

Figure 4.10: Multiple neuron verification: Mean time to fire (Mtf) and Duration vs the average RMSE and SNR of 4 neurons for a constant spike width of 20 ms

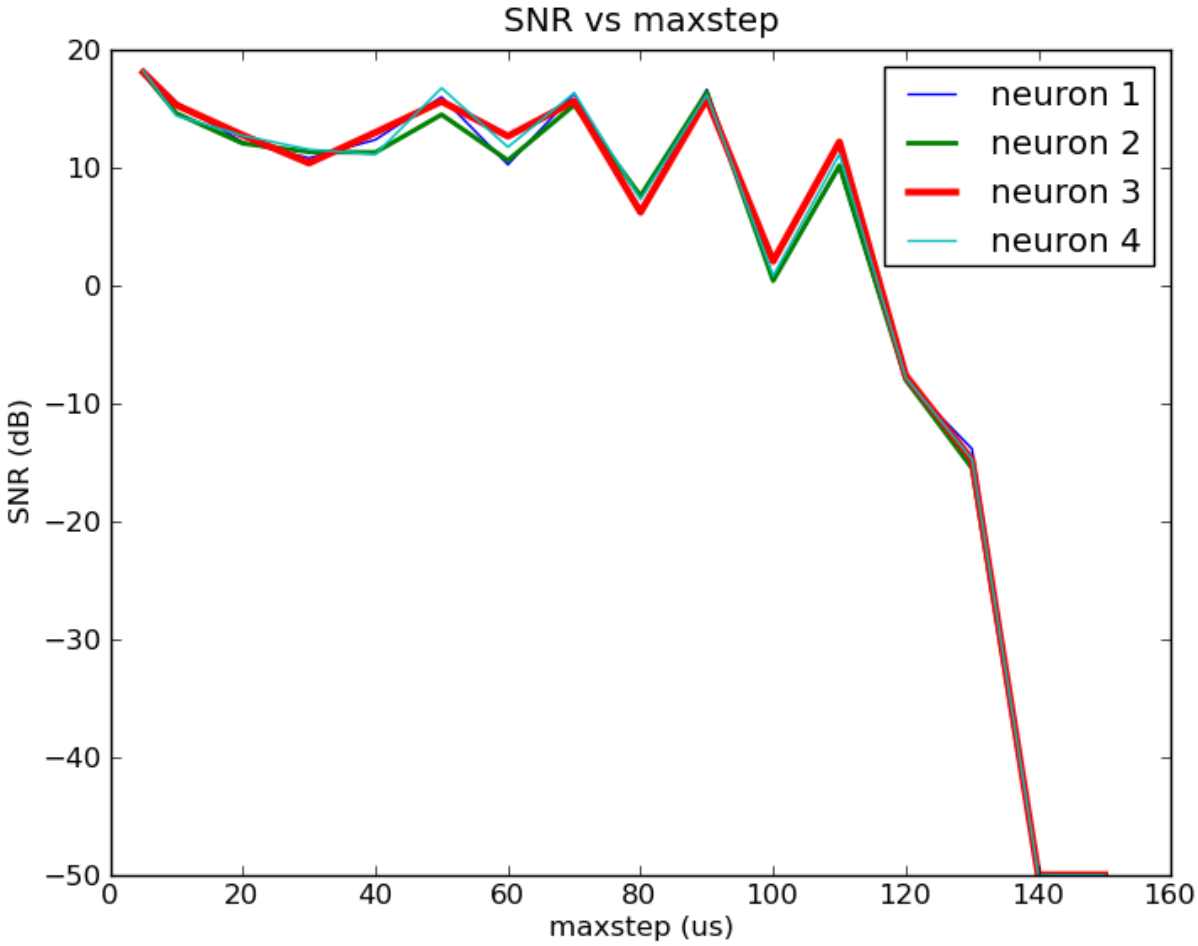


Figure 4.11: SNR for each neuron of the 2 by 2 network, relative to different maxstep values. Each SNR value is the mean of 20 iterations

and Mtf, affect the total compression of time points (Figure 4.12) as well as the average SNR and RMSE of the 4 neurons (Figure 4.13).

