Resum

En el present volum inclou el annexos A i B referits al prototipus d'eina de software en matlab implementada seguint la metodologia i els models explicats a la memòria.

L'annex A mostra tota la informació referent al codi utilitzat, algoritme base, funcions i estructures de dades desenvolupades.

L'annex B és el manual d'usuari del programa.

Sumari

SUMARI 3
A. PROTOTIPUS DE SOFTWARE EN MATLAB 5
A.1. Algoritme base 5
A.2. Estructures de dades 7
A.2.1. Estructures de dades per la definició de xip 7
A.2.2. Estructura de dades Stimuli 17
A.2.3. Estructures de dades Matrix 20
A.2.4. Estructura de dades Boundary 28
A.2.5. Estructures de dades Frequency results 30
A.2.6. Estructura de dades de definició de l'output 32
A.2.7. Funcions consultores universals 34
A.3. Funcions 37
A.3.1. Funcions de definició d'Input 37
A.3.2. Funcions principals 52
A.3.3. Funcions consultores 85
A.3.4. Altres functions 100
A.4. Comparació de resultats amb simulador de circuits 105
A.5. Referències 106
B. MANUAL D'USUARI 107
B.1. Executant el programa 107
B.2. Creant inputs 107
B.3. Entenent l'output 115

A. Prototipus de software en Matlab

A.1. Algoritme base

Hem implementat en matlab un prototip de software que a partir de una definició de xip amb la seva activitat estima el soroll que tindrà (nivell de tensió i bucle de corrent entre la alimentació local). La implementació és compatible amb matlab 6.5 i matlab 7

La següent figura mostra l'algoritme base utilitzat.

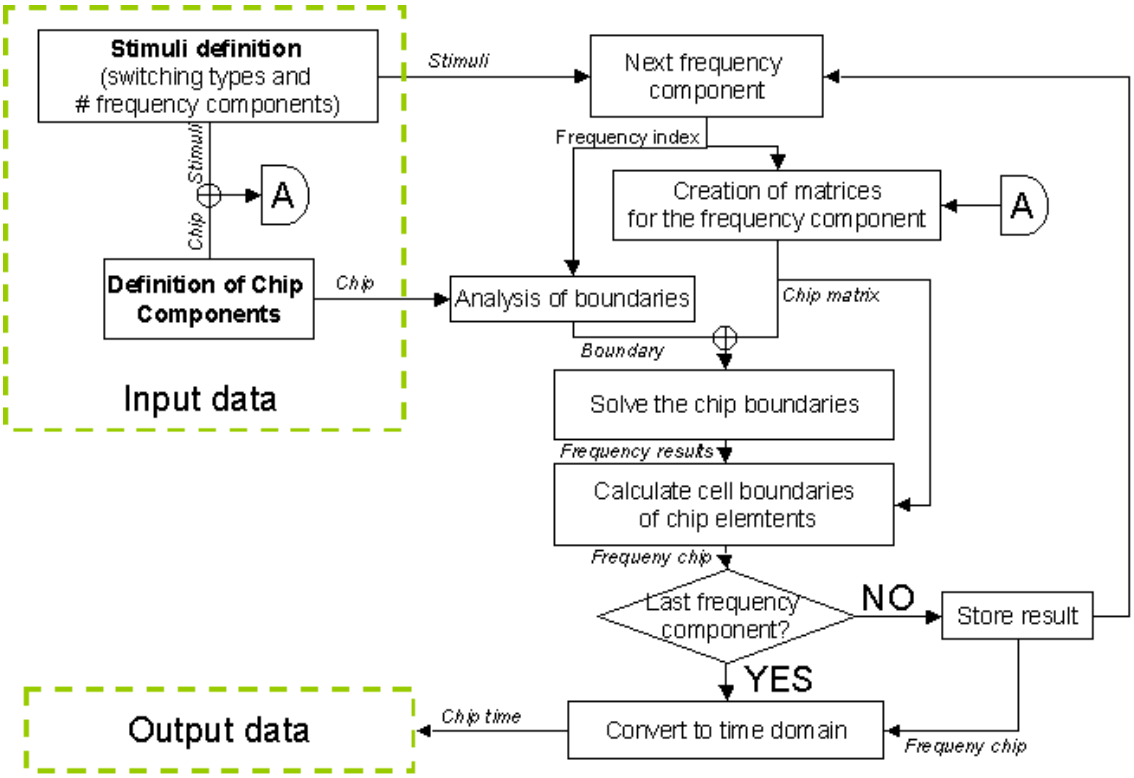

Figura A.1.1: Algoritme base. En els rectangles operacions, a les fletxes, estructures de dades amb la informació.

Les següents fitxers de funcions implementen les operacions anteriors:

- *Av_chip.m*: Funció font, salvar resultats i obtenció de la següent freqüència
- *getStim.m*: Definició del estímuls.
- *getInputs.m:* Definició del components del xip.

- *buildChip.m*: Creació de matrius donada una freqüència.
- *giveBoundaries.m*: Anàlisis del contorn del xip.
- *SolveChip.m*: Solució de les variables de contorn.
- *getResults.m*: Càlcul dels contorns dels elements del xip.
- *convertToTime.m*: Conversió al domini del temps:

EL codi fa els càlculs a través de la creació i la manipulació d'estructures de dades. La següent secció mostra com estan definides aquestes estructures.

A.2. Estructures de dades

Una estructura de dades és una organització de informació que facilita el seu enteniment i el seu procés. Per tal de comprendre i fer servir estructures de dades és convenient definir funcions constructores, que creen la estructura de dades, i definir funcions consultores que donen una informació de dins.

Aquest procediment permet fer millores sobre el codi amb facilitat degut a que qualsevol canvi en la definició de l'estructura de dades només suposa adaptar les funcions consultores i constructores a la nova estructura.

A.2.1. **Estructures de dades per la definició de xip**

Estructura de dades *Chip*

A l'estructura de dades *Chip* és on es defineixen les característiques del xip. Pels nostres objectius es defineix com a xip a un conjunt de *blokcs* i de *stripes* en un espai de dos dimensions. Aquest espai de dos dimensional es divideix en files (eix y) i columnes (eix x). Un xip en l'espai és l'àrea mínima rectangular que inclou tots els *blocks* i *Stripes* que li pertanyen.

L'estructura de dades *Chip* guarda tota la informació referent a la composició i posició del xip. La figura següent mostra com ha estat definida aquesta estructura de dades.

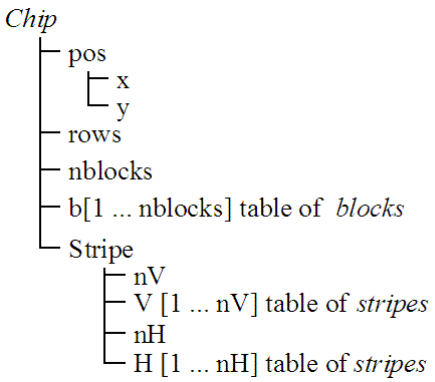

Figura A.2.1: Estructura de dades *Chip*.

El significat dels camps és el següent:

- pos.x: Posició de la coordenada x on està posicionat el cantó inferior a l'esquerre del xip. En unitats de columna absolutes.
- pos.y: Posició de la coordenada y on està situat el cantó inferior a l'esquerre del xip. En unitats de fila absolutes.
- rows: Longitud en files del xip.
- nblocks: Nombre de blocs en que el xip es composa.
- b: Vector de *blocks*.
- Stripe.nV: Nombre de stripes orientats verticalment.
- V: Vector de *stripe*s verticals.
- Stripe.nH: Nombre de stripes orientats horitzontalment.
- H: Vector de *stripe*s horitzontals.

Funcions constructores:

- **iniChip.m**: function [c] = iniChip(param)

INICHIP inicialitza un xip. El resultat *c* is una estructura *chip* amb els camps inicialitzats a 0. *Param* pot ser qualsevol valor, per a Matlab 6.5 és necessari posar algun paràmetre d'entrada.

- **addBlock.m:** function [c] = addBlock(c, b, x, y)

ADDBLOCK inclou el *block b* com a component del xip *c* en les coordenades (*x*, *y*) (coordenades absolutes del cantó inferior a l'esquerre del bloc).

- **addStripe.m:** function [c] = addStripe(c, S, x, y, type)

ADDSTRIPE inclou el stripe *S*, com a component del xip *c* en les coordenades (*x*, *y*) (coordenades absolutes del cantó inferior a l'esquerre del bloc). *Type* dona el tipus d'stripe que *S* és, pot ser 'V' si és vertical *stripe* o 'H' si és horitzontal *stripe*.

Funcions consultores:

- **giveBlock.m:** function [b] = giveBlock(c, k)

GIVEBLOCK. El resultat és el *block k* del *chip c.*

- **giveStripe.m:** function [S] = giveStripe(c, type, k)

GIVESTRIPE. El resultat és l'stripe *k* del *chip c*. *Type* és 'V' si és *stripe* vertical o 'H' si és *stripe* horitzontal.

Estructura de dades *Block*

L'estructura de dades *block* té com a missió guardar la informació dels conjunts de cel·les de referència, agrupades regularment en files i columnes formant blocs. La figura següent mostra com esta constituïda la estructura *block*:

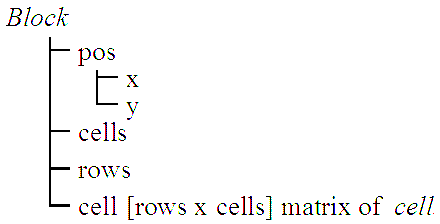

Figura A.2.2: Estructura de dades *Block*.

El significat dels camps és:

- pos.x: Posició de la coordenada x on està situat el cantó inferior a l'esquerre del *block*. En unitats de columna absolutes.
- pos.y: Posició de la coordenada y on està situat el cantó inferior a l'esquerre del *block*. En unitats de fila absolutes.
- cells: nombre de cel·les per fila dins del bloc (o nombre de columnes dins del bloc).
- rows: nombre de files de que es composa el bloc.

- cell: matriu de *cell* estructures. *cell(i,j)* correspon a la cel·la situada a la fila *i* columna *j* del bloc.

Funcions constructores:

- **CreateBlocklength.m:** function [block] = createBlockLength(rows,length,fu)

CREATEBLOCKLENGTH crea una estructura de dades *block* a través de saber la seva longitud *length* en [μm], el seu nombre de files *rows* i el factor d'utilització de les files *fu*. El factor d'utilització és la relació entre la longitud real d'una fila del bloc i la que tindria si la fila estigués feta de *reference gates*.

- **CreateBlockcells.m:** function [block] = createBlockCell(rows, cells, fu)

CREATEBLOCKCELL crea una estructura de dades *block* a través de saber el seu nombre de cel·les per fila, el seu nombre de files i el seu factor d'utilització *fu*.

- **addCell.m:** function [b] = addCell(b, cell, r, c)

ADDCELL inclou les propietats de la cel·la *cell* a la cel·la de la columna *c*, fila *r* del bloc *b*.

Funcions consultores:

- **giveCell.m:** function [cell] = giveCell(B, r, c)

GIVECELL retorna l'estructura de dades *cell* que representa la cel·la situada a la fila *r,* columna *c* del bloc *B*.

Funcions associades:

- **ObtainZ1Z2.m**: function [Z1, Z2] = obtainZ1Z2(block, f)

OBTAINZ1Z2 calcula la impedància sèrie z_1 i la paral·lel z_2 per a totes les cel·les del bloc *B* a la freqüència *f*. El resultat Z1 i Z2 són dos matrius Z1(*i* ,*j*) i Z2(*i* ,*j*) amb els valors de is z_1, z_2 per a cada cel·la *j* de la fila *i* del bloc *B* a la freqüència *f*.

Estructura de dades *Cell*

La cel·la és la unitat bàsica (1 columna de longitud per 1 fila d'ample) que té propietats elèctriques. L'estructura de dades que representa a la cel·la és el *cell*. Les propietats de la cel·la són representats per la sub-estructura *RLG*:

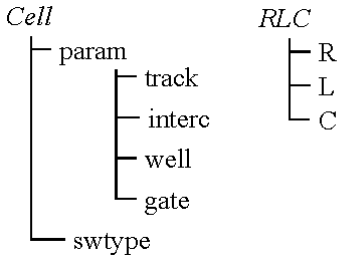

Figura A.2.3: Estructures de dades *Cell* i *RLC*.

Track, *interc*, *well* i *gate* són estructures de dades *RLC*. L'estructura de dades *RLC* conté la informació dels components d'una branca elèctrica composta per una resistència en sèrie amb una capacitància i una inductància. *Swtype* és un enter amb el tipus de commutació de la cel·la (*swtype*=0 significa cel·la sense commutació); els tipus es defineixen a l'estructura de dades *stimuli*.

Funcions constructores:

- **createCell.m:** function [cell] = createCell(track, well, interc, gate, swtype)

CREATECELL genera una estructura de dades *cell* a través de fer servir les propietats elèctriques dels metalls de l'alimentació (*track*), del pou (*well*), de les interconnexions (*interc*) i de la porta (*gate*), cada una definida com a *RLC*. La cel·la resultant tindrà el tipus de commutació *swtype* (*swtype*=0 significa no commuta, els altres tipus es defineixen a *stimuli*).

- **GetInputs.m:** function [T] = getTrackParam(length, cells)

GETTRACKPARAM retorna una estructura *RLC* amb la representació de la impedància dels conductors d'alimentació del bloc per a una cel·la, a través de saber la longitud de les files *length,* i el nombre de cel·les per fila *cells*. Propietats com amplada de conductor o la *sheet resistance* estan definits dins de la funció per a CMOS090.

- **GetInputs.m:** function [T] = getWell(length, cells)

GETWELL retorna una estructura RLC amb la representació de la impedància de pou per a una cel·la, a través de saber la longitud de fila *length* i el nombre de cel·les per fila *cells*. Els paràmetres que defineixen les propietats del pou estan definits dins de la funció.

- **GetInputs.m:** function [T] = getInterconnect(s)

GETINTERCONNECT retorna una estructura RLC amb la representació de la impedància de interconnexió per una cel·la. *s* és una cadena de caràcters amb el tipus d'interconnexió requerida, els tipus d'interconnexió estan definits dins la funció. Actualment esta disponible el tipus 'philips' amb les dades estadístiques de Philips per a interconnexió de cel·la.

- **GetInputs.m:** function [T] = getGate(length, cells)

GETGATE retorna una estructura RLC amb la representació de la impedància de porta. *s* és una cadena de caràcters amb el tipus de cel·la requerida. Els tipus de porta estan definits dins de la funció, actualment hi ha definits dos, 'reference' i 'decap'.

Funcions consultores:

- **giveType.m**: function [type] = giveType(cell)

GIVETYPE retorna el tipus de commutació de la cel·la *cell*. Els tipus de commutació estan definits a la estructura de dades *stimuli*, *type*=0 sempre significa no commutació.

- **GiveTrack.m**: function [R, L, C] = giveTrack(cell)

GIVETRACK retorna els valors de resistència, inductància i capacitància (*R*, *L*, *C*) que defineixen la impedància sèrie dels conductors d'alimentació de la cel·la *cell.*

- **GiveGateParam.m**: function [R, C, L] = giveGateParam(cell)

GIVEGATEPARAM retorna els valors de resistència, inductància i capacitància (*R*, *L*, *C*) que defineixen la impedància sèrie de la impedància de porta de la cel·la *cell.*

- **GiveInterc.m**: function [R, L, C] = giveInterc(cell)

GIVEINTERC retorna els valors de resistència, inductància i capacitància (*R*, *L*, *C*) que defineixen la impedància sèrie de les interconnexions de la cel·la *cell.*.

- **GiveWell.m**: function [R, C, L] = giveWell(cell)

GIVEWELL retorna els valors de resistència, inductància i capacitància (*R*, *L*, *C*) que defineixen la impedància sèrie de la proporció de pou corresponent a la cel·la *cell.*

Estructura de dades *Stripe*

L'stripe és un component del xip amb la funció de crear una xarxa secundaria que alimenti les alimentacions internes dels blocs tot connectant aquestes a fonts de tensió a través dels pads.

És composa per dos conductors un per V_{DD} i un altre per *GND.* Igual que els blocs els stripes estan segmentats en cel·les. Si la posició de l'stripe es vertical, les cel·les formen una columna, i si es horitzontal formen una fila. L'estructura de dades que representa a l'stripe es mostra a la figura següent:

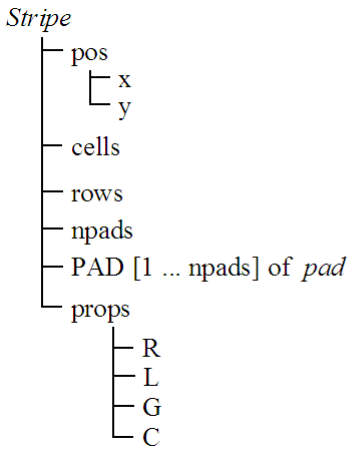

Figura A.2.4: Estructura de dades *Stripe*.

El significat dels camps és:

- pos.x: Coordenada x amb la posició del cantó inferior a la esquerre de l'stripe. En unitats de columnes absolutes.
- pos.y: Coordenada y amb la posició del cantó inferior a la esquerre de l'stripe. En unitats de files absolutes.
- cells: Nombre de divisions en columnes de l'stripe.
- rows: Nombre de files de l'stripe.
- npads: Nombre de pads situats en l'stripe.
- PAD: Vector de *PADs* amb la informació sobre els pads de l'stripe.
- props: Propietats elèctriques de l'stripe. R és resistència dels conductors, L inductància dels conductors, G i C resistència i capacitància entre un conductor V_{DD} i un conductor *GND*.