It is clear from Figure 4.12, that the `maxstep` parameter value is proportional to the total compression that the Spectre simulator achieves in comparison to the MATLAB implementation. A `maxstep` value of 50 μs, which matches the value of the constant step used in the MATLAB implementation, leads to approximately 0% compression of time points, while a 100 μs `maxstep` leads to almost 50 % compression.

Even though `maxstep` values less than 50 μs are detrimental to the total compression of time points, they cause a significant increase in the SNR value and a subsequent decrease in the RMSE value as illustrated in Figure 4.13. In addition, the Spectre simulator becomes

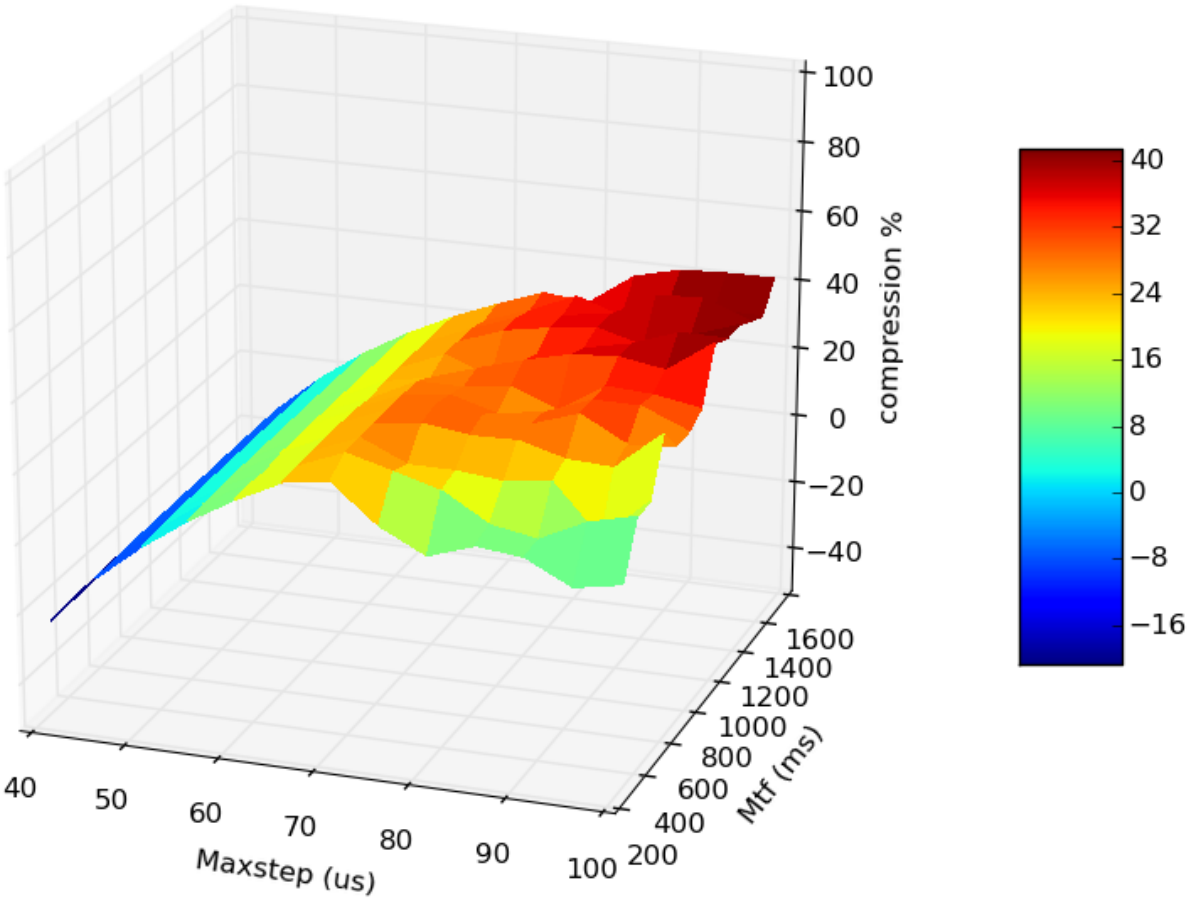
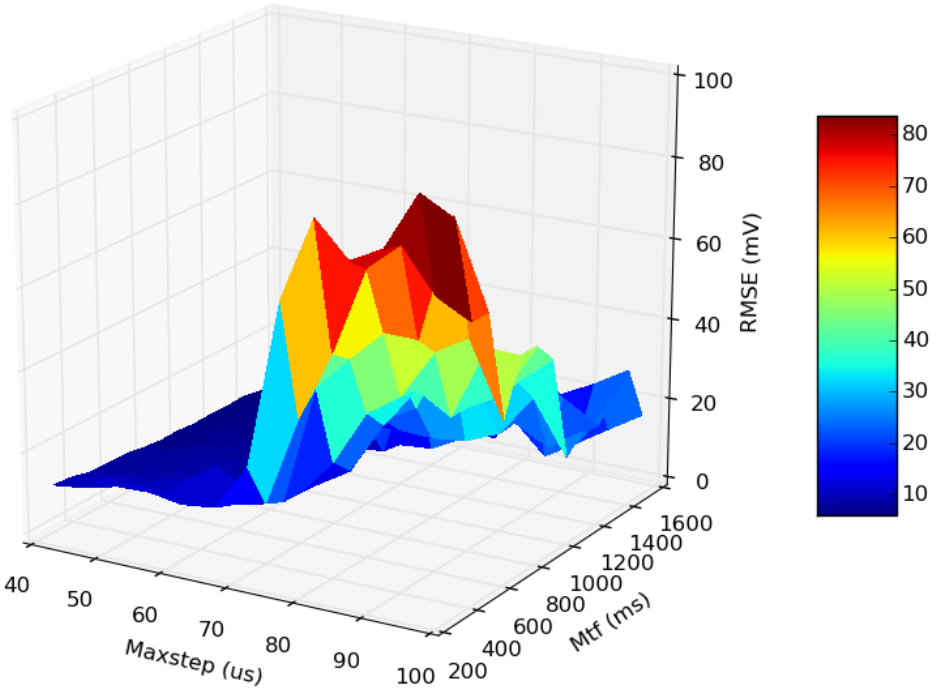
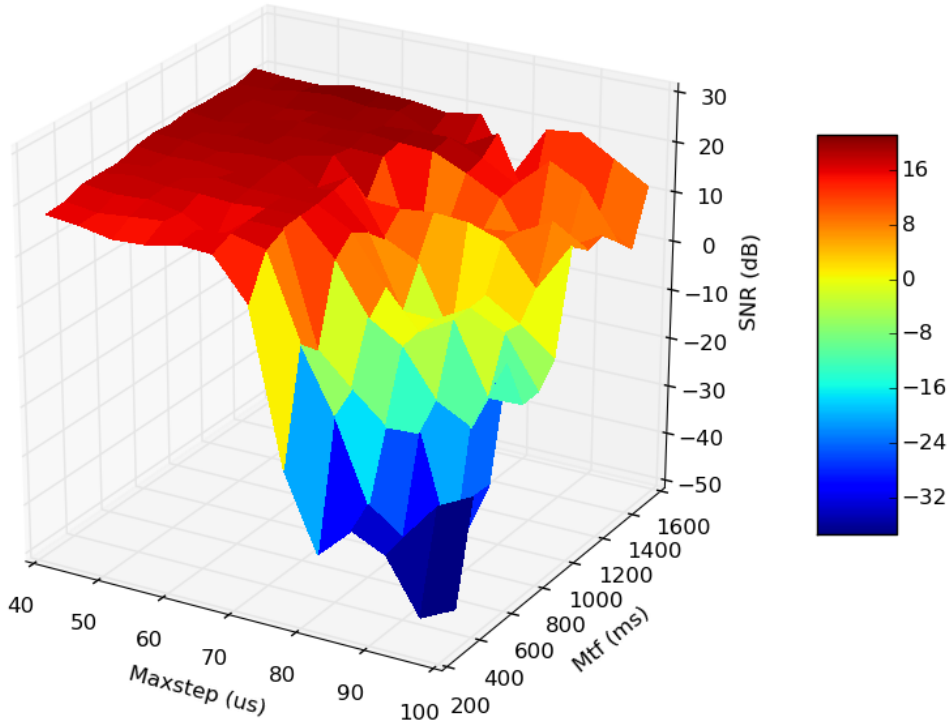