Funcions constructores:

- **createStripe.m:** function [S] = createStripe(props, size, type)

CREATESTRIPE posa valor als camps *props*, *cols* i *rows* i inicialitza els altres camps del *Stripe S*. L'input *props* és una estructura de dades amb les propietats (R, L, G, C) de l'stripe, *size* dona la llargària de l'stripe, el nombre de files per a un stripe *Vertical* o de columnes de cel·les per a un *Horitzontal. Type* és el caràcter 'V' o 'H' depenent de la orientació de l'stripe, vertical o horitzontal.

- **addPAD.m:** function [s] = addPAD(s, pad)

ADDPAD inclou el pad *pad* en l'stripe *s*.

- **getInputs.m:** function [S] = getStripHProps(l)

GETSTRIPHPROPS retorna els valors dels paràmetres R, L, G, C que defineixen l'stripe horitzontal. El resultat *S* és el camp *stripe.props* de la estructura de dades *Stripe*. L'input *l* pot tenir qualsevol valor, la seva utilització es necessària perquè a la versió 6.5 de matlab6.5 tota funció necessita un input.

- **getInputs.m:** function [S] = getStripVProps(l)

GETSTRIPVPROPS retorna els valors dels paràmetres R, L, G, C que defineixen l'stripe vertical. El resultat *S* és el camp *stripe.props* de la estructura de dades *Stripe*. El input *l* pot tenir qualsevol valor, la seva utilització es necessària perquè a la versió 6.5 de matlab6.5 tota funció necessita un input.

Funcions consultores:

- **givePad.m:** function [pad] = givePad(s, n)

GIVEPAD retorna el *pad* n de l'*stripe s*.

- **giveStripProps:** function [R, L, G, C] = giveStripProps(S)

GIVESTRIPPROPS retorna la resistència i la inductància dels conductors de l'stripe (R, L) i la resistència i capacitància entre els conductors de V_{DD} i de GND de una cel·la del stripe *S*.

Pad structure

El pad el component de l'stripe que connecta el xip amb el mon extern, la connexió es fa a través de *bonding wires*, fils que connecten als pins, aquest component està inclòs dins de les propietats del pad.

Tota la informació necessària per definir un pad esta estructurada segons mostra la següent figura:

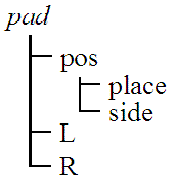

Figura A.2.5:Estructura de dades *PAD*.

El significat els camps és:

- pos.place: numero de la fila o columna dins de l'stripe on s'ha situat el pad.
- pos.side: Pot tenir tres valors 'r' , 'l' o 0. Dona la informació de si el pad es connecta amb el mon exterior per la seva dreta ('r') o per la seva esquerra ('l'). El seu valor és 0 en els stripes horizontals.
- L: Valor de inductància de la *bonding wire*.
- R: Valor de resistència de la *bonding wire*.

Funcions constructores:

- **MakePADH.m:** function [P] = makePadH(x, L, R)

MAKEPADH crea una estructura de pad *P* per a un stripe horitzontal. El pad es situarà a la cel·la *x* amb *bonding wire* inductància i resistència *L* i *R*.

- **MakePADV.m:** function [P] = makePadH(x, L, R, c)

MAKEPADH crea una estructura pad *P* per a un stripe vertical. El pad es situa a la fila *x* en el costat *c* de l'stripe amb *bonding wire* inductància i resistència *L* i *R*. El costat *c* és un caràcter amb valors 'l' o 'r' depenent del costat, al costat dret 'r' i al costat esquerre 'l'.

Funcions consultores:

- **givePadPos.m:** function [pos, side] = givePadPos(pad, type)

GIVEPADPOS retorna la posició *pos* (fila o columna on s'ha situat el pad *pad*) i el costat respecte l'stripe on s'ha situat ('l', esquerra; 'r' dreta; 0, si és horitzontal). *Type* és un caràcter indicant si l'stripe on està situat el pad es vertical 'V' o horitzontal 'H'.

- **givePadProps.m:** function [R, L] = givePadProps(pad)

GIVEPADPROPS retorna els valors de resistència *R* i de inductància *L* per a la *bonding wire* del pad *pad*.

A.2.2. Estructura de dades Stimuli

L'objecte de la estructura de dades *Stimuli* és la definició dels estímuls. Com a estímul estan els diferents tipus de corrents de commutació que tindran les cel·les del xip. La estructura de dades *stimuli* també inclou tota la informació referent als temps i freqüències necessàries per a fer l'anàlisi. La figura següent mostra la configuració de la estructura de dades *stimuli*.

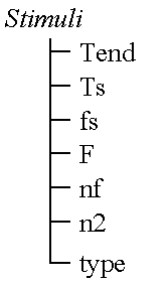

Figura A.2.6: Estructura de dades Stimuli.

El significat dels camps és:

- Tend: Període de les commutacions.
- Ts: Període de mostreig.
- fs: Pas de freqüència necessari per fer l'anàlisi.
- F: Vector amb l'eix de freqüències.
- nf: Nombre de freqüències de l'anàlisi (freqüències positives, negatives i component continua (freqüència zero).
- n2: Nombre de freqüències necessàries per la descripció (freqüències positives i component continua (freqüència zero).
- type: Matriu amb els les formes d'ona en el domini de la freqüència segons tipus de commutació. Cada corrent de commutació is un vector amb els components freqüencials, la matriu es composa de columnes d'aquests vectors, l'índex de columna determina el tipus commutació.

Funcions constructores:

- **createStim.m:** function [stim] = createStim(Tend, Ts)

CREATESTIM crea i inicialitza la estructura de dades *stimuli stim* amb formes d'ona de corrent que en el domini del temps tenen un període de commutació de *Tend* i estan mostrejades segons *Ts*.

- **addType.m:** function [stim] = addType(stim, type, fwav)

ADDTYPE inserta un nou tipus de corrent de commutació a la estructura de dades *stimuli stim*. *Type* és l'enter amb el nom del nou tipus de corrent de commutació, *fwav* és un vector amb les components freqüencials de la forma d'ona del corrent.

Funcions consultores:

- **give_freq.m:** function [f] = give_freq(stim, i)

GIVE_FREQ retorna el valor de freqüència [Hz] al que pertany la posició *i* de l'eix de freqüències. *i* és un índex, *i*=1 significa freqüència 0 [Hz].

- **give_nf.m:** function [nf] = give_nf(stim)

GIVE_NF retorna el nombre de freqüències necessàries de l'estímul *stim* per tenir un corrent definit. Aquest nombre inclou freqüències positives i negatives.

- **give_n2.m:** function [n2] = give_nf(stim)

GIVE_N2 retorna el nombre mínim de freqüències necessàries a l'estímul *stim* per poder operar amb un corrent. Aquest número només inclou les freqüències positives o 0.

Funcions relacionades:

- **gen_step.m:** function [fw2, F] = gen_step(t1, t2, Ts, Tend)

GEN_STEP genera un vector amb les components freqüencials d'un corrent de commutació (*fw2)* que correspon a un graó en el domini del temps. Els paràmetres d'entrada són: el temps on es fa la pujada del graó (*t1)*, el temps de la baixada del graó (*t2)*, el temps de mostreig (*Ts*) i el període de repetició (*Tend*). *F* és un vector amb l'eix de freqüències.

Si una estructura *stimuli* té el mateix mostreig *Ts* i el mateix període de commutació *Tend*, la sortida fw2 pot ser inserida com a nou tipus de commutació fent servir la funció *addType*.m.

- **nfreqs.m:** function [n2, nf] = nfreqs(freq)

NFREQS retorna el nombre de punts que té el vector de freqüències *freq* (*n2*), i el nombre de punts necessaris (*nf*) per poder aplicar la transformada de Fourier inversa (IFFT). La entrada *freq* és un vector amb l'eix de freqüències composat només de les freqüències positives (incloent el 0).

A.2.3. Estructures de dades Matrix

La família de estructures de dades *Matrix* tenen com a objecte la descripció i emmagatzematge de les matrius de tots els components del xip per a una determinada freqüència.

Estructura de dades *Chip Matrix*

La estructura de dades *Chip matrix* és l'arrel de totes les estructures *Matrix* en ella està emmagatzemada tota la informació necessària per descriure el comportament dinàmic del xip. El xip està dividit segons les cel·les, horitzontalment en files i verticalment en columnes. Cada divisió té associada un voltatge i un corrent.

Donat que en un xip hi hauran masses columnes, per tal d'estalviar espai a la memòria, s'ha fet un altre tipus de divisió vertical, seguint el vector d'estructures *x*. Aquest vector divideix el xip verticalment en columnes de components.

El costat esquerre de cada component defineix els límits de la columna; la solució d'aquesta divisió donarà el valor de les variables de contorn de cada component. Aquesta divisió vertical afecta a totes les files, si en alguna divisió vertical existeixen files sense pertànyer a cap costat esquerre de component aleshores se li assigna *empty space*.

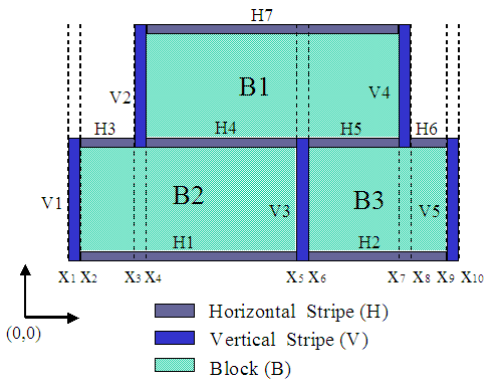

Figura A.2.7. Exemple de divisió vertical en un xip.

La informació s'organitza globalment i per components. Globalment *chip matrix* fa servir les divisions verticals per descriure les transferències de voltatges i corrents entre divisions verticals amb matrius **AX** i **BX**. Per components guarda altres estructures *matrix* amb les matrius necessàries per calcular els voltatges i corrents interns d'un component a partir dels valors d'un dels seus contorns. La configuració de *Chip Matrix* es mostra a la següent figura:

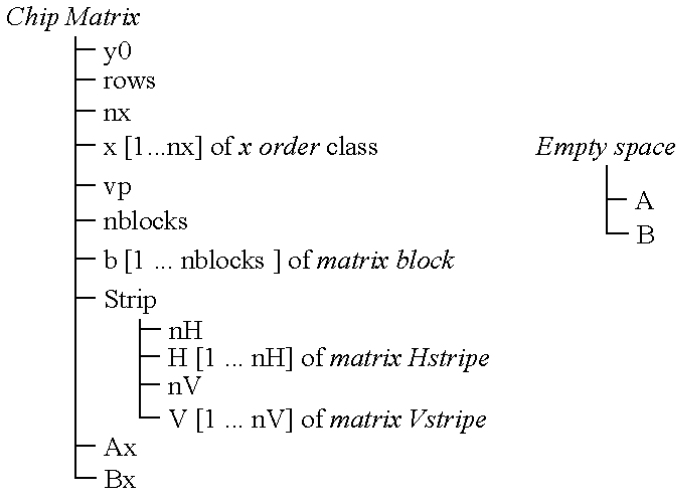

Figura A.2.8: Estructures de dades Chip Matrix i *Empty* space

El significat dels camps de *Chip Matrix* és:

- y0: Coordenada absoluta del costat esquerre del xip.
- rows: Ample del xip (nombre de files del xip).
- nx: nombre de divisions verticals per elements del xip.
- x: vector de estructures *x order* amb la informació relacionada amb les divisions verticals per elements.
- vp: Voltatge de la font d'alimentació connectada al pad per a la freqüència de en que està definida la estructura *chip matrix*.
- nblocks: nombre de blocs.
- b: vector de estructures de dades *matrix block*.
- Stripe.nH: nombre de stripes orientats horitzontalment en el xip.
- Stripe.H: vector de estructures de dades *matrix Hstripe*.
- Stripe.nH: nombre de stripes orientats verticalment en el xip.
- Stripe.H: Vector de estructures de dades *matrix Vstripe*.

- Ax: *Ax*(*i*,*j*,*k*) matriu tridimensional amb les relacions globals (per a totes les files) donades per les matrius A_x entre divisions verticals. L'índex *k* indica la divisió vertical *x* a la que pertany.
- Bx: *Bx(i,k)* matriu de dos dimensions amb els vectors de termes independents B_x entre les divisions verticals *x*. L'índex *k* indica la divisió vertical a la que pertany.

($\mathbf{x}_{k+1}=\mathbf{A}\mathbf{x}_k\cdot\mathbf{x}_k+\mathbf{B}\mathbf{x}_k$. $\mathbf{x_k}$ és un vector de voltatges i corrents per tota la divisió vertical).

Associades a algunes funcions de *Chip Matrix* està l'estructura de dades *Empty space*.

Funcions constructores:

- **BuildChip.m:** function [M] = BuildChip(c, stim, fi)

BUILDCHIP crea *M,* una estructura de dades *chip matrix,* a partir del *chip* definit a *c* amb l'estimulació definida a *stim* per a la freqüència definida per l'índex *fi*.

Funcions consultores:

- **numx.m:** function [nx] = numx(CM)

NUMX retorna el nombre de divisions verticals per element que té el xip definit per *CM*. *CM* és una estructura de dades *chip matrix*.

- **giveXorder.m:** function [x] = giveXorder(CM, k)

GIVEXORDER retorna la estructura de dades *x X-order* amb la informació de la composició de la divisió vertical *k*-essima del xip definit per la *chip matrix* estructura *CM*.

- **giveGlobalM.m:** function [AX, BX] = giveGlobalM(CM)

GIVEGLOBALM retorna les matrius *AX* i *BX* de transferència entre divisions verticals del xip definit per la *chip matrix* estructura *CM*. *AX* és una matriu tridimensional *AX(i,j,k)*, *BX* is

una matriu de dos dimensions *BX(i,k)*. Per a cada *k* AX és una matriu de dos dimensions i BX in vector columna que compleixen: $\begin{bmatrix} \mathbf{v} \\ \mathbf{i} \end{bmatrix}_{k+1} = \mathbf{AX}(k) \cdot \begin{bmatrix} \mathbf{v} \\ \mathbf{i} \end{bmatrix}_{k} + \mathbf{BX}(k)$; amb **v** i **i** vectors columna amb les variables de les divisions verticals x_k i x_{k+1} .

- **giveM.m:** function [M] = giveM (MS, order)

GIVEM retorna la estructura de dades *M* (*matrix block*, *matrix Vstripe* o *matrix Hstripe*) que defineix el component descrit al input *order. Order* és un estructura de dades *y-order*.

- **giveABmatrices.m:** function [A, B] = giveABmatrices(M, c)

GIVEABMATRICES retorna les matrius *A* i els vectors *B* que defineixen la relació entre contorns $\mathbf{b}_+ = \mathbf{A} \cdot \mathbf{b}_- + \mathbf{B}$ (amb **b** un vector amb les variables de contorn, voltatges i corrents) per *M*. *M* pot ser *matrix block*, *matrix Vstripe*, *matrix Hstripe* o *Empty space*. El altre input, *c* és un caràcter amb el tipus de la estructura de dades que és *M* ('b' per a *matrix block*, 'V' per a *matrix Vstripe*, 'H' *matrix Hstripe* o '0' per a *Empty space*).

- **Y0.m:** function [y] = Y0(CM)

Y0 retorna l'origen de coordenades y (en unitats de fila) del xip representat per *CM* (*chip matrix*).

Estructures de dades *X-order* i *Y-order*

Les estructures de dades X-order, Y-order tenen la funció d'organitzar el xip a través de conèixer la posició en l'espai dels seus components. X-order divideix el xip verticalment considerant el contorn esquerre de cada element com a divisió vertical de tot el xip. Y-order emmagatzema la informació de tots els elements que comparteixen la mateixa divisió vertical, de a baix a dalt.

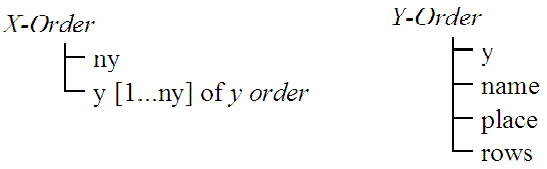

Figura A.2.9: Estructures de dades X-Order (esquerra) i Y-Order (dreta).

Funcions constructores:

- **sort_x.m:** function [x, nx] = sort_x

SORT_X crea una taula *x* d'estructures *X-order* associada al *chip c*. El criteri d'ordenació de la taula és seguint la direcció horitzontal del xip, d'esquerre a dreta, tots els components amb mateixa coordenada x per al contorn esquerra. L'altre sortida *nx* és un enter amb el nombre de divisions verticals (nombre d'estructures X-order) en que s'ha dividit el xip estructures en *x*.

- **createYorder.m:** function [Y] = createYorder(y0, name, place, rows)

CREATEYORDER crea *Y*, una estructura *Y-order* per a un element *bloc/stripe/empty space*. *y0* és la coordenada y del cantó de a baix a l'esquerra d'aquest element. *Name* és el nom de l'element ('b' per a un bloc, 'V' per a un stripe vertical, 'H' per a un stripe horitzontal i 0 per a un *empty space*). *Place* és el número que té assignat l'element en l'estructura de representació del xip. *Rows* és el nombre de files que té aquest element.

Funcions consultores

- **giveYorder.m:** function [y] = giveYorder(x, k)

GIVEYORDER retorna *y*, una estructura *Y-Order* que defineix el *k*-èssim element que comparteix la divisió vertical definida a la *X-Order* estructura *x*.

- **numy.m:** function [ny] = numy(x)

NUMY retorna el nombre d'elements que hi ha en una mateixa divisió vertical definida en *x* (*X-Order*).

- **givePlace.m:** function [place] = givePlace(yorder)

GIVEPLACE retorna *place,* un enter que indica el lloc que ocupa l'element definit a *yorder* (*Y-Order* estructura) dins de la representació de xip *chip*.

- **name_y.m:** function [name] = name_y(y)

NAME_Y retorna el tipus d'element que és *y* (*y-order*). Els possibles noms són 'b' per a un bloc, 'H' per a un horitzontal stripe, 'V' per a un vertical stripe, o '0' per a un *Empty space*.

Estructura de dades *Matrix block*

Matrix block conté tota la informació necessària per descriure i calcular el comportament dinàmic d'un bloc a una freqüència. La configuració de l'estructura de dades es mostra a continuació:

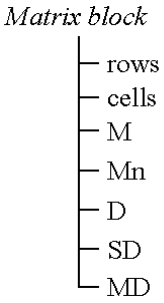

Figura A.2.10: Estructura de dades *Matrix block*.

El significat dels camps és:

- rows: Nombre de files que té el bloc.
- cells: Nombre de cel·les a cada fila. Nombre de columnes del bloc.
- M: *M(i,j,k)* matriu de tres dimensions amb la matriu de transferència **M** de la columna *k* a la columna *k+1*.
- Mn: Matriu de transferència entre contorns del bloc (del contorn dret al contorn esquerra).

- D: *D(i,j,k)* matriu de tres dimensions que conté la matriu que transfereix el corrent de commutació de les cel·les entre les columnes *k* i *k+1* a les variables de la columna *k+1*.
- SD: *SD(i,k)* matriu amb els vectors de termes independents de la funció de transferència entre la columna *k* i la columna *k+1*. Conté el resultat de $\mathbf{SD}_k \cdot \mathbf{i}_{swk}$.
- MD: Vector amb el terme independent de la funció de transferència entre el contorn esquerra i el contorn dret del bloc.

Funcions constructores:

- **createMblock.m:** function [MM] = createMblock(B,stim,fi)

CREATEMBLOCK crea una estructura de dades *matrix block* en *MM* amb la representació matricial del *block B* per a l'estímul *stim*, a la freqüència definida per l'índex *fi*.

Funcions consultores:

- **giveMDmatrix.m:** function [M, SD] = giveMDmatrix(B)

GIVEMDMATRIX retorna les matrius **M***(i,j,k)* i **SD**(*j*,*k*) (**SD=D·i**$_{sw}$) del la estructura *block matrix B*. Per cada *k* **M** i **SD** contenen la relació entre les columnes *k* i *k+1:* $\begin{bmatrix}\mathbf{V}\\ \mathbf{I}\end{bmatrix}_{k+1} = \mathbf{M}_k \cdot \begin{bmatrix}\mathbf{V}\\ \mathbf{I}\end{bmatrix}_k + \mathbf{SD}_k$.

Estructures de dades *Matrix Vstripe* i *matrix Hstripe*

Matrix Vstripe i *Matrix Hstripe* contenen tota la informació necessària per descriure i calcular el comportament dinàmic dels stripes verticals i horitzontals a una determinada freqüència. La configuració de la estructura de dades s'organitza com segueix:

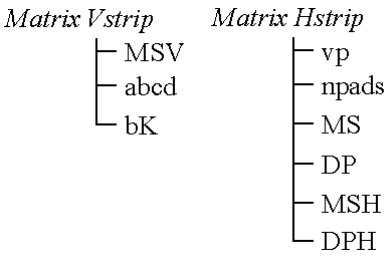

Figura A.2.11. Estructures de dades *Matrix Vstrip* i *Matrix Hstripe*.

El significat dels camps per *Matrix Vstripe* és:

- MSV: Matriu de transferència entre el contorn esquerra i el contorn dret del stripe vertical.
- abcd: Matriu 2x2 amb els valors de la línia de transmissió.
- bK: Matriu **ΔK**.

Per *Matrix Hstripe*:

- vp: Voltatge de la font d'alimentació a la que estan connectats els pads.
- npads: Nombre de pads.
- MS: *MS(i,j,k)* matriu tridimensional amb les matrius de transferència de la columna *k* a la columna *k+1*.
- DP: *DP(i,k)* matriu composada de vectors columna amb els termes independents de la funció de transferència entre la columna *k* i la columna *k+1*.
- MSH: Matriu de transferència entre els contorns esquerra i dret de l'stripe.
- DPH: Vector amb el terme independent de la funció de transferència entre els contorns esquerra i dret de l'stripe.

Funcions constructores:

- **createMstripeV.m:** function [M] = createMstripV(S, f)

CREATEMSTRIPEV crea una estructura *matrix Vstripe* a *M* per a la freqüència *f,* a partir del *stripe S*.

- **createMstripeH.m:** function [M] = createMstripH(S, f)

CREATEMSTRIPEH crea una estructura *matrix Hstripe* a *M* per a la freqüència *f,* a partir de l'*stripe S*.

Funcions consultores:

- **giveSHmatrices.m:** function [MS, DP, vp] = giveSHmatrices(MH)

GIVESHMATRICES retorna les matrius **MS** i **DP** i el voltatge de la font que alimenta al pads (*vp*) de l'stripe horitzontal representat per la *matrix stripeH* estructura *MH*.

- **giveSVmatrices.m:** function [Mabcd, bK] = giveSVmatrices(MV)

GIVESVMATRICES retorna les matrius *Mabcd* i *bK* que representen el comportament del stripe vertical descrit a MV (*matrix stripeV*).

A.2.4. Estructura de dades Boundary

L'estructura de dades *Boundary* té missió de guardar la informació referida als contorns dels xip per a una determinada freqüència. Aquesta informació versa sobre quins voltatges o quins corrents són coneguts i quins no ho són.

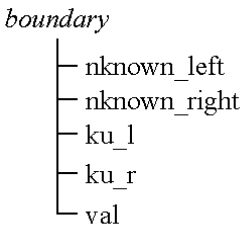

Figura A.2.12. Estructura de dades *Boundary*.

El significat dels camps és:

- nknown_left i nknown_right: Enters amb el número de variables conegudes en el contorn esquerra i en el contorn dret.

- ku_l i ku_r: Vectors amb un 1 o un 0 depenent si la variable de contorn es coneguda o no per als contorns esquerra i dret. La primera meitat del vector conté les variables de voltatges i la segona meitat la de corrents.

- val: Vector que té el valor del voltatges i corrents coneguts per als contorns esquerra i dret. Està ordenat primer per les variables del contorn esquerra seguit de les variables del contorn dret. Les variables estan ordenades primer voltatges i després corrents.

Funcions constructores:

- **giveBoundaries.m:** function [bon] = giveBoundaries(c, fi)

GIVEBOUNDARIES analitza el la estructura *chip c* i retorna la informació referent als contorns en una estructura *boundary* per a la freqüència indicada per l'índex *fi*.

Funcions consultores:

- **give_values.m:** function [vals] = give_values(bound)

GIVE_VALUES retorna els valors de les variables de contorn del xip conegudes en el vector *vals*. La entrada *bound* és una estructura *boundary*

- **give_ku.m:** function [ku_l, ku_r] = give_ku(bound)

GIVE_KU retorna els vectors amb la informació de variable coneguda o desconeguda per als contorns esquerra o dret del xip (*ku_l*, *ku_r*). La entrada *bound* és una estructura *boundary*.

A.2.5. Estructures de dades Frequency results

Per a cada freqüència els resultats de voltatges i corrents de cada cel·la del xip és guarden segons component al que pertanyen (bloc o stripe) en una estructura de dades anomenada *Frequency Chip*. Cada component té associat una estructura de dades *Frequency Results* amb dos camps, *v* i *i*, matrius amb les tensions i corrents de les seves cel·les.

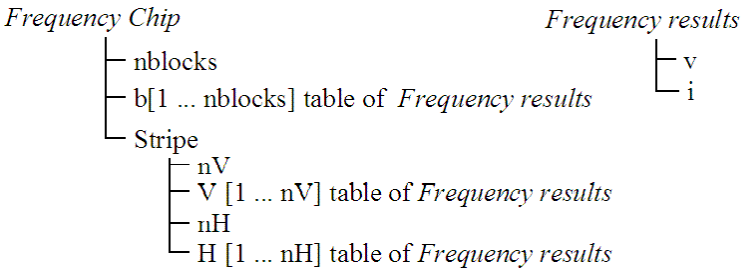

Figura A.2.13. Estructures de dades *Frequency Chip* i *Frequency results*.

El significat dels camps de la estructura *Frequency Chip* són:

- nblocks: Nombre de blocs del xip que tenen resultats freqüencials.
- b: Vector de *Frequency results* amb els valors de voltatge i corrent dels blocs del xip per a una freqüència.
- Stripe.nV: Nombre de vertical stripes.
- Stripe.V: Vector de *Frequency results* amb els resultats de voltatges i corrents per als stripes verticals a una freqüència.
- Stripe.nH: Nombre de stripes horitzontals.
- Stripe.H: Vector de *Frequency results* amb els resultats de voltatges i corrents per als stripes horitzontals a una freqüència.

Funcions constructores:

- **SolveChip.m:** function [S] = SolveChip(CM, bound)

SOLVECHIP soluciona el xip descrit per CM (*chip matrix*) amb per a les condicions de contorn definides a *bound* (*boundary*). Com a resultat retorna *S*, un vector d'estructures de dades *frequency results* amb la solució als contorns de tots els components del xip. La substructura *x_order* de *CM* organitza el vector *S* segons divisions verticals *x*.

- **getResults.m:** function [R] = getResults (sol, CM)

GETRESULTS a partir de las solució del xip *sol* (vector de *frequency results* seguint l'ordre descrit a la sub-estructura *x_order* de *CM*) i de la descripció matricial del xip CM (*Chip matrix*) calcula tots els voltatges i corrents de dins dels elements del xip. Com a resultat retorna *R* una estructura de dades *Frequency chip* amb la solució del xip per a la freqüència de construcció de *CM*.

Funcions consultores:

- **giveV.m:** function [v] = giveV(RF, type, name, fi)

GIVEV a partir de RF (estructura de dades *frequency chip* amb la solució del xip) a la freqüència que l'índex *fi* retorna una matriu *v(r,c)* amb els nivell de tensió de totes les files *r* i columnes *c* del component definit per *type* amb nom *name*. *Type* és 'b' , 'H' o 'V' depenent que el component requerit sigui un bloc, horitzontal stripe o vertical stripe. *Name* és l'enter que defineix el nom del component.

- **giveI.m:** function [i] = giveI(RF, type, name, fi)

GIVEI a partir de RF (estructura de dades *frequency chip* amb la solució del xip) a la freqüència que l'índex *fi* retorna una matriu *i(r,c)* amb els llaços de corrent de totes les files *r* i columnes *c* del component definit per *type* amb nom *name*. *Type* és 'b' , 'H' o 'V' depenent que el component requerit segui un bloc, horitzontal stripe o vertical stripe. *Name* és l'enter que defineix el nom del component.

A.2.6. Estructura de dades de definició de l'output

Les dades de sortida són els resultats en el domini del temps de totes les cel·les del xip. Aquest informació està organitzada dins de la estructura de dades *Chip Time*. Els valors de cada component del xip es guarden en l'estructura de dades *Time results*. *Time results* guarda la evolució dels nivells de tensió i corrents de cada cel·la per a cada component del xip en les matrius tridimensionals *v* i *i*. Una cel·la d'un component (stripe o bloc) situat a la fila *r*, columna *c* a l'índex de temps *t* presenta un nivell de tensió $v(r,c,t)$ i corrent $i(r,c,t)$.

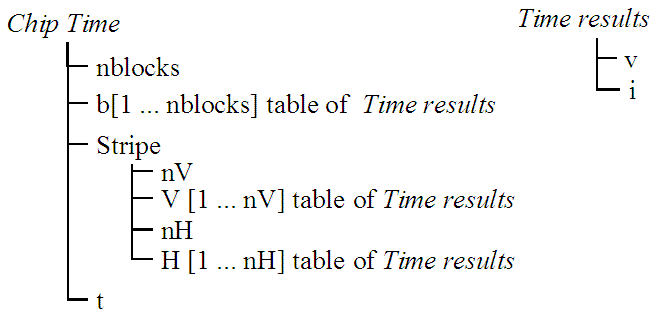

Figura A.2.14. Estructures de dades *Chip Time* i *Time results*.

El significat dels camps de *Chip Time* són:

- nblocks: Nombre de blocs.
- b: Vector de *Time results* amb la evolució dels nivells de tensió i corrents dels blocs.
- Stripe.nV: Nombre de stripes verticals.
- Stripe.V: Vector de *Time results* amb la evolució dels nivells de tensió i corrents dels stripes verticals.
- Stripe.nH: Nombre de stripes horitzontals.
- Stripe.H: Vector de *Time results* amb l'evolució dels nivells de tensió i corrents dels stripes horitzontals.
- t: Vector amb els valor de l'eix de temps.

Funció constructora:

- **convertToTime.m:** function [T] = convertToTime(R, stim)

CONVERTOTIME calcula els resultats en el domini del temps a partir dels resultats en el domini de la freqüència i els guarda a *T* estructura de dades *chip time*. *R* és un vector d'estructures *frequency chip*, amb els resultats de totes les freqüències d'interès. *stim* és la estructura de dades amb l'estímul aplicat que conté la informació de les freqüències usades.

Funcions consultores:

- **giveResultsTime.m:** function [v, i] = giveResultsTime(R, c, k)

GIVERESULTSTIME retorna els resultats de *R*, *chip time*, per al *k*-èssim bloc/stripe de *R*. *c* pot ser 'b', 'V', o 'H' depenent de si els resultats que requerits són per a un bloc, o un stripe vertical o horitzontal. *k* és el nom del component. Les sortides *v*, *i* són matrius tridimensionals amb *v(i,j,t)* (o *i(i,j,t)*) el valor del nivell de tensió per a la fila *i*, columna *j* a l'índex de temps *t*.

- **giveTimeAxis.m:** function [t] = giveTimeAxis(RT)

GIVETIMEAXIS retorna el vector amb l'eix de temps. L'entrada *RT* és una estructura de dades *chip time* amb la solució del xip en el domini del temps.

Funcions associades:

- **PlotResults.m:** function [ ] = PlotResults(R)

PLOTRESULTS dibuixa els gràfics de nivells de tensió i corrents per a tots els stripes i blocs VS temps. En el gràfic de bloc dibuixa els nivells de tensió i corrent de totes les cel·les de una fila i una columna ambdues definides dins de la funció. *R* és una estructura *chip time* amb els resultats en el domini del temps.

A.2.7. Funcions consultores universals

Hem classificat les funcions com a constructores o consultores per a una determinada estructura de dades. Existeixen, però, un altre estil de consultores que donen informació sobre algun camp comú a varies estructures de dades, aquestes funcions són:

- **numStripes.m:** function [n] = numStripes(c, type)

NUMSTRIPES retorna el nombre d'stripes de *c* que són de tipus *type*. *Type* pot ser 'H' o 'V' depenent de la orientació horitzontal o vertical de l'stripe. La estructura *c* pot ser un C*hip*, un C*hip matrix*, un *Frequency Chip* o un *Chip Time*.

- **numBlocks.m:** function [n] = numBlocks(c)

NUMBLOCKS retorna el nombre de blocs que té la estructura de dades *c*. *c* pot ser C*hip*, C*hip matrix*, *Frequency Chip* o *Chip Time*.

- **numCells.m:** function [n] = numCells(c)

NUMCELLS retorna el nombre de cel·les que té la estructura de dades *c*. *c* pot ser *Block, Stripe* o *Matrix Block*.

- **numrows.m:** function [n] = numrows(c)

NUMROWS retorna el nombre de files que té la estructura de dades *c*. *c* pot ser *Chip, Block, Stripe* o *Matrix block.*

- **posX.m:** function [x] = posX(a)

POSX retorna la posició x (en unitats de columna) que té el component representat per l'estructura de dades *c* (*Chip*, *Block* o *Stripe*).

- **posY.m:** function [y] = posY(a)

POSY retorna la posició y (en unitats de fila) que té el component representat per l'estructura de dades *c* (*Chip*, *Block* o *Stripe*).

A.3. Funcions

En aquest apartat es presenten totes les funcions desenvolupades a partir de les comandes i funcions pròpies de matlab [1]. Donat la gran quantitat de funcions, aquestes s'han estructurat segons el tipus de tasca amb la que van relacionades, aquestes s'han dividit en:

- Funcions de definició d'input: Totes les funcions relacionades amb la definició dels inputs.
- Funcions principals: Totes les funcions que participen en la realització dels càlculs.
- Funcions consultores: Totes les que la seva única tasca es retornar informació d'una estructura de dades.
- Altres funcions: Totes les funcions auxiliars que no es poden classificar amb les anteriors etiquetes.

Cada funció s'introdueix amb fitxer en el que està implementada (fitxer .m) seguit del codi original. Les subfuncions es presenten a continuació de les funcions a les que complementen. El símbol % indica comentari.

A.3.1. Funcions de definició d'Input

Les següents functions tenen l'objectiu de configurar les dades d'entrada, és a dir, la definció de xip i la configuració del estímuls.