Figure 4.12: Average compression of 4 neurons as a result of variations in maxstep and Mtf for a constant spike width of 25 ms and a duration of 3.5 s

sensitive to the Mtf of the input current for a `maxstep` larger than 50 μs and produces a more satisfying SNR as the Mtf increases. It is clear from 4.13 that increasing the `maxstep` to more than 70 μs will produce satisfying accuracy for mtf values larger than 1 s.



(a) RMSE (mV)



(b) SNR (dB)

Figure 4.13: Multiple neuron verification: Mean time to fire (Mtf) and maxstep vs the average RMSE, SNR of 4 neurons for a constant spike width of 25 ms and a duration of 3.5 s

Chapter 5
Conclusions & Future Work

5.1 Conclusion

This work offers some insight into approaching neuron modeling problems with the use of EDA tools. A brief overview of the EDA possibilities motivates the need for a connection between EDA and neuron modeling.

The main course of action involves using the Verilog-A language to implement highly detailed neuron models, in order to achieve different simulation patterns through the Spectre simulator, which allows experimentation with different inputs, simulation settings and neuron inter-connection schemes.

We present the process of creating the Verilog-A implementation and compare its code base with the original MATLAB code as well as the different compiling methods and simulation settings. Emphasis is given in the adaptive step size of the Spectre simulator, which is illustrated through a simulation of a BSIM4 inverter model.

The verification of the single neuron model starts off with the representation of a single cell membrane voltage oscillation, where the regions of interest for the action potential can be identified. After changing to the more accuracy-friendly `conservative` preset, we conduct a simulation of 150 iterations and examine the two main outlier cases (phase shift, innacurate excitations) that cause an increase in the RMSE and a decrease in the SNR compared to the original MATLAB signal.

Further verification includes simulating our model for sweeping input parameters in order to display 3D graphs showing the relationship between input parameters and accuracy metrics. For the first simulation, a constast current of 50 mA and 20 ms spike width is applied

with a varying mean time between each firing at different execution times. In the second experiment, the Spike Width and the Mtf are the two sweeping variables while the execution time remains constant at 400 ms. Both experiments justify how small Mtf combined with large execution times are the reasons for high error values.

As far as the multi-neuron implementation is concerned, a seperate model is implemented through verilog-A, that also takes the inter-connections between the different compartments of the infOli neuron into consideration. Again, the new model is verified in comparison with the equivalent MATLAB model and a graph is presented, showing the SNR values of the Spectre signal with repsect to the change in the `maxstep` setting of the Spectre simulator.

5.2 Future Work

Our work approaches the field of EDA tools for neural networks in the lower abstraction level, where emphasis is given on the transient response of neuron cells. In Chapter 4, we implement a Verilog-A model for a single neuron and a multi-neuron network and observe the increased accuracy and time compression of the Spectre simulator in comparison to MATLAB.

This work can be further extended by exploring more possibilities provided by the Spectre simulator. The transient response of the models can be explored in terms of adjustments to the relative error parameters, the integration methods, or the analysis' convergance settings. A small fraction of this exploration is presented in Subsection 4.3, where the relation between the `maxstep` parameter of the simulator and the achieved compression is presented.

Another major course of action, also discussed in Subsection 3.3, would be to expand the multi-neuron verilog-A model so that it can form an arbitary network whose size is given as input by the user. Verilog-A allows this kind of implementation with the use of node vectors. Such exploration could show how the increased component complexity interferes with the accuracy the simulator provides. In addition, this arbitary network could enable user-defined inter-connectivity schemes of adjusting intensity, by editing the conductance values of the

gap junctions through the netlist. Consequently, different connectivity patterns could be explored.