- getInputs.m:

```
function [ c ] = getInputs
%Definition of chip. Creation of chip data structure .


%example with 1 block of 10 rows and 200um length (107 ref. gates), with ring and  2 pads of 0.6nH
of bonding wire placed in the vertical stripes.

x0=0; % x reference of coordinates (in column units)
y0=0; % y reference of coordinates (in cell units)
```

```
c=iniChip(0);  %inicialize the chip (put to 0 fields)

nblocks=1;    %number of blocks
activity=0.2;

%CREATION OF BLOCKS
for ii=1:nblocks
   %block construction
   rows=10;    %number of rows
   fu=1.187;   %utilization factor of standar cells ocupation (from philips data, yes is upper than one)
   length=50*4;    %[um] length of the rows
   b=createBlockLength(rows,length,fu); %b is a block structure

   %cells construction
   %parameters
   tracks=getTrackParam(length,numCells(b));
   well=getWell(length,numCells(b));
   interc=getInterconnect('philips');
   gate=getGate('reference');
   %switching cells
   i=0; %activity counter
   swtype=0;
   a=1;    %type choose variable
   for rb=1:rows
      for cb=1:numCells(b)
         %actitvity creation

         switch floor(i*activity)*a
            case  1
               swtype=1;   %switching cell type 1
               i=0;
               a=2;
            case  2
                swtype=2;   %switching cell type 1
               i=0;
               a=1;
         end

         i=i+1;
```

```
            cell=createCell(tracks,well,interc,gate,swtype);
            b=addCell(b,cell,rb,cb);
            swtype=0; %non switching
        end
    end

%       cell=createCell(tracks,well,interc,gate,1);
%       b=addCell(b,cell,5,105);


    %position of the block in the chip
    y=y0;     %block y down coordinate (in rows units)
    if ii==1
        x=x0;    %block x left coordinate (in columns units)
    else
        x=posX(giveBlock(c,ii-1))+numCells(giveBlock(c,ii-1))+1;      %+1 because the vertical strip
between two blocks
    end
    c=addBlock(c,b,x,y); %add the block b in x,y coordinates inside chip
    clear('b')
end

%CREATION OF STRIPES
nV=2;   %number of vertical stripes
nH=2;   %number of horizontal stripe

%pad properties
Rpad=0*0.01;
Lpad=0.6e-9;

%VERTICAL STRIPES
rows=numrows(c)+2; %because i will put two horizontal stripes more (two rows more)
y=y-1; %y coordinate of the stripes down (in rows units)
%vertical stripes creation

Props=getStripVProps(0); %Properties are resitance, inductance, capacitance and reactance of the
stripes
SV=createStripe(Props,rows,'V');  %creation of a vertical 'V' stripe
```

```
SV2=createStripe(Props,rows,'V');

P=makePADV(2,Lpad,Rpad,'l'); %pad in row 6, in 'l' left side, with Lpad, Rpad properties
P2=makePADV(11,Lpad,Rpad,'r'); %pad in row 7, in 'r' right side, with Lpad, Rpad properties

SV=addPAD(SV,P);    %add pad to the stripe
SV2=addPAD(SV2,P2);

%x coordinate of the stripes (in column units)
xr=posX(giveBlock(c,1))+numCells(giveBlock(c,1));   %right to the block 1
xl=posX(giveBlock(c,1))-1;      %left to the block 1

c=addStripe(c,SV,xl,y,'V');  %add the vertical 'V' stripe SV in x,y coordinates inside chip c
c=addStripe(c,SV2,xr,y,'V');
clear('SV','P','P2');

b=giveBlock(c,1); %b is the block 1 in chip c
%HORIZONTAL STRIPES

Props=getStripHProps(0); %Properties are resitance, inductance, capacitance and reactance of the
stripes
columns=numCells(b);    %columns of the stripe
SH=createStripe(Props,columns,'H');  %creation of a horizontal 'H' stripe
SH2=createStripe(Props,columns,'H');

x=posX(b);    %x coordinate of the stripes left (in column units)
y=posY(b)-1;   %horizontal stripe below the block
c=addStripe(c,SH,x,y,'H');  %add the vertical 'V' stripe SV in x,y coordinates inside chip c

y=posY(b)+numrows(b); %horizontal stripe above the block
c=addStripe(c,SH2,x,y,'H');



%-------------------------------------------------------------------------------------------------------------
function [ T ] = getTrackParam(length,cells)
%give the tracks properties  in a RLC structure
%length is in [um]
w=1.2;  %[um] widht
```

```
T.R=1/w*0.095*length/cells; %[ohm/cell]  0.095=sheet resistance ohm/sq w=1.2 widht
T.L=6.713e-15/w*length/cells;  %[H/cell] 6.713e-15='sheet inductance' H/sq for 200 rows of 1.2um of wihdt, 4um is the pitch between vdd/gnd tracks;
T.C=0;
%-----------------------------------------------------------------------------------------------------------
function [ W ] = getWell(length,cells)

W.C=0.8e-15*length/cells; %[F/1.12um]
W.R=177/length*cells;    %[ohm 1.12 um]
W.L=0;
%-----------------------------------------------------------------------------------------------------------
function [ I ] = getInterconnect(s)
%interconection
switch s
   case 'philips'
      Lav=35.2;%10   %[um]Lenght of interconnects that go out from a equivalent cell output
   case 'nothing'
      Lav=0;
end
I.C=10.5e-15/35.2*Lav;    %[fF] Philips data says average interconnect capacitance is 10.5fF for 35.5um so i correct for another average interconnect length Lav
I.R=0.514*Lav;   % 18.91;       [ohm/um]*[um] L
I.L=0;   %164.1e-12;         %80.41e-12*Lav/75;

%-----------------------------------------------------------------------------------------------------------
function [ C ] = getGate(s)
% put in cell C the a series branch between vdd and gnd with gate resistance, capacitance and inductance
switch s
   case 'reference'
      C.R=1/(0.0021185); %[ohm/standard cell]
      C.C=3.3e-15;    %[F/standard cell]
      C.L=0;
   case 'decap'
      C.R=1;
      C.C=100e-15;
      C.L=0;
end
%-----------------------------------------------------------------------------------------------------
function [ S ] = getStripHProps(l)
```

```
%Give the R, L, G, C parameters that define the stripe.
%a piece of strip of 1.12 um length has.... w=12um
length=1.12;   %[um]
w=12;   %[um]
S.R=length*70e-3/w; %[ohm]
S.L=0.4530e-9*length*1e-3;    %[H]  (auto inductance of 2 conductors of 12um width and 330um
thickness separated 2um  0.453 is nH/mm
S.C=length*4.965e-15;  %[F]
S.G=0;

%-------------------------------------------------------------------------------------------------
function [ S ] = getStripVProps(l)
%Give the R, L, G, C parameters that define the stripe.
%a piece of strip of 4um (8um from vdd to next vdd) length has... w=12um;
%0.453 is nH/mm
length=4;   %[um] from row to row
w=12;   %[um] width

S.R=length*70e-3/w; %[ohm]  (70 mOhm/sq)
S.L=0.4530e-9*length*1e-3;   %[H]  (auto inductance of 2 conductors of 12um width and 330um
thickness separated 2um  0.453 is nH/mm
S.G=0;  %[ohm]
S.C=length*4.965e-15;  %[F] c=4.965fF/um

%-------------------------------------------------------------------------------------------------
```

- getStim.m:

```
function [ stim ] = getstim
%GETSW will create the types of switching currents in frequency domain and will
%store all frequencial data in the stim struct

%Example of two stim of a step defined by t1, t2 and Tend.
%                              ___________
%          ____________________|           |_____________
%          0                   t1          t2            Tend
%Simulation time 1ns.
Tend=1e-9;     %[s]
Ts=10e-12;     %[s] Sample time
```

```
stim=createStim(Tend,Ts);   %create the struct stim of Tend lenght with a sample of Ts



%creation of a step in frequency domain
type=1;     %is associated to a kind of switching cell.

t1=250e-12; %[s]
t2=350e-12; %[s]
sw1=gen_step(t1,t2,Ts,Tend); %sw1 has the frequencial components of the step. F is the frequency
axis
%sw1 is a vector row

stim=addType(stim,type,sw1);     %add the current description in frequency domain to for a type of
cell.

%creation of another step delayed in time
type=2;     %is associated to a kind of switching cell.
t1=300e-12; %[s]
t2=400e-12; %[s]
sw2=gen_step(t1,t2,Ts,Tend); %fd1 has the frequency components of the step. F is the frequency axis

stim=addType(stim,type,sw2);     %add the current description in frequency domain to for a type of
cell.

%-------------------------------------------------------------------------------------------------
```

- addBlock.m

```
function [ c ] = addBlock(c,b,x,y )
%ADDBLOCK adds a b block, into chip c in x,y coordintaes (down left of the
%block)

c.nblocks=c.nblocks+1; %number of blocks
b.pos.x=x;  %block left x position  (in cell units)
b.pos.y=y;  %blocy down y position (in row units)
c.b(c.nblocks)=b;
```

```
%upgrading the general data of the chip
%x position of chip in column units (more left there is chip)
if x < c.pos.x
   c.pos.x=x;
end
%y position of chip in row units (more down there is chip)
if y < c.pos.y
   c.pos.y=y;
end

%total number of rows of the chip
if c.rows < y+numrows(b)-c.pos.y
   c.rows=y+numrows(b)-c.pos.y;
end

%-------------------------------------------------------------------------------------------
```

- addCell.m:

```
function [ b ] = addCell( b,cell,r,c )
%ADDCELL Adds a cell in column c row r of block b

b.cell(r,c)=cell;

%-------------------------------------------------------------------------------------------
```

- addPAD.m:

```
function [ s ] = addPAD( s, pad )
%ADDPAD adds a pad to a stripe structure (vertical or horizontal)
%  s is a stripe structure, pad is a pad structure,

s.npads=s.npads+1; %upgrade of number of pads
s.PAD(s.npads)=pad;

%-------------------------------------------------------------------------------------------
```

- addStripe.m

```
function [ c ] = addStripe( c,S,x,y,type )
%ADDSTRIPE adds a stripe S, into chip c in x, y coordinates (down left of the
%stripe).
%  Type shows if is a vertical 'V' stripe or horizontal 'H' stripe.

if type=='V'
    c.Stripe.nV=c.Stripe.nV+1; %one more vertical stripe
    S.pos.x=x; %stripe left x position  (in column units)
    S.pos.y=y; %Stripe left y position  (in row units)

    c.Stripe.V(c.Stripe.nV)=S; %addition of the stripe

else
    c.Stripe.nH=c.Stripe.nH+1; %one more vertical stripe
    S.pos.x=x; %stripe left x position  (in column units)
    S.pos.y=y; %Stripe left y position  (in row units)

    c.Stripe.H(c.Stripe.nH)=S;  %addition of the stripe
end

%upgrading the general data of the chip
%x position of chip in column units (more left there is chip)
if x < c.pos.x
    c.pos.x=x;
end
%y position of chip in row units (more down there is chip)
if y < c.pos.y
    c.pos.y=y;
end

%total number of rows of the chip
if c.rows < y+numrows(S)-c.pos.y
    c.rows=y+numrows(S)-c.pos.y;
end

%-------------------------------------------------------------------------------------------
```

- addType.m:

```
function [ stim ] = addType( stim, type, fwav)
%ADDTYPE adds another type to the stim structure
%  stim is a structure that has all the frequencial information of current
%  stimulis. Type is the type of waveform. fwav is the current waveform in
%  frequency domain.

stim.type(:,type)=fwav;

%-------------------------------------------------------------------------------------------
```

- createBlockCell.m:

```
function [ block ] = createBlockCell( rows,cells,fu )
%CREATEBLOCKCELL create the fields rows, length, fu and cells of a block
%struct, and inicialite the other fields.

%   The block is defined by rows of reference gates, all rows have the same number of cells. There
are two ways of define a row, by the cells inside or by length.
%   fu mean utilization factor, the proportion of length of row that has a reference gate (not only tracks)
inside
%   length is the length of the rows in [um]
RG=1.12;   %[um] of a reference gate

block.rows=rows;  %rows of this block
block.fu=fu;   %utilization factor of standard cells occupation
block.cells=cells;  %equivalent cells per row

block.length=cells*RG*fu;  %length of the block

%initializations
block.pos.x=0;
block.pos.y=0;

end

%-------------------------------------------------------------------------------------------
```

- createBlockLenght.m:

```
function [ block ] = createBlockLength( rows,length,fu )
%CREATEBLOCKLENGTH put the fields rows, length, fu and cells of a block
%struct, and initialize the other fields.

%   The block is defined by rows of reference gates, all rows have the same number of cells. There
are two ways of define a row, by the cells inside or by length.
%   fu mean utilization factor, the proportion of length of row that has a reference gate (not only tracks)
inside
%   length is the length of the rows in [um]
RG=1.12;   %[um] of a reference gate

block.rows=rows;  %rows of this block
block.fu=fu;   %utilization factor of standard cells occupation
block.length=length;  %length of the block
block.cells=floor(length*fu/RG);  %equivalent cells per row

%inicializations
block.pos.x=0;
block.pos.y=0;

%-----------------------------------------------------------------------------------------------
```

- createCell.m:

```
function [ cell ] = createCell( track,well,interc,gate,swtype )
%CREATECELL generate a cell stuct. Cell struct has two fields, param with the RLGC model
parameters defined in the fields track, well,
%interc and cell,  and swtype (the type of switch 0=no switch)

%cell RLGC parameters
cell.Param.track=track;
cell.Param.well=well;
cell.Param.interc=interc;
cell.Param.gate=gate;

%cell activity type
```

```
cell.swtype=swtype;

%-------------------------------------------------------------------------------------------
```

- createStim.m:

```
function [ stim ] = createStim( Tend,Ts )
%CREATESTIM creates and initializes a stim struct,
%  Tend is the total time of the stimulus, Ts is the time resolution

stim.Tend=Tend; %total stim time
stim.Ts=Ts; %sample time
stim.fs=1/Tend; %frequency step
stim.nf=floor(Tend/Ts)+1;   %number of points (+1 because 0)
stim.n2=floor(Tend/Ts/2)+1;   %number of frequency points (only positive and 0 frequences)
stim.F=[0:1/Tend:1/2/Ts];    %frequency range axis (only positive and 0)

%inicialitation of types of stims
%type is a matrix of frequency components of a switching current. Each column is a type of waveform,
and row is the frequency. Row 1 is the continuous component.
stim.type(:,1)=zeros(stim.n2,1);  %a  column vector with all 0 for all frequencies (positives including 0)


%-------------------------------------------------------------------------------------------
```

- createStripe.m:

```
function [ S ] = createStripe( props, size, type )
%CREATESTRIPE put the fields props, cols and rows to the stripe struct and
%inicialize the other fields.
%   props is a structure with the properties (R L G C) of stripe
%   size gives the number of rows in the Vertical case or the columns in
%   the Horizontal case
%   Type is a char 'V' or 'H' if is vertical or horizontal

S.props=props;
if type=='V'
    S.rows=size;
```

```
   S.cells=1;
   %pad inicialization
   P=makePADV(0,0,0,0);
   S.PAD=P;
else
   S.rows=1;
   S.cells=size;
   %pad inicialization
   P=makePADH(0,0,0);
   S.PAD=P;
end

%inicialization
S.npads=0;  %number of pads in the stripe

%-----------------------------------------------------------------------------------------------------
```

- gen_step.m:

```
function [ fw2, F] = gen_step(t1,t2,Ts,Tend)
%generate step inputs
%   fw2 is a vector column with the frequency components of a step defined
%   by t1, t2, Ts, and Tend. Ts is time resolution
%                                  ____________
%          ______________________|              |______________
%         0                        t1            t2              Tend

I=18.95e-5; %[A] %amplitude of step

a=zeros(1,floor(t1/Ts));  %+1 because time=0
b=ones(1,floor(t2/Ts+1)-floor(t1/Ts));
c=zeros(1,floor(Tend/Ts)-floor(t2/Ts));
v=cat(2,a,b*I,c);
time=[0:Ts:Tend];
F=[0:1/Tend:1/2/Ts];    %row vector
n=size(v); %number of  frequency components (positive, negative and 0)
%size function give the dimensions (#rows,#columns)
n=n(2); %total number of frequency points
n2=floor(n/2)+1; %number of positive and zero frequency points
```

```
fw1=fft(v')./(n);    %n(2) is number of points put in the fft;
%the fft has to be divided by the number of points.
fw2=fw1(1:n2,1);

%-------------------------------------------------------------------------------------------
```

- iniChip.m

```
function [ c ] = iniChip( param )
%INICHIP create an empty chip. Result is a chip structure with chip fields initialized to 0.
%  param is not used parameter (is necessary an input for matlabV6.5)

%left down coordinates
c.pos.x=0;  %in column units
c.pos.y=0;  %in row units
%total number of rows
c.rows=0;
%ini blocks
c.nblocks=0;
%ini stripes
c.Stripe.nV=0;
c.Stripe.nH=0;

%-------------------------------------------------------------------------------------------
```

- makePadH.m:

```
function [P] = makePadH(x,L,R)
%MAKEPADH create a structure pad for a horizontal stripe

%x is the position (row units), L and R are pad properties
P.pos.place=x;
P.pos.side=0;
P.L=L; %[H]
P.R=R; %[ohm]

%-------------------------------------------------------------------------------------------
```

- makePadV.m:

```
function [P] = makePadV(x,L,R,c)
%MAKEPADV create a structure pad for a vertical stripe

%x is the position (row units), L and R are pad properties, c is a char
%meaning the side where the pad will be placed 'r' or 'l'
P.pos.place=x;
P.pos.side=c;
P.L=L; % [H]
P.R=R; %[ohm]

%-----------------------------------------------------------------------------------------------
```

A.3.2. Funcions principals

Les següents funcions són les encarregades de realitzar tot el procés de càlculs que condueix a l'estimació del soroll.

- Av_chip.m:

```
%Main function, controles the calculation algoritm
clear;  %erase all variables before begin
t=cputime;

%INPUTS
chip=getInputs;  %things needed about chip structure ,the position and length of blocks, stripes and pads
stimuli=getstim;  %create currents waveforms and give the frequency data (frequency range).

%frequency points of stimuli
n2=give_n2(stimuli);   %n2 is half0 number of frequency points that the current waveforms have

%frequency solution
for fi=1:n2 %half + 0 frequencies

   CM=BuildChip(chip,stimuli,fi);    %result is a estruct of representation of the response of the %chip for this frequency, that is, all the block and strip matrices

   boundaries=giveBoundaries(chip,fi);  %give the boundary conditions in boundaries struct

   solution=SolveChip(CM,boundaries);      %result is all boundary conditions in all blocks %inside the chip for this frequency
   results=getResults(solution,CM);
   RF(fi)=results;    %solution in frequency domain for the frequency index fi

end
tsolve=(cputime-t) %seconds to solve
RT=convertToTime(RF,stimuli);
tftt=(cputime-t-tsolve) %seconds to convert to time
PlotResults(RT);
tall=(cputime-t)  %seconds to do all
```

```
%-------------------------------------------------------------------------------------------
```

- BuildChip.m:

```
function [ M ] = BuildChip( c, stim, fi )
%BuildChip Build all the matrix description of the chip for the frequency
%of the index fi
%   c is a chip representation. stim has the frequency information. M is a struct with all matrix
representation

f=give_freq(stim,fi);   %frequency of analysis [Hz]
%PARTIAL MATRICES. For the substructures of chip (as block or stripes).
%BLOCK
%the matrices are M, D, and MD(acumulation of the switwing part)
nB=numBlocks(c);
for ii=1:nB
    M.b(ii)=createMblock(giveBlock(c,ii),stim,fi);
end

%STRIPES
%creation of strip vertical matrices
nV=numStripes(c,'V');
for ii=1:nV
   M.Stripe.V(ii)=createMstripV(giveStripe(c,'V',ii),f);
end

%creation of strip horitzontal matrices
nH=numStripes(c,'H');
for ii=1:nH
   M.Stripe.H(ii)=createMstripH(giveStripe(c,'H',ii),f);
end
M.vp=obtainVpad(fi,'index');

%GLOBAL MATRICES. Representation of chip
%chip organization following x coordinate, from left to right
[x, nx]=sort_x(c);    % find all positions where block or strip begin and sort from left to right.
% x is a structures

%put the x, and nx fields in the M structure. X
```

```
M.x=x;  %it has the information of how is sort the blocks and stripes
M.nx=nx;    %number of different x coordinates where a stripe/block begins


for k=1:nx
%    k
 %ordering information
  ny=numy(x(k));   %number of horizontal structures in the vertical division
  y=giveYorder(x(k),1);   %first structure definition
  [AAy, BBy]=giveMatrix(y,M);    %matrices for the structure with (x(k),y) coordinates


   if ny>1
      for ii=2:ny
         y=giveYorder(x(k),ii);
         [A, B]=giveMatrix(y,M);   %Give the matrix A and vector B that [v;i]xy+1=Ax*[v;i]x+Bxy for the
structure (block/stripe/emptyspace) defined in y
         AAy=makebig(AAy,A);
         BBy=makebigB(BBy,B); % [BBy;B]
      end
   end
   % store the partial matrices
    Ax(:,:,k)=AAy;     %   [v;i]x+1=Ax*[v;i]x+Bx
    Bx(:,k)=BBy;
end


%build the rest of struct fields
%   [v;i]x+1=Ax*[v;i]x+Bx
M.Ax=Ax;    % A matrix from x to next x
M.Bx=Bx;    % B vector from x to next x
M.nblocks=nB;    %number of blocks in the chip
M.Stripe.nV=nV;    %number of vertical stripes in the chip
M.Stripe.nH=nH;    %number of horizontal stripes in the chip
M.rows=numrows(c);  %maximum number of rows in a chip column
M.y0=posY(c);   %reference of row origin


%--------------------------------------
function [AB2] = makebig(AA,A)
%creation of big matrix;
%   AA and A are two matrices of blocks/stripe/empty space structures
%   The structure that AA represent is above the structure A represent,
%   AB2 is the matrix of the full structure, relation [v;i] with [v;i] in
```

```
%   the right

AB=blkdiag(AA,A);   %block diagonal matrix
nA=size(AA);    %dimensions of matrix AA
nB=size(A);     %dimensions of matrix A
AB2=changeBase(AB,nA(1),nB(1)); %put ok the base, that is firt all voltages, then currents



%---------------------------------------
function [BB2] = makebigB(BB,B)
%creation of big matrix
%   BB and B are two vectors (independent term) of blocks/stripe/empty space structures
%   The structure that BB represent is above the structure B represent,
%   BB2 is the matrix of the full structure, relation [v;i] with [v;i] in  the right

BB1=cat(1,BB,B); %[B;BB]

%put ok the base, that is firt all voltages, then currents
nA=size(BB);    %dimensions of vectors
nB=size(B);
BB2=changeBaseB(BB1,nA(1),nB(1));

%---------------------------------------
function [AB2] = changeBaseB(AB,nA,nB)
%AB is a transfer matrix composed by two matrices A and B, that tranfer
%[v,i] to [v;i], so AB tranfer [v;i;v;i] to [v;i;v;i];
%AB2 is the composition of AB but tranfering also [v;i] to [v;i]

XB=getBase([nA,nB]);  %[vvvvvv...,iiii....]=XB*[vi,vi,vvii,....]
AB2=XB*AB;

%-----------------------------------------------------------------------------------------
function [AB2] = changeBase(AB,nA,nB)
%AB is a transfer matrix composed by two matrices A and B, that tranfer
%[v,i] to [v;i], so AB tranfer [v;i;v;i] to [v;i;v;i];
%AB2 is the composition of AB but tranfering also [v;i] to [v;i]

XB=getBase([nA,nB]);  %[vvvvvv...,iiii....]=XB*[vi,vi,vvii,....]
AB2=XB*AB*XB^-1;
```

```
%-------------------------------------------------------------------------------------------
function [A, B] = giveMatrix(y,M)
%give the matrix A and vector B that [vi]right=A*[vi]left+B for the
%block/stripe described in x. M has the information of matrices

%   y is a Y-order struct, in the fist row there is a character 'b', 'V' or 'H' that
%   mean if is a block, stripV or stripH. The second row is the identification
%   of it.
name=name_y(y); %say if it is block 'b'  or stripe vertical 'V' or horizontal 'H'

S=giveM(M,y);   %give a matrix structure for the component described in y
[A, B]=giveABmatrices(S,name);

%-------------------------------------------------------------------------------------------
```

- createMblock.m

```
function [ MM ] = createMblock(B,stim,fi)
%createMblock create the matrix block structure MM.
%MM is a struct with the block matrix representation B is a block description, stim is the %stimulation
description and fi is index of frequency

%   General idea is with two matrices we can have the relation between column n
%   and n+1 (of voltages and currents, of course)
%   [v;i]n+1=M*[v;i]n+D*stimulation
%
%   For a total estructure we will have from 1 to N
%   [v;i]N=Mn*[v;i]1+MD

f=give_freq(stim,fi);   %frequency of analysis [Hz]

%block information
rows=numrows(B);    %number of rows
cells=numCells(B);   %number of cells per row (or columns in the block)

isw=Sw_map(B,stim,fi);  %map of currents of each block cell for the frequency in the index fi
```

```
[Z1, Z2]=obtainZ1Z2(B,f);   %Z1 and Z2 are matrices with the values of this two impedances of each block cell for the frequency f

%initializations
Mn=diag(ones(2*rows,1));  %identity matrix
MD=zeros(2*rows,1); %zero vector

for c=1:cells
%Construction of matrices M  for each column
   %matrix M construction
   A=obtainA(Z1,Z2,rows,c,f);    %A is a matrix component of matrix M
   M12=obtainM12(A,Z1,Z2,c,f);       %       -Z2*(A(:,:)^2-diag(ones(1,rows)));
   M21=obtainM21(rows,Z2,c,f);  %-1/Z2*diag(ones(1,rows));

   M1(:,:)=cat(2,A(:,:),M12);
   M2(:,:)=cat(2,M21,A(:,:));
   M(:,:,c)=cat(1,M1,M2);

   %Matrix D construction (column: [v;i]rigth=M*[v;i]left+D*isw=M*[v;i]left+SD)
   D1=obtain_D1(A,Z1,Z2,c,f);
   D2=-diag(ones(1,rows));
   D(:,:,c)=cat(1,D1,D2);  %matrix
   SD(:,c)=D(:,:,c)*isw(:,c);     %vector

%Construction of matrices M  for all block (from one side to the other) block:[v;i]rigth=Mn*[v;i]left+MD
   Mn(:,:)=M(:,:,c)*Mn(:,:);
   MD=M(:,:,c)*MD+SD(:,c);    %matrix of current switches influence at right side

end


%construction of the fields of the block matrix MM structure
MM.cells=cells;
%from column to column
MM.M=M;
MM.D=D; %matrix before the current multiplication
MM.SD=SD;      %vector after multiply D by switching current
%from block boundary to block boundary
MM.Mn=Mn;
```

```
MM.MD=MD;
MM.rows=rows;

%-------------------------------------------
function [ A ] = obtainA(ZZ,YY,rows,c,f)
%build matrix A, component of matrix M. ZZ is track impedance matrix for
%all cells, YY is cell impedance (Z2) matrix for all cells. f is frequency
if f~=0
   for rr=1:rows
      A(rr,rr)=ZZ(rr,c)/YY(rr,c)+1;
      if rr~=rows
         A(rr,rr+1)=ZZ(rr,c)/2/YY(rr,c);
      end
      if rr~=1
         A(rr,rr-1)=ZZ(rr,c)/2/YY(rr,c);
      end
   end
else    %frequency 0Hz
   for rr=1:rows
      A(rr,rr)=1;
   end
end



%-------------------------------------------------------------------------
function [D1] = obtain_D1(A,Z1,Z2,c,f)
%build a piece of matrix D for a column c at frequency f
rows=max(size(A));
if f~=0
   D1=diag(Z2(:,c))*(A(:,:)-diag(ones(1,rows)));
else
   A=obtainA(Z1,ones(rows,c),rows,c,-1);
   D1=(A-diag(ones(1,rows)));
end

%-------------------------------------------------------------------------
function [M12] = obtainM12(A,Z1,Z2,c,f)
%calculate a part of M matrix. f is frequency.c is the column we are
rows=max(size(A));
if f~=0
```

```
    M12=-diag(Z2(:,c))*(A(:,:)^2-diag(ones(1,rows)));
else
    A=obtainA(Z1,ones(rows,c),rows,c,-1);    %-1 is to made a 'normal' matrix A but with Z2 as a 1;
    M12=-2*(A-diag(ones(1,rows)));
end



%-------------------------------------------------------------------------------
function [M21] = obtainM21(rows,Z2,c,f)
%calculate a part of M matrix. f is frequency.c is the column we are

if f~=0
    M21=-(diag(Z2(:,c))^-1);

else
    M21=zeros(rows,rows);
end

%-------------------------------------------------------------------------------
function [isw] = Sw_map(B,stim,fi)
%Sw_map give a matrix with values of switching current for the frequency of index fi,
%for the block B

rows=numrows(B);    %rows of the block
cols=numCells(B);   %cells per row (or columns)

for r=1:rows
    for c=1:cols
        cell=giveCell(B,r,c); %give the cell at row r column c
        type=giveType(cell);    %give the type of switch the cell has (0=not switching)
        isw(r,c)=giveStimCurrent(stim,type,fi); %give the frequencial component for the index fi of the
situmulation of type type
    end
end

%-------------------------------------------------------------------------------------------
```

- createMstripH.m:

```
function [ M ] = createMstripH( S, f )
%M is a struct with the horizontal stripe matrix representation
%S is a stripe destription,
%f is the index of  frequency

%  [v;i]left=MSH*[v;i]right+DPH
%              |PAD1   |PAD2
%  left ----- ---------- ---------- right
%       ----- ---------- ----------
%   MS1,DP1   MS2 DP2       MS3
%
%     MSH=MS3*MS2*MS1
%     DPH=MS3*MS2*DP1*vpad1+MS3*DP2*vpad2

%Stripe information
cells=numCells(S);
vp=obtainVpad(f,'f');   %voltage of alimentation 1.2V in dc 0V in ac
npads=numPads(S);



%initializations
MSH=diag(ones(1,2));    %identity matrix
DPH=zeros(2,1); %zeros vector

%from pad to pad in the horizontal strip [v,i]rigth=MS*[v,i]left+DP*vpad
MS(:,:,:)=obtainMS(S,f); %MS is the matrix [a,b;c,d] that has the relation from pad to pad in the
horizontal strip
if S.npads~=0
   DP(:,:,:)=obtainDP(S,f);    %DP is the matrix [D1;D2] that has the influence of the voltage of the pad


   for i=1:npads
      MSH=MS(:,:,i)*MSH;  %MSH is the matrix between left and right side of the strip
[v,i]left=MSH*[v,i]right+MP
      DPH=MS(:,:,i)*DPH(:,:)+DP(:,:,i)*vp;               %MP is the influence of voltage input pad
   end
   DPH=MS(:,:,npads+1)*DPH;
   MSH=MS(:,:,npads+1)*MSH; %
```

```
else
   MSH=MS;
   DP=zeros(2,1);
end
%structure fields
M.vp=vp;    %voltage of pad
M.npads=numPads(S); %number of pads
M.MSH=MSH;  %matrix from stripe boundary to stripe boundary
M.DPH=DPH;  %independent term from stripe boundary to stripe boundary
M.MS=MS;    %matrices from begin to pad, pad to pad, and last pad to end
M.DP=DP;    %independent terms from begin to pad, pad to pad, and last pad to end



%-------------------------------------------
function [ M ] = obtainMS(S,f)
%return the matrices begin of strip to first pad, pad to pad and last pad
%to end, is neded to describe the strip. f is frequency

%Stripe information
[R,L,G,C]=giveStripProps(S);  %electrical parameters
cells=numCells(S);    %cell divisions in the stripe
npads=numPads(S);   %number of pads in stripe S

%calculations
Z1=getZRLC(R,L,-1,f);   %impedance branch of an L and R in series
Z2=getZRLC(-1,-1,C,f);  %impedance branch of C
[a,b,c,d]=obtainTLvalues(Z1,Z2);    % a b c d parameters of a strip segment (as a segment of
transmission line without pad)
if npads~=0
   long=0; %number of cells between pad to pad or begin to first pad

   for ii=1:npads
      %pad data
      pad=givePad(S,ii);
      pos=givePadPos(pad,'H');
      [R,L]=givePadProps(pad);
      long=pos-long;
```

```
        ZL=getZRLC(R,L,-1,f);   %give the value of an impedance defined as a series of R L and C for a
frequency f. Negaitve input means that there is not this component.

        if ZL~=0
            M(:,:,ii)=[1,0;-1/ZL,1]*[a,b;c,d]^(long);   %(long cells of normal transmission line plus pad)
        else
            M(:,:,ii)=[1,0;-1e22,1]*[a,b;c,d]^(long);
        end
    end

    long=cells-pos; %number of cells between last pad and the end of stripe
else
    long=cells;
end

M(:,:,npads+1)=[a,b;c,d]^long;

%-------------------------------------------------------------------------------------------------------
 function [DP] = obtainDP(S,f)
%return the matrix DP that is relation the influence of the pad to the
%transmission of [v;i]left to right   (rigth=MS*left+DP*vpad)

for ii=1:S.npads
    ZL=getZRLC(S.PAD(ii).R,S.PAD(ii).L,-1,f);   %give the value of an impedance defined as a series of
R L and C for a frequency f. Negaitve input means that there is not this component.
    if ZL~=0
        DP(:,:,ii)=[0;1/ZL];
    else
        DP(:,:,ii)=[0;1e22];
    end
end
DP(:,:,S.npads+1)=zeros(2,1);

%-------------------------------------------------------------------------------------------------------
```

- createMstripV.m:

```
function [ M ] = createMstripV( S, f )
%M is a struct with the vertical stripe matrix representation
```

```
%S is a stripe description, f is the frequency of calculation

%   General idea is with two matrices we can have the relation between the wwo sides of the stripe
%   (of voltages and currents, of course)
%   [v;i]rigth=MSV*[v;i]left

%Stripe information
rows=numrows(S);
%construction of matrices
%voltage part
I=diag(ones(1,rows)); %with left voltage (is the same, so identity matrix)
MSV1=cat(2,I,zeros(rows,rows));  %with left current (is independent, so zeros)

%current part;
[R, L, G, C]=giveStripProps(S); %electrical parameters
Z1=getZRLC(R,L,-1,f);   %impedance of stripe tracks at f [Hz]; -1 means that there is not this
component in the series branch
Z2=getZRLC(G,-1,C,f);   %impedance in between tracks at f [Hz]

[a,b,c,d]=obtainTLvalues(Z1,Z2);

bK=getbK(d,rows);    %bK is a lower triangular matrix with 0 in the diagonal i d^k below k=0:+1:until
the end of the column
cI=getcI(a,rows);    %cI is a matrix with a in the diagonal and -1 in the upper sub diagonal

RV=(I+d*bK)^-1*(c*bK+1/b*cI);

MSV2=cat(2,RV,I);
MSV=cat(1,MSV1,MSV2);

[MpadL, MpadR]=getMpad(S,f); %give the transfer matrices to the left or right boundary (MpadL,
MpadR) and the pad configuration matrix with 'l', 'r' or '0' of  and global row position(Mside)

%build fields of matrix stripe vertical structure
M.MSV=MpadR*MSV*MpadL;
M.abcd=[c,d;a,b];
M.bK=bK;
```

```
%-------------------------------------------------------------------------------------------
function [bK] = getbK(b,n)
%build the matrix dK
bK=zeros(n,n);
for k=0:n-2
   bK=bK+diag((b^k)*(ones(n-k-1,1)),-k-1);
end

%-------------------------------------------------------------------------------------------
function [cI] = getcI(c,n)
%build an identity matrix with c value in the next to diagonal positions
cI=zeros(n,n);
for k=1:n-1
   cI(k,k)=c;
   cI(k,k+1)=-1;
end

 %-------------------------------------------------------------------------------------------
function [MpadL, MpadR] = getMpad(S,f)
%build the boundary pad matrices left and right to the stripe.)
%
% stripe information
rows=numrows(S);
npads=numPads(S);
% Initializations
MpadR=diag(ones(2*rows,1)); %identity matrix
MpadL=MpadR;

for k=1:npads
   pad=givePad(S,k);
   [x, side]=givePadPos(pad,'V');  %give the row were is the pad and the side
   [R, L]=givePadProps(pad);
   ZL=getZRLC(R,L,-1,f);   %series branch of r and l not c (-1)

   if side=='r'
      MpadR(x,rows+x)=-ZL;
   else
      MpadL(x,rows+x)=-ZL;
   end
end
```