Taking the above into consideration, we could construct a datasheet that would provide suggestions for appropriate simulation settings which could lead to specific margins for the required accuracy and compression.

Last but not least, EDA tools can be used for higher abstraction levels in neuron simulation, i.e event-driven simulations where each neuron model implements functions to accept and handle incoming events. Event-based simulations will accelerate the modeling phase of larger neural networks.

References

[1] "Human brain project." https://www.humanbrainproject.eu/roadmap, 2013. 1

[2] "Blue brain project - in brief." http://bluebrain.epfl.ch/page-56882-en.html, 2013. 1

[3] "Erasmus brain project." http://erasmusbrainproject.com/, 2015. 1, 4

[4] Don Monroe, "Neuromorphic computing gets ready for the (really) big time," *Communications of the ACM*, vol. 57, pp. 13–15, June 2014. 2

[5] D. Robert, "Neuromorphic chips," *MIT Technology Review*, April 2013. 2

[6] Sathishkumar Balasubramanian, Pete Hardee, Cadence Design Systems, *Solutions for Mixed-Signal SoC Verification Using Real Number Models.* Cadence Design Systems, 2013. 2

[7] Rodopoulos, D. and Mahato, S.B. and de Almeida Camargo, V.V. and Kaczer, B. and Catthoor, F. and Cosemans, S. and Groeseneken, G. and Papanikolaou, A. and Soudris, D., "Time and workload dependent device variability in circuit simulations," in *IC Design Technology (ICICDT), 2011 IEEE International Conference*, pp. 1–4, May 2011. 2, 13

[8] Andrew Jahn, "Introduction to computational modeling: Hodgkin-huxley model." http://andysbrainblog.blogspot.gr/2013/10/introduction-to-computational-modeling.html, October 2013. 2, 11, 13

[9] J. R. De Gruijl, P. Bazzigaluppi, M. T. G. de Jeu, and C. I. De Zeeuw, "Climbing fiber burst size and olivary sub-threshold oscillations in a network setting," *PLoS Comput Biol*, vol. 8, p. e1002814, 12 2012. 2, 6, 19, 37, 38

[10] Dayan Peter, "Levels of analysis in neural modeling," *Encyclopedia of Cognitive Science*, 2006. 4, 5

[11] A. Burkitt, "A review of the integrate-and-fire neuron model: I. homogeneous synaptic input.," *Biological Cybernetics*, vol. 95, Issue 1, pp. 1–19, 2006. 5

[12] E. M. Izhikevich, "Simple model of spiking neurons," *IEEE Transactions on neural networks, vol.14,no.6*, November 2003. 5

[13] Hodgkin Al, Huxley Af., "A quantitative description of membrane current and its application to conduction and excitation in nerve," *The Journal of Physiology*, August 1952. 5, 13, 17, 32

[14] D. Zeeuw, H. FE, L. Bosman, M. Schonewille, L. Witter, and S. Koekkoek, "Spatiotemporal firing patterns in the cerebellum," Nature Review Neuronscience, May 2011. 6

[15] C. I. D. Zeeuw, C. C. Hoogenraad, S. Koekkoek, T. J. Ruigrok, N. Galjart, and J. I. Simpson, "Microcircuitry and function of the inferior olive," *Trends in Neurosciences*, December 1998. 6

[16] David Kriesel , "A brief introduction to neural networks." `http://www.dkriesel.com`, 2007. iv, 6, 7, 27

[17] Wulfram Gerstner and Werner M. Kistler, "Spiking neuron models," tech. rep., 2002. iv, 6, 10, 19

[18] T. Yu and G. Cauwenberghs, "Analog vlsi neuromorphic network with programmable membrane channel kinetics," in *Circuits and Systems, 2009. ISCAS 2009. IEEE International Symposium on*, pp. 349–352, May 2009. iv, 6, 7

[19] B. Hanafi and Y. Abdelmaksoud, "Emulation of ion channel dynamics in cmos vlsi," University of California San Diego, La Jolla, CA 92093. 7