%----------

- Sort_x.m:

```
function [x, nx] = sort_x(c)
%SORT_X organizes the block or stripe structures inside the chip c. Sort_x
%creates a X-order structure associated to the chip structure c. The
%criteria is follow from left to right the x coordinate where a block or
%stripe begins (its left position) and take the characteristics of all
%structures presents at this coordinate place in a table of y-order
%structures sorted by the y coordinate from down to up.

%  nx is the number  of vertical divisions done to the chip c.

%chip composition
rows=numrows(c);    %number of rows divisions of chip
blocks=numBlocks(c);    %blocks
sV=numStripes(c,'V');   %vertical stripes
sH=numStripes(c,'H');   %horizontal stripes

%put in a table the information (in columns): x position, y position, type of structure,
%number inside chip, and rows.  Each stripe or block will be a row of the
%table
for s=1:sV
    stripe=giveStripe(c,'V',s);
    xsV(s,1)=posX(stripe);
    xsV(s,2)=posY(stripe);
    xsV(s,3)='V';
    xsV(s,4)=s;
    xsV(s,5)=numrows(stripe);
end
clear('Stripe');
for s=1:sH
    stripe=giveStripe(c,'H',s);
    xsH(s,1)=posX(stripe);
    xsH(s,2)=posY(stripe);
    xsH(s,3)='H';
    xsH(s,4)=s;
```

```
    xsH(s,5)=1;
end

for b=1:blocks
    block=giveBlock(c,b);
    xb(b,1)=posX(block);
    xb(b,2)=posY(block);
    xb(b,3)='b';
    xb(b,4)=b;
    xb(b,5)=numrows(block);
end

%put the 3 tables in 1
if sV~=0
    xsb=xsV;
end
if sH~=0
    xsb=cat(1,xsb,xsH);
end
if blocks~=0 && ((sV || sH )~=0)
    xsb=cat(1,xsb,xb);
else
    xsb=xsb;
end
%xx is a table with same columns of xsb, but sorted from lower x position
%to high x position. If x is same the criteria is from lower y to high y
xx=sortrows(xsb);   %will sort the table from lower to upper value following the first column (x
%coordinate). xx is a table
nx=size(xx);    %dimensions of table
nx=nx(1);   %number of rows, or number of stripes+blocks

%Construction of x struct
j=1;    %counter of the different coordinate x of the structures.
i=1;    %row in the table xx
k=1;    %number of subchip structures in same coordinate (block or stripes)
x(j).y(k)=createYorder(xx(i,2),xx(i,3),xx(i,4),xx(i,5));
x(j).ny=k;
i=2;

k=2;
```

```
while i<=nx     %not finish the stripes or blocks (not finished rows of table xx)
   if   xx(i,1)==xx(i-1,1) %same x coordinate as previous row
      x(j).y(k)=createYorder(xx(i,2),xx(i,3),xx(i,4),xx(i,5));
      x(j).ny=k;
      k=k+1;
   else    %different x coordinate as previous row
      k=1;    %first row with the new x coordinate
      j=j+1;
      x(j).y(k)=createYorder(xx(i,2),xx(i,3),xx(i,4),xx(i,5));
      x(j).ny=k;
      k=2;
   end
   i=i+1;
end

nx=j;   %number of different x coordinates, number of chip vertical divisions

%Check for empty spaces
%Begin again
for i=1:nx
   a=putEmptySpaces(x(i),posY(c),posY(c)+rows); %Xorder with empty spaces
   x(i)=a;
end
%-------------------------------------------------------------------------------------------------

function [ x2 ] = putEmptySpaces(x,y0,ymax)
% x is a xorder structure without check for empty spaces, x2 is the same structure including empty
spaces
%y0 is the origin of coordinates

nye=0;   %number of blocks+stripes+empty spaces; sharing vertical division
ny=numy(x); %number of blocks+stripse, sharing vertical division
Y=giveYorder(x,1);
%Compare with the lower edge of chip
if y0 < Y.y  %there is empty space
   nye=nye+1;
   empty_rows=Y.y-y0;
   Y2=createYorder(posY(c),'0',0,empty_rows);
   nye=nye+1;
   x2.y(nye)=Y2;
```

```
end
nye=nye+1;
x2.y(nye)=Y;
for i=2:ny

   Y2=giveYorder(x,i);
   if emptySpace(Y,Y2)     %there is a empty space
      nye=nye+1;
      empty_rows=Y2.y-Y.y-Y.rows;
      Y3=createYorder(posY(c),'0',0,empty_rows);;
      nye=nye+1;
      x2.y(nye)=Y3; %store the empty space
      nye=nye+1;
      x2.y(nye)=Y2; %store the not empty space
   else
      nye=nye+1;
      x2.y(nye)=Y2;
   end
   Y=Y2;
end
%compare with the upper edge of chip
if Y.y+Y.rows+1<ymax
   nye=nye+1;
   empty_rows=ymax-Y.y-Y.rows;
   Y2=createYorder(posY(c),'0',0,empty_rows);
   x2.y(nye)=Y2;

end
x2.ny=nye;  %number of horizontal divisions for the x vertical division

%-------------------------------------------------------------------------------------
function [ e ]=emptySpace(Y1,Y2);
%say if there is empty space between two Yorder Y1 Y2 structures

if Y1.y > Y2.y
   e=(Y2.y+Y2.rows < Y1.y);
else
   e=(Y1.y+Y1.rows < Y2.y);
```

```
end

%-------------------------------------------------------------------------------------------
```

- giveBoundaries.m:

```
function [ bon ] = giveBoundaries( c,fi )
%GIVEBOUNDARIES analyzes the chip c and give a struct with the boundaries data.
%  fi is index of frequencies

%the result 'rl' is a vector 2*(2*rows*'#boundaries') with ones where vi is know and 0 if
%is unknown. The order in columns is following x coordinate, vix1, vix2,
%....
%kv is a vector dimension=(2*rows,1) with the values of boundary known and sorted first vi left and
then vi right

%inizialize
rows=numrows(c);   %number of rows of the chip
y0=posY(c); %center of coordinates

right_uk=zeros(2*rows,1);   %vector with position =1 is the boundary is known or =0 if is unkown. First
voltages, then currents
left_uk=zeros(2*rows,1);
r=0;    %number of voltage boundaries on right side
l=0;    %number of voltage boundaries on left side
left_values=zeros(rows,1);  %value of the known boundary
right_values=zeros(rows,1);



%Boundary pads in vertical strips can only be left or right and it gives the
%boundaries of the chip
for i=1:numStripes(c,'V')
    SV=giveStripe(c,'V',i);
    if numPads(SV)~=0
        for ii=1:numPads(SV)
            y=posY(SV);
            pad=givePad(SV,ii);
            [position, side]=givePadPos(pad,'V');   %position relative to stripe. Left or right
            if  side=='l'  %left side pad
```

```
            pos=position+y-y0;  %position relative to chip

            left_uk(pos,1)=1;    %known voltage
            l=l+1;  %# left pads
            left_values(l,1)=obtainVpad(fi,'index');   %pad voltage at frequency index fi

         else
            pos=position+y-y0;
            right_uk(pos,1)=1;     %known voltage

            r=r+1;   %# right pads
            right_values(r,1)=obtainVpad(fi,'index');  %pad voltage at frequency index fi

         end
      end
   end
end

%rl 0 or 1 if is known or not, fist right v, i, then left v, i. Now v are put, we have to put i, that is, where
we don't know v.
for i=1:rows
   %left boundary
   if left_uk(i)==0 %not voltage known
      left_uk(i+rows,1)=1;     %current is known (=0)
   end
   %rigth boundary
   if right_uk(i)==0      %not voltage known
      right_uk(i+rows,1)=1;   %current is known (=0)
   end
end

bon.nknown_left=l;  %number of known boundaries
bon.nknown_right=r;

bon.ku_l=left_uk;   %vector 1=known 0=unknown in left boundary
bon.ku_r=right_uk;

bon.val=cat(1,left_values,right_values);   %values of the known boundaries first left then right
```

```
%-------------------------------------------------------------------------------------------
```

- SolveChip.m:

```
function [ S ] = SolveChip( CM, boundary )
%SOLVECHIP Give the solution of voltages level and currents at all
%blocks/stripes boundaries using the matrices of CM (chip matrix).
%boundary has the information of the known and unknown boundaries
%

%Chip data
rows=numrows(CM);   %number of rows that the chip has (y dimension)
nx=numx(CM);    %number of stripe/block/emptyspace structure columns that the chip is divided

known=getknown(boundary,nx,rows);  %the result is a vector 2*(2*rows*'nx+1') with ones where vi is
know and 0 if is unknown. The order in columns is following x coordinate, vix1, vix2, .... kn is a vector
dimension=(2*rows,1) with the values of boundary known and sorted first vi left and then vi right
vk=give_values(boundary);   %give the values of the known boundaries, first vi left then vi right

[Ax,Bx]=giveGlobalM(CM);    %give the matix strucutre of all relationships between the divisions of the
chip. For each x division [vi]x+1=Ax*[vi]x+Bx

%sistem of equations to solve
AA=buildMM(Ax,nx,rows);    %result is a not square matrix with all the blocks/stripes equations
BB=buildDD(Bx,nx);       % 0=AA*[vivivivivi]+BB
% now 0=AA*[vivivivi]+BB

XB=getXB(known,2*rows);   %this matrix will sort the [vivivivivi] vector by [viknown,
viunknown]=XB*[vivivivi]
AA2=AA*XB^-1;   %now, AA2 has the first 2*rows column that will be multiplied be vi values that are
known

[Ak, Au]=divideMM(AA2); %Au will be a square matrix, and Ak will be multiplied by the known vi
values and will became a vector
%%now 0=Ak*[viknown]+Au*[viunknown]+D

B=(Ak*vk+BB);% 0=Au*[viunknown]+B
```

```
%Solve linear equation
%  [T, Aubal]=balance(Au); %Aubal=T\Au*T
%  Bbal=T^-1*B;
%  vubal=-Aubal^-1*Bbal;
%  vu=T*vubal;      % 0=T^-1*Mu*T*(T^-1*vu)+T^-1*D  and vubal=T^-1*vu
vu=linsolve(Au,-B);

vi2=cat(1,vk,vu);   %result vi but ordered by previously known and previously unknown
vi=XB^-1*vi2;   %vi;  sorted by x1,x2,x3 from left to right

%store boundary results following the x coordinate of block/stripe structures.
%vi is a vector of nx*rows*2, each chip block/stripe/emptyspace structure
%column has rows voltages and rows currents.
for i=1:nx+1
    S(i).v=vi(1+(i-1)*2*rows:(i-1)*2*rows+rows);
    S(i).i=vi((i-1)*2*rows+1+rows:(i-1)*2*rows+2*rows);
end
%-----------------------------------------------------------------------------------------------

function [XB] = getXB(a,nk)
%give the change base matrix that puts first the known values, and later
%the unknown. a is a vector with 0 when is unknown and 1 when is known. a
%vector is firt left v and i, and then v and i of right side.

%[viknown,viunknown]=XB*[vi,vi,...,vi]
%viunknown will be a vector with viunkonwn going from left to right
%nk is the number of known values

k=0;    %known counter
u=0;    %unknown counter

na=size(a);

for ii=1:na(1)
    if a(ii)==1
        k=k+1;
        XB(k,ii)=1;
    else
        u=u+1;
```

```
        XB(nk+u,ii)=1;
    end
end
 %-------------------------------------------------------------------------------------------------

function knw = getknown(bon,nx,row)
%the result knw is a vector 2*(2*rows*'#boundaries') with ones where vi is know and 0 if
%is unkwown. The order in columns is following x coordinate, vix1, vix2,
%....
%   bon is a chip boundary struct (only left and right of chip )
%   nx is the number of internal boundaries of the chip
%   row is the number of rows that the chip is divided

[left, right]=give_ku(bon);
intermediate=zeros(2*row,1);    %intermediate boundaries, nothing is known
knw=left;
for i=1:nx-1
    knw=cat(1,knw,intermediate);    %put the two vectors in one. One next to the other
end
knw=cat(1,knw,right);

%-------------------------------------------------------------------------------------------------
function [Mk, Mu] = divideMM(MM)
%give the known matrix Mk that is the first half, and the unknown matrix
%the second half MM=[Mk Mu]
m=size(MM);     %dimensions of MM
mk=m(2)-m(1);
Mk(:,:)=MM(1:m(1),1:mk);   %k fist columns are multiplied by vi known
Mu(:,:)=MM(1:m(1),mk+1:m(2));



%-------------------------------------------------------------------------------------------------
function [ M ] = buildMM(Mx,nx,rows)
% Mx are all the matrices of 2rows*2rows that [v;i]x+1=Mx[v;i]x+Dx; M will
% be a not square matrix with all the equations = 0, from left to right
% [Mx1  -I           ] [ [v;i]x1 ]  [Dx1]
% |0    Mx2 -I       | | [v;i]x2 |  |Dx2|
% |0     0  Mx3 -I   |*| [v;i]x3 |+|      |=0
% |     .....        | |         |  [Dxn]
```

```
% [0       0 Mxn -I ] [[v;i]xn+1]
%
%Position xn+1 has not Mx and Dx because is the right side, that is, there
%is not xn+2

I=diag(ones(2*rows,1));

for k=1:nx
   M((k-1)*2*rows+1:k*2*rows,(k-1)*2*rows+1:k*2*rows)=Mx(:,:,k);
   M((k-1)*2*rows+1:k*2*rows,(k)*2*rows+1:(k+1)*2*rows)=-I;
end

%-------------------------------------------------------------------------------------------
function [ D ] = buildDD(Dx,nx)
% Mx are all the matrices of 2rows*2rows that [v;i]x+1=Mx[v;i]x+Dx; M will
% be a not square matrix with all the equations = 0, from left to right
% [Mx1  -I            ] [ [v;i]x1 ]  [Dx1]
% |0    Mx2 -I        | | [v;i]x2 |  |Dx2 |
% |0     0  Mx3 -I    |*| [v;i]x3 |+|    |=0
% |     .....         | |         |  [Dxn]
% [0       0 Mxn -I ] [[v;i]xn+1]
%
%Position xn+1 has not Mx and Dx because is the right side, that is, there
%is not xn+2
D=Dx(:,1);
for k=2:nx
   D=cat(1,D,Dx(:,k));

end

%-------------------------------------------------------------------------------------------
```

- getResults.m:

```
function [ R ] = getResults (sol, CM )
%GETRESULTS sol is the array with the values of all [vi] from left to right
%at the position of start of each structure (block, stripV or stripH). CM
%contains all the matrices and also CM.x have the information of how is
%sort sol.
```

```
c_pos_y=Y0(CM); %origin of coordinates
for x=1:numx(CM)   %look for block or strip start
   col=giveXorder(CM,x);   %column of blocks/stripes/emptyspaces
   for y=1:numy(col) %look for component at the same x position
      YY=giveYorder(col,y);   %block/stripe/emptyspace at position y of division col
      vi=getvi(sol(x:x+1),YY,c_pos_y);
      place=givePlace(YY);
      switch name_y(YY)
         case 'b'
            B=giveM(CM,YY); %block matrices
            R.b(place)=CalculateBlock(vi,B);    %results of block place
            clear('B');
         case 'V'
            S=giveM(CM,YY); %stripe matrices
            [R.Stripe.V(place)]=CalculateStripV(vi,S);
            clear('S');
         case 'H'
            S=giveM(CM,YY); %stripe matrices
            [R.Stripe.H(place)]=CalculateStripH(vi,S);
            clear('S')
         case '0'
            %nothing at that x,y, no block no strip beginning, only empty space.

            %otherwise

      end
   end
end
R.nblocks=numBlocks(CM);
R.Stripe.nV=numStripes(CM,'V');
R.Stripe.nH=numStripes(CM,'H');

%-----------------------------------------------------------
function [ vi ] = getvi(VI,y,y0)
%from general solution all column of position x, give the solution that
%belongs to coordinates y, y+rows
posy=y.y;    %low y coordinate
rows=y.rows;
```

```
pl=posy-y0;  %place inside the vector VI where is the information of the v of the fist row of the
component x(x).y(y) (that stats at x,y)
for ii=1:2
    v(:,ii)=VI(ii).v(pl+1:pl+rows); %first c.rows are v and second c.rows are i inside VI
    i(:,ii)=VI(ii).i(pl+1:pl+rows);
end
vi=cat(1,v,i);


%-------------------------------------------------------------
function [ R, err ] = CalculateStripV(vi,SV)
%vi(:,1)is the voltages, current on left side of the vertical strip, vi(:,2) is on the right side


rows=max(size(SV.MSV))/2;
R.v=vi(1:rows,1);
%current going trough the strip from down to up
c=SV.abcd(2,1);
d=SV.abcd(2,2);
R.i=c.*SV.bK*vi(1:rows,1)+d.*SV.bK*(vi(rows+1:2*rows,1)-vi(rows+1:2*rows,2));


%errors for testing\
ERR=vi(:,2)-SV.MSV*vi(:,1);
err=max(abs(ERR));


%-------------------------------------------------------------
function [ R, err ] = CalculateStripH(vi,SH)
%vi(:,1)is the voltages, current on left side of the vertical strip, vi(:,2) is on the right side


pieces=numPads(SH)+1;   %pieces of the strip |----|----| =3 divisions, 2 pieces, and 1 pad)
%voltage and current going trough the strip from left to right
%.MS(:,:,i) is the matrix from i to i+1 portion of stripH (beginning to first pad, or pad to pad, or pad to
end), D(:,:,i) i the
%influence of i pad of H strip. The last piece doest have pad, so there is not D.
VIh(:,1)=vi(:,1);
[MS, DP, vp]=giveSHmatrices(SH);    %relation between internal boundaries and voltage pad


VIh(:,2)=MS(:,:,1)*vi(:,1)+DP(:,1)*vp;
for i=2:pieces-1
    VIh(:,i+1)=MS(:,:,i)*VIh(:,i)+DP(:,1,i)*vp;
end
```

```
VIh(:,pieces+1)=MS(:,:,pieces)*VIh(:,pieces);
R.v(1,:)=VIh(1,:);
R.i(1,:)=VIh(2,:);  %1 and 2 is because is as sigle row, that is, there is only 1 voltage and 1 current per
piece of strip
%errors\
ERR=vi(:,2)-VIh(:,pieces+1);
err=max(abs(ERR));

%-------------------------------------------------------------------------------------------
```

- calculateBlock.m:

```
function [ Rb, err ] = CalculateBlock(sol,B)
%CALCULATEBLOCK Calculate all the voltages and currents inside because sol is the
%boundary conditions at left. M has all matrices from left to right. b is
%the input information of the block
%  sol(:,1) is left solution, sol(:,2) is right solution.

rows=numrows(B);
cells=numCells(B);
[M,D]=giveMDmatrix(B);

Rb.v(:,1)=sol(1:rows,1);
Rb.i(:,1)=sol(rows+1:2*rows,1);

RR(:,1)=M(:,:,1)*sol(:,1)+D(:,1);     %M is from cell to cell, D is D*isw product
Rb.v(:,2)=RR(1:rows,1);
Rb.i(:,2)=RR(rows+1:2*rows,1);

for i=2:cells
    RR(:,i)=(M(:,:,i)*RR(:,i-1)+D(:,i));
    Rb.v(:,i+1)=RR(1:rows,i);
    Rb.i(:,i+1)=RR(rows+1:2*rows,i);
end

%verification of errors only for test
ERR=RR(:,cells)-sol(:,2);
err=max(abs(ERR));
```

```
Rb.v(:,1)=sol(1:rows,1);
Rb.i(:,1)=sol(rows+1:2*rows,1);

Rb.rows=rows;   %number of rows

%-------------------------------------------------------------------------------------------
```

- convertToTime.m:

```
function [ T ] = convertToTime(R, stim)
%CONVERTTOTIME R is a struct with all the information of voltages and
%currents, in frequency domain. stim has the information of chip stimulation, and
%frequencies.  T has the evolution of voltages
%and currents in time.

%time/frequency data
t=getTimeAxis(stim);    %time axis
n2=give_n2(stim);   %number of positive + 0 frequencies
nf=give_nf(stim);   %number of frequencies (positives + negatives +0)

FR=addConjugate(R,n2);    %FR has the information with all points negative frequencies included

%creation of time results structure
T=doIFFT(FR,nf);    %results in time domain
T.t=t;  %addition of time axis
%another fields
T.nblocks=numBlocks(R(1)); %number of blocks
T.Stripe.nV=numStripes(R(1),'V');  %number of verical stripes
T.Stripe.nH=numStripes(R(1),'H');  %number of horizontal stripes

%------------------------------------------------------------
function [ FR ] = addConjugate(R,n2)
%Do the pre-process before do ifft, adds the conjugate to complete the frequency results.

nblocks=numBlocks(R(1));
nH=numStripes(R(1),'H');
nV=numStripes(R(1),'V');
```

```
for f=1:n2-1   %half frequencies, not 0. n2 is positive frequencies + 0Hz


  for b=1:nblocks
    FR.b(b).rows=numRows(R(b).b(b));
    %copy frequency 0
    if f==1
      FR.b(b).v(:,:,1)=giveV(R,'b',b,f); %R(1).b(b).v;
      FR.b(b).i(:,:,1)=giveI(R,'b',b,f); %R(1).b(b).i;
    end
    %copy from frequency 1 to nyquist(placed in n2)
    FR.b(b).v(:,:,f+1)=giveV(R,'b',b,f+1); %R(f+1).b(b).v;
    FR.b(b).i(:,:,f+1)=giveI(R,'b',b,f+1); %R(f+1).b(b).i;
     %adding conjugate
    FR.b(b).v(:,:,f+n2)=conj(giveV(R,'b',b,n2-f+1)); %R(n2-f+1).b(b).v);
    FR.b(b).i(:,:,f+n2)=conj(giveI(R,'b',b,n2-f+1)); %R(n2-f+1).b(b).i);
  end
  for H=1:nH
    %copy from frequency 1 to nyquist(placed in n2)
    FR.Stripe.H(H).v(1,:,f+1)=giveV(R,'H',H,f+1); %R(f+1).Stripe.H(H).v(1,:);
    FR.Stripe.H(H).i(1,:,f+1)=giveI(R,'H',H,f+1); %R(f+1).Stripe.H(H).i(1,:);
     %adding conjugate
    FR.Stripe.H(H).v(1,:,f+n2)=conj(giveV(R,'H',H,n2-f+1)); %R(n2-f+1).Stripe.H(H).v);
    FR.Stripe.H(H).i(1,:,f+n2)=conj(giveI(R,'H',H,n2-f+1)); %R(n2-f+1).Stripe.H(H).i);
    %copy frequency 0
    if f==1
      FR.Stripe.H(H).v(1,:,1)=giveV(R,'H',H,f); %R(1).Stripe.H(H).v;
      FR.Stripe.H(H).i(1,:,1)=giveI(R,'H',H,f); %R(1).Stripe.H(H).i;
    end
  end
  for V=1:nV
    %copy frequency 0
    if f==1
      FR.Stripe.V(V).v(:,1,1)=giveV(R,'V',V,f); %R(1).Stripe.V(V).v;
      FR.Stripe.V(V).i(:,1,1)=giveI(R,'V',V,f); %R(1).Stripe.V(V).i;
    end
    %copy from frequency 1 to nyquist(placed in n2)
    FR.Stripe.V(V).v(:,1,f+1)=giveV(R,'V',V,f+1); %R(f+1).Stripe.V(V).v;
    FR.Stripe.V(V).i(:,1,f+1)=giveI(R,'V',V,f+1); %R(f+1).Stripe.V(V).i;
    %adding conjugate
```

```
        FR.Stripe.V(V).v(:,1,f+n2)=conj(giveV(R,'V',V,n2-f+1)); %R(n2-f+1).Stripe.V(V).v);
        FR.Stripe.V(V).i(:,1,f+n2)=conj(giveI(R,'V',V,n2-f+1)); %R(n2-f+1).Stripe.V(V).i);
    end
end

%some more fields
FR.nblocks=nblocks; %number of blocks
FR.Stripe.nV=nV;  %number of vertical stripes
FR.Stripe.nH=nH;  %number of horizontal stripes

%-------------------------------------------------------------
function [ T ] = doIFFT(FR,nf)
nblocks=numBlocks(FR);

for b=1:nblocks
    for row=1:FR.b(b).rows
        [vector_v, vector_i]=putINvector(FR,b,row,'b',nf);   %vector_vi(frequency,column of row 'row')
        v_row=ifft(nf.*vector_v);
        i_row=ifft(nf.*vector_i);
        for t=1:nf
            vt(row,:,t)=v_row(t,:);
            it(row,:,t)=i_row(t,:);
        end

    end
    T.b(b).v=vt;
    T.b(b).i=it;
    clear('vt','it');
end

NumH=numStripes(FR,'H');
for nH=1:NumH
    [vector_v, vector_i]=putINvector(FR,nH,1,'H',nf);   %vector_vi(frequency,stripes values)
    v_row=ifft(nf.*vector_v);
    i_row=ifft(nf.*vector_i);
    for t=1:nf

        vt(1,:,t)=v_row(t,:);
        it(1,:,t)=i_row(t,:);
```

```
    end
    T.Stripe.H(nH).v=vt;
    T.Stripe.H(nH).i=it;
    clear('vt','it');
end

NumV=numStripes(FR,'V');    %number of vertical stripes with calculated result
for nV=1:NumV
    [vector_v, vector_i]=putINvector(FR,nV,1,'V',nf);   %vector_vi(frequency,column of row 'row' and for
strips values for positions)
    v_row=ifft(nf.*vector_v);
    i_row=ifft(nf.*vector_i);
    for t=1:nf
        vt(:,1,t)=v_row(t,:);
        it(:,1,t)=i_row(t,:);
    end
    T.Stripe.V(nV).v=vt;
    T.Stripe.V(nV).i=it;
    clear('vt','it');
end

%-------------------------------------------------------------
function [v, i] = putINvector(FR,pos,row,c,nt)
%c is a char: 'b', 'H' or 'V' to know what is data input

switch c
    case 'b'
        for t=1:nt
            v(t,:,1)=FR.b(pos).v(row,:,t);
            i(t,:,1)=FR.b(pos).i(row,:,t);
        end

    case 'V'
        for t=1:nt
            v(t,:,1)=FR.Stripe.V(pos).v(:,1,t);
            i(t,:,1)=FR.Stripe.V(pos).i(:,1,t);
        end

    case 'H'
```

```
        for t=1:nt
            v(t,:,1)=FR.Stripe.H(pos).v(1,:,t);
            i(t,:,1)=FR.Stripe.H(pos).i(1,:,t);
        end

        %otherwise
end

%-------------------------------------------------------------------------------------------------
function [ t ] = getTimeAxis(stim)
%give the time axis
nf=stim.nf;
time_step=stim.Ts;

t=[0:nf-1]*time_step;

%-------------------------------------------------------------------------------------------------
```

- PlotResults.m

```
function [ ] = PlotResults( R )
%Plot results of stripes and blocks (one row and one column).
%R is the time domain results of chip

bb=numBlocks(R);   %number of blocks
for ii=1:bb
    xy=size(R.b(ii).v); %dimensions of block
    %block row plot
    row=4;  %row to be ploted
    figure;
    vr(:,:,1)=R.b(ii).v(row,1:floor(xy(2)/10)+1:xy(2),:);
    plot(R.t,vr(:,:,1))
    title(sprintf('v%u row',ii))

    figure
    ir(:,:,1)=R.b(ii).i(row,1:floor(xy(2)/10)+1:xy(2),:);
    plot(R.t,ir(:,:,1))
    title(sprintf('i%u sw row',ii))
```

```
    %block column plot
    figure;
    col=floor(xy(2)/2)+1;   %middle point. Column to be ploted
    v(:,:,1)=R.b(ii).v(:,col,:);
    plot(R.t,v(:,:,1))
    % % hold on
    % v(:,:,1)=R.b.v(:,117,:);
    % plot(R.t,v(:,:,1),'.-')
    %
    title(sprintf('v column%u',ii))

    figure
    i(:,:,1)=R.b(ii).i(:,col,:);
    plot(R.t,i(:,:,1))

    title(sprintf('i sw column%u',ii))
    clear('v','vr','ir','i');
end

%stripe plots
VV=numStripes(R,'V');
for ii=1:VV
    figure
    vV1(:,:,1)=R.Stripe.V(ii).v(:,1,:);
    plot(R.t,vV1(:,:,1))
    title(sprintf('v stripeV%u',ii))

    figure
    iV1(:,:,1)=R.Stripe.V(ii).i(:,1,:);
    plot(R.t,iV1(:,:,1))
    title(sprintf('i stripeV%u',ii))
    clear('vV1','iV1');
end

HH=numStripes(R,'H');
for ii=1:HH
    figure
    vH1(:,:,1)=R.Stripe.H(ii).v(1,:,:);
    plot(R.t,vH1(:,:,1))
```

```
    title(sprintf('v stripeH%u',ii))

    figure
    iH1(:,:,1)=R.Stripe.H(ii).i(1,:,:);
    plot(R.t,iH1(:,:,1))
    title(sprintf('i stripeH%u',ii))
    clear('vH1','iH1');
end

%--------------------------------------------------------------------------------------------
```

A.3.3. Funcions consultores

En aquesta secció s'han agrupat les funcions consultores que retornen alguna informació de les estructures de dades.

- give_freq.m:

```
function [ f ] = give_freq( stim, i )
%GIVE_FREQ give the value of frequency that belongs to the i position
%  i is a index. i=1 mean frequency 0 for the stimulus stim

f=stim.F(i);

%-------------------------------------------------------------------------------------------
```

- give_ku.m:

```
function [ ku_l, ku_r ] = give_ku( bound )
%GIVE_KU give the vectors of known or unknown boundary of bound struct
%       This vectors have 1 if the boundary is known or 0 if is unknown.
%       They are ordered per row, first voltges, later currents
ku_l=bound.ku_l;
ku_r=bound.ku_r;

%-------------------------------------------------------------------------------------------
```

- give_n2.m:

```
function [ n2 ] = give_n2( stim )
%GIVE_N2 show the number of positive with 0 frequencies that the stimulies
%are made

n2=stim.n2;


%-------------------------------------------------------------------------------------------
```

- give_nf.m:

```
function [ nf ] = give_nf( stim )
%GIVE_NF show the number of frequencies needed to define the stimulies
%with also negative frequencies. The current stimulis stored only have positive and zero frequencies

nf=stim.nf;

%-----------------------------------------------------------------------------------------------
```

- give_values.m:

```
function [ vals ] = give_values( bound )
%GIVE_VALUES give the values of the known boundaries of the chip, stored in
%bound
%  The values are ordered first left known voltage, left known currents, right known voltages
%  right known currents

vals=bound.val;



%-----------------------------------------------------------------------------------------------
```

- giveABmatrices.m:

```
function [ A, B ] = giveABmatrices( M, c )
%GIVEABMATRICES give the matrix A and vector B that have the relationship
%between boundaries. M is a matrix block or stripe. c a char 'V', 'H' or
%'b', '0'

switch c
   case 'b'
      A=M.Mn;
      B=M.MD;
   case 'V'
      A=M.MSV;
      a=size(A);  %dimensions of matrix
      B=zeros(a(1),1);    %vertical stripe has not independent term
```

```
   case 'H'
      A=M.MSH;
      B=M.DPH;
   case '0'
      A=S.A;
      B=S.B;
      %otherwise
end
```

```
%-------------------------------------------------------------------------------------------
```

- giveBlock.m:

```
function [ b ] = giveBlock( c, k )
%GIVEBLOCK Gives the block k of the chip c

b=c.b(k);
```

```
%-------------------------------------------------------------------------------------------
```

- giveCell.m :

```
function [ cell ] = giveCell( B, r, c )
%GIVECELL gives a cell of block B in row r column c

cell=B.cell(r,c);
```

```
%-------------------------------------------------------------------------------------------
```

- giveGateParam.m :

```
function [ R, C, L ] = giveGateParam( cell )
%GIVEGATEPARAM show the gate parameters (from RLGC model) of the cell cell

R=cell.Param.gate.R;
C=cell.Param.gate.C;
L=cell.Param.gate.L;
```

```
%-------------------------------------------------------------------------------------------
```

- giveGlobalM.m :

```
function [ AX, BX ] = giveGlobalM( CM )
%GIVEGLOBALM give the matrix structure AX BX that has the global matrices per
% column division x of chip stored in CM.
%
%       [vi]x+1=AX[vi]x+BX
AX=CM.Ax;
BX=CM.Bx;
```

```
%-------------------------------------------------------------------------------------------
```

- giveI.m :

```
function [ i ] = giveI( RF, type, name, fi )
%GIVEI  gives a matrix with the current loop of all rows and columns of
%the RF compotnent defined by type and name at frequency index fi. type is 'b' , 'H' or 'V' %meaning
block, horizontal stripe or vertical stripe. name is an integeer with the name of %component (position
inside)

switch type
   case 'b'
      i=RF(fi).b(name).i;
   case 'H'
      i=RF(fi).Stripe.H(name).i;
   case 'V'
      i=RF(fi).Stripe.V(name).i;
   otherwise
      i='error'   %error
end
```

```
%-------------------------------------------------------------------------------------------
```

- giveInterc.m:

```
function [ Ri,li,Ci ] = giveInterc( cell )
%GIVEINTERC give the values of interconnect (from RLGC model) of the cell
%cell

%interconnects
Ri=cell.Param.interc.R;    %resistance
li=cell.Param.interc.L;    %inductance
Ci=cell.Param.interc.C;    %capacitance



%-------------------------------------------------------------------------------------------
```

- giveM.m:

```
function [ M ] = giveM ( MS, order )
%GIVEM give all the matrix struct related to the component defined in order

name=name_y(order);
p=givePlace(order);
rows=numrows(order);

switch name
   case 'V'    %vertical stripe
      M=MS.Stripe.V(p);
   case 'H'    %horizontal stripe
      M=MS.Stripe.H(p);
   case 'b'    %block
      M=MS.b(p);
   case '0'    %empty space, no M structure
      M.A=diag(ones(2*rows,1));   %nothing at that x,y, no block no strip begining.
      M.B=zeros(2*rows,1);
   otherwise
      M='error'
end

%-------------------------------------------------------------------------------------------
```

- giveMDmatrix.m:

```
function [ M, SD ] = giveMDmatrix( B )
%GIVEMDMATRIX give the matrices M and D*isw of B

M=B.M;
SD=B.SD;

%-----------------------------------------------------------------------------------------
```

- givePad.m:

```
function [ pad ] = givePad( s, n )
%GIVEPAD give the pad n of stripe s

pad=s.PAD(n);

%-----------------------------------------------------------------------------------------
```

- givePadPos.m:

```
function [ pos, side ] = givePadPos( pad, type )
%GIVEPADPOS Give the position (row or column were is the pad pad) and the
%side if the pad come from a vertical stripe
%  type is only 'V' or 'H'

if type=='V'
   pos=pad.pos.place;
   side=pad.pos.side;
else    %horizontal stripe pad
   pos=pad.pos.place;
   side=0; %there is not side concept in horizontal stripes
end


%-----------------------------------------------------------------------------------------
```

- givePadProps.m:

```
function [ R, L ] = givePadProps( pad )
%GIVEPADPROPS give the R and L values of pad (+bonding wire)

R=pad.R;
L=pad.L;

%-------------------------------------------------------------------------------------------
```

- givePlace.m:

```
function [ place ] = givePlace( yorder )
%GIVEPLACE give the place of the structure that yorder has the information

place=yorder.place;

%-------------------------------------------------------------------------------------------
```

- giveResultsTime.m:

```
function [ v, i ] = giveResultsTime( R, c, k )
%GIVERESULTSTIME Give the results of R struct for the k c block/stripe
%  c is a char 'b', 'V', or 'H' meaning the wanted result
%  k is name (position) of the c structure.
%   v, i are three-dimensional matrices with v(i,j,t) the value of voltage of
%   row i, column j at time t.
switch c
   case 'b'
      v=R.b(k).v; %voltages
      i=R.b(k).i; %currents
   case 'V'
      v=R.Stripe.V(k).v; %voltages
      i=R.Stripe.V(k).i;  %currents
   case 'H'
      v=R.Stripe.H(k).v;  %voltages
      i=R.Stripe.H(k).i;  %currents
end
```

```
%-------------------------------------------------------------------------------------------
```

- giveSHmatrices.m:

```
function [ MS, DP, vp ] = giveSHmatrices( MH )
%GIVESHMATRICES gives the representative matrices of the horizontal stripe
%and the voltage of pads.
%that allow the calculation of internal nodes

MS=MH.MS;
DP=MH.DP;
vp=MH.vp;   %pad voltage
```

```
%-------------------------------------------------------------------------------------------
```

- giveStimCurrent.m:

```
function [ i ] = giveStimCurrent( stim, type, fi )
%GIVESTIMCURRENT Give the frequencial component of switching current that
%follows the type type, at frequency index fi.

if type==0  %not switching cell
   i=0;
else
   i=stim.type(fi,type);
end
```

```
%-------------------------------------------------------------------------------------------
```

- giveStripe.m:

```
function [ S ] = giveStripe( c, type, k )
%GIVESTRIPE give the stripe k from chip c
%  Type is 'V' or 'H'
if type=='V'
   S=c.Stripe.V(k);
else
```

```
   S=c.Stripe.H(k);
end

%-------------------------------------------------------------------------------------------
```

- giveStripProps.m:

```
function [ R, L, G, C ] = giveStripProps( S )
%GIVESTRIPPROPS Give the resistance and inductance of stripe tracks (R,L).
%Give the resistance and capacitance (in series) between the vdd and gnd
%lines. S is a stripe structure.

R=S.props.R;
L=S.props.L;
G=S.props.G;
C=S.props.C;

%-------------------------------------------------------------------------------------------
```

- giveSVmatrices.m:

```
function [ Mabcd, dK ] = giveSVmatrices( MV )
%GIVESVMATRICES gives the representative matrices that allow the
%calculation of internal nodes of the vertical stripe.

Mabcd=M.abcd;
dK=M.bK;

%-------------------------------------------------------------------------------------------
```

- giveTimeAxis.m:

```
function [ t ] = giveTimeAxis( RT )
%GIVETIMEAXIS gives a vector with the time axis of the results stored in
%the chip time structure RT

t=RT.t;
```

```
%-------------------------------------------------------------------------------------------
```

- giveTrack.m:

```
function [ R, L, C ] = giveTrack( cell )
%GIVETRACK Give the parameters of the tracks for a cell (following the
%RLGC model)

R=cell.Param.track.R;
L=cell.Param.track.L;
C=cell.Param.track.C;
```

```
%-------------------------------------------------------------------------------------------
```

- giveType.m:

```
function [ type ] = giveType( cell )
% cell is a cell representation. The function gives the switching type it has.

type=cell.swtype;   %swtype=0 mean not switching cell. The switching types are defined in the %stim
stuct and are an input
```

```
%-------------------------------------------------------------------------------------------
```

- giveV.m:

```
function [ v ] = giveV( RF, type, name, fi )
%GIVEV gives a matrix with the voltage of all rows and columns of
%the RF at frequency f. type is 'b' , 'H' or 'V' meaning block, horizontal stripe or
%vertical stripe. name is an integer with the name of component (position inside)

switch type
   case 'b'
      v=RF(fi).b(name).v;
   case 'H'
      v=RF(fi).Stripe.H(name).v;
```

```
    case 'V'
        v=RF(fi).Stripe.V(name).v;
    otherwise
        v='error'   %error
end

%-------------------------------------------------------------------------------------------
```

- giveWell.m:

```
function [ Rw,Cw, Lw ] = giveWell( cell )
%GIVEWELL give the well parameters (following the RLGC model) of cell cell

Rw=cell.Param.well.R;
Cw=cell.Param.well.C;
Lw=cell.Param.well.L;

%-------------------------------------------------------------------------------------------
```

- giveXorder.m:

```
function [ x ] = giveXorder( CM,k )
%GIVEXORDER give the x field of CM struct.
%  x field has the order of the chip vertical divisions

x=CM.x(k);

%-------------------------------------------------------------------------------------------
```

- giveYorder.m:

```
function [ y ] = giveYorder( x,k )
%GIVEYORDER give the component y of ordered horizontally in x in vertical position k

y=x.y(k);
```

```
%-------------------------------------------------------------------------------------------
```

- name_y.m:

```
function [ name ] = name_y( y )
%NAME_Y say if the  component y (y-order) is a vertical stripe, horizontal
%stripe, block or empty space

name=y.name;
```

```
%-------------------------------------------------------------------------------------------
```

- nfreqs.m:

```
function [ n2, nf ] = nfreqs( freq )
%nfreqs: Say how many points have the vector freq, and how many are needed
%to can do ifft if freq is a vector with the range of frequencies of interest

n=size(freq);
n2=n(2);
nf=2*n2-1;
```

```
%-------------------------------------------------------------------------------------------
```

- numBlocks.m:

```
function [ n ] = numBlocks( c )
%NUMBLOCKS give the number of blocks that the structure c has

n=c.nblocks;
```

```
%-------------------------------------------------------------------------------------------
```

- numcells.m:

```
function [ c ] = numCells( a )
%NUMCELLS give the number of cells per row of a (stripe or block)
```

```
c=a.cells;

%-------------------------------------------------------------------------------------------
```

- numPads.m:

```
function [ npads ] = numPads( S )
%NUMPADS give the number of pads placed in stripe s

npads=S.npads;

%-------------------------------------------------------------------------------------------
```

- numrows.m:

```
function [ r ] = numrows( box )
%ROWS give the number of rows of a block or stripe

r=box.rows;

%-------------------------------------------------------------------------------------------
```

- numStripes.m:

```
function [ n ] = numStripes( c, type)
%NUMSTRIPES give the number of stripes type, that the chip c has
% type is 'V' for a vertical stripes or 'H' for horizontal
if type=='V'
   n=c.Stripe.nV;
else
   n=c.Stripe.nH;
end

%-------------------------------------------------------------------------------------------
```

- numx.m:

```
function [ nx ] = numx( CM )
%NUMX give the number of x pieces that the chip is divided
%  The criteria of division is the begining of a block/stripe structure.
%   CM is a matrix chip structure

nx=CM.nx;

%-------------------------------------------------------------------------------------------
```

- numy.m:

```
function [ ny ] = numy( x )
%NUMY give the number of structures putted in column that begin at same x
%coordinate inside the chip

ny=x.ny;

%-------------------------------------------------------------------------------------------
```

- posx.m

```
function [ x ] = posX( a )
%POSX give the position x (in column units) of a input (block, stripe or
%chip)

x=a.pos.x;
%-------------------------------------------------------------------------------------------
```

- posy.m

```
function [ y ] = posY( a )
%POSY give the position y (in rows units) of a input (block, stripe or
%chip)

y=a.pos.y;
%-------------------------------------------------------------------------------------------
```

- Y0.m:

```
function [ y ] = Y0( CM )
%Y0 give the origin of coordinates of the chip matrix structure CM

y=CM.y0;
%-------------------------------------------------------------------------------
```

A.3.4. Altres functions

En aquesta secció estan agrupades totes les funcions que no es poden classificar com a cap de les anteriors.

- obtainVpad.m:

```
function [ v ] = obtainVpad(f,type)
%give the voltage vdd in frequency domain. Type mean if f is index or
%frequency
switch type
   case 'index'
      if f==1
         v=1.2;
      else
         v=0;
      end
   case 'f'
      if f==0
         v=1.2;
      else
         v=0;
      end
end

%-----------------------------------------------------------------------------------------
```

- parallel.m

```
function [ Z ] = parallel( z1, z2 )
%parallel: take z1 and z2 as a impedance value and do the parallel of them

Z=1./(1./z1+1./z2);
```

```
%-----------------------------------------------------------------------------------------
```

- ObtainZ1Z2.m

```
function [ Z1, Z2 ] = obtainZ1Z2( B, f )
%obtain series and shunt impedances for the cells of block B at frequency f
%following the RLGC model
%  Z1 and Z2 will be matrices with the values of z1 and z2 impedances of each cell

%Block information
rows=numrows(B);    %rows of block
cols=numCells(B);   %columns of block (or cells per row)

for r=1:rows
   for c=1:cols
      cell=giveCell(B,r,c);
      %cell parameters:
      %interconnects (resistance inductance and capacitance)
      [Ri,li,Ci]=giveInterc(cell);
      %well (resistance and capacitance)
      [Rw,Cw]=giveWell(cell);
      %cell   (resistance and capacitance)
      [R,C]=giveGateParam(cell);
      %tracks (resistance and inductance
      [Rs,Ls]=giveTrack(cell);

      %impedance calculation
      if f~=0
         %track impedance Z1
         Z1(r,c)=Rs+Ls*2*pi*j*f;

         %Cell impedance Z2
         %cell capacitance
         Y1=-1/(2*pi*f*C)*j;

         %with interconnects
         Y2=Ri-1/(2*pi*f*Ci)*j+2*pi*f*li*j; %Ci Ri are in series
         YY=R+1/(1/Y1+1/Y2);     %R  is in series with the parallel of cell capacitance and interconnects
things
```

```
        %with well
        Z2(r,c)=parallel(YY,Rw-1/(2*pi*f*Cw)*j);  %well in in parallel with the other

      else    %frequency 0Hz
        Z1(r,c)=Rs;
        Z2(r,c)=j*1e122;
      end
   end
end

%-------------------------------------------------------------------------------------------------------------
```

- obtainTLvalues.m

```
function [ a, b, c, d ] = obtainTLvalues(Z1,Z2)
%For a piece of transmission line defined by Z1 and Z2 obtain the values a, b,
%c and d that [v;i]left=[a,b;c,d]*[v;i]right
% i        z1    z1      i
% -> ---==----==---  ->
%             |
% v  left   [ ]Z2   v rigth
%             |
%     ----------------

if abs(Z2)<1e20
   a=1+Z1/Z2;
   b=-(Z1*Z1/Z2+2*Z1);
   c=-1/Z2;
else
   a=1;
   b=-2*Z1;
   c=0;

end
 d=a;
```

```
%-------------------------------------------------------------------------------------------
```

- getZRLC.m:

```
function [ ZRLC ] = getZRLC( R,L,C,f )
%getZRLC obtain the value of impedance defined as a series of R L and C.
%  Negative input means that there is not this component

ZRLC=0;
if R>0
   ZRLC=ZRLC+R;
end
if L>0
   ZRLC=ZRLC+2*pi*f*L*j;
end
if C>=0
   if C==0 || f==0
      ZRLC=1e122;
   else
      ZRLC=ZRLC-1/(2*pi*f*C)*j;
   end
end
```

```
%-------------------------------------------------------------------------------------------
```

- getBase.m:

```
function [ XB ] = getBase( dim )
%getBase AB is a matrix made by adding in block diagonal the matrices M and
%in column vectors MD of [v,i]left=M[v,i]right+MD so, XB will put all v
%first and then the i by XB*AB*XB^-1+XB*MD
%
ndim=max(size(dim));    %number of submatrices we want to sort
dim2=dim./2;            %vector of dimensions of the submatrices but only v or i (not v+i as the input)
c=cumsum(dim);
r=cumsum(dim2);

XB=zeros(c(ndim),c(ndim));
```

```
I=diag(ones(dim(1)/2,1));
XB(1:r(1),1:dim2(1))=I;
XB(r(ndim)+1:r(ndim)+r(1),1+dim2(1):2*dim2(1))=I;

for ii=2:ndim
    I=diag(ones(dim(ii)/2,1));
    XB(r(ii-1)+1:r(ii),c(ii-1)+1:c(ii-1)+dim2(ii))=I;
    XB(r(ndim)+r(ii-1)+1:r(ndim)+r(ii),c(ii-1)+dim2(ii)+1:c(ii-1)+dim(ii))=I;
end

%-------------------------------------------------------------------------------------------
```

- CreateYorder.m:

```
function [ Y ] = createYorder( y0, name, place, rows )
%CREATEYORDER creates a Yorder structure. y0 is the y coordinate of down
%left of a block/stripe/empty space, name is the name of the thing we are
%ordering('b','V','H', or 0), place is the position, and rows the number of
%rows it has.

Y.y=y0;
Y.name=name;
Y.place=place;
Y.rows=rows;
```

A.4. Comparació de resultats amb simulador de circuits

Per tal de detectar errors en el codi i globalment validar el procediment, s'han realitzat diverses comparacions entre el simulador de circuits de Philips PSTAR i la implementació en matlab que s'ha fet. El model fet servir en ambdós casos és l'RLGC amb els valors per CMOS090 de l'annex C.

Els resultats d'algunes simulacions es mostren a continuació, les línies amb punts representen el resultat per matlab i les línies contínues el resultat de PSTAR:

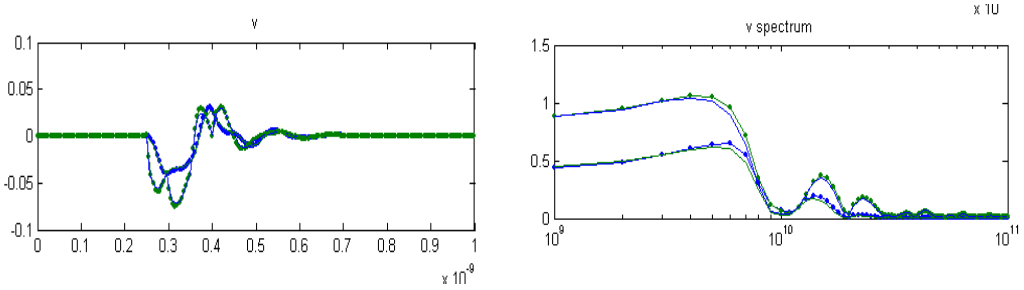

Figura A.4.1: Comparació entre PSTAR i MATALB per l'evolució temporal (esquerra) i la magnitud de l'espectre (dreta) de dues cel·les de la mateixa columna en un bloc encerclat per stripes.

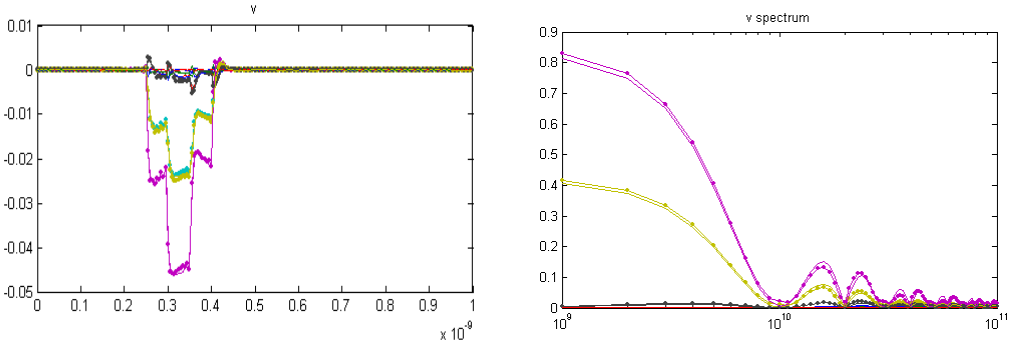

Figura A.4.2: Comparació entre PSTAR i MATALB per l'evolució temporal (esquerra) i la magnitud de l'espectre (dreta) de les cel·les de la columna central d'un bloc encerclat per stripes.

Els resultats de ambdós procediments són similars, en el procediment per matlab s'ha detectat soroll degut a la discretització en la freqüència, aquest soroll es mostra en l'evolució temporal com a unes petites oscil·lacions en els instants anteriors a una gran variació de pendent.

A.5. Referències

[1] MATHWORKS. *MATLAB functions reference*. 2006
[http://www.mathworks.com/access/helpdesk/help/techdoc/ref/ref.html]

B. Manual d'usuari

B.1. Executant el programa

Una vegada oberta la finestra de matlab 7 o matlab 6.5 s'ha d'anar al directori on estan totes les funcions fitxer de l'annex B, per executar el programa s'ha d'escriure en el nom del fitxer mare