[20] Ananthanarayanan, R. and Esser, S.K. and Simon, H.D. and Modha, D.S., "The cat is out of the bag: cortical simulations with 109 neurons, 1013 synapses," in *High Per-*

formance Computing Networking, Storage and Analysis, Proceedings of the Conference on, pp. 1–12, Nov 2009. iv, 7, 10

[21] G. Lyras, D. Rodopoulos, A. Papanikolaou, and D. Soudris, "Hypervised transient spice simulations of large netlists & workloads on multi-processor systems," in *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '13, (San Jose, CA, USA), pp. 655–658, EDA Consortium, 2013. 8

[22] "Mentor graphics corporation stock quote & summary data." `http://www.nasdaq.com/symbol/ment`, 2015. 8

[23] "Cadence design systems, inc. stock quote & summary data." `http://www.nasdaq.com/symbol/cdns`, 2015. 8

[24] "Synopsys, inc. stock quote & summary data." `http://www.nasdaq.com/symbol/snps`, 2015. 8

[25] "Software pricing trends - how vendors can capitalize on the shift to new revenue models.." `http://www.pwc.com/en_us/us/technology-innovation-center/assets/softwarepricing_x.pdf`, 2007. iv, 8

[26] L. W. Nagel and D. Pederson, "Spice (simulation program with integrated circuit emphasis)," Tech. Rep. UCB/ERL M382, EECS Department, University of California, Berkeley, Apr 1973. 9

[27] L. W. Nagel, *SPICE2: A Computer Program to Simulate Semiconductor Circuits*. PhD thesis, EECS Department, University of California, Berkeley, 1975. 9

[28] Intel Corporation, "Microprocessor quick reference guide." `http://www.intel.com/pressroom/kits/quickreffam.htm`. iv, 10

[29] Dimitrios Rodopoulos, "Neuron modeling: Drawing hints from the eda industry," tech. rep., Microlab–ECE–NTUA, July 2015. 9

[30] Daniel Gajski, "Guest editors' introduction: New vlsi tools.e," *IEEE Computer*, December 1983. iv, 11

[31] John Rinzel, "Computational modeling of neuronal systems." http://www.cns.nyu.edu/~rinzel/CMNSF07/Neuronal%20dyns%20cell_CMNSF07.pdf, 2007. 10

[32] "Open mpi: Open source high performance computing." http://www.open-mpi.org/. 11

[33] M. Diesmann and M. Gewaltig, "Nest (neural simulation tool)." http://www.scholarpedia.org/article/NEST_%28NEural_Simulation_Tool%29. 11

[34] N. Imam, K. Wecker, J. Tse, R. Karmazin, and R. Manohar, "Neural spiking dynamics in asynchronous digital circuits," *In Neural Networks (IJCNN), The 2013 International Joint Conference*, August 2013. 11

[35] Cadence Design Systems, Inc., *Spectre Circuit Simulator User Guide*, 2004. vi, 16, 24, 29, 30

[36] SIMetrix Technologies Ltd., "Simetrix verilog-a manual," tech. rep., September 2010. 18, 19, 21

[37] William H. Press, Saul A. Teukolsky, William T. Vetterling and Brian P. Flannery, *Numerical recipies in C: The art of scientific computing*, 1992. 23

[38] Butcher, John C., *Numerical Methods for Ordinary Differential Equations, 2nd Edition.* John Wiley & Sons, March 2008. 24

[39] Atkinson, Kendall E., *An Introduction to Numerical Analysis, 2nd Edition.* John Wiley & Sons, 1989. 24

[40] George Chatzikonstantis, "Energy aware mapping of a biologically accurate inferior olive cell model on the single-chip cloud computer," December 2013. 25

[41] "Spectre/spectrerf simulation accuracy." http://www.designers-guide.org/Forum/YaBB.pl?num=1173722684. vi, 29

[42] Ivan Poliacek, Jan Jakus, "Biophysics of action potential & synapse." http://eng.jfmed.uniba.sk/fileadmin/user_upload/editors/Biof_Files/skusky/prednasky/Action_potential__synapse.ppt. 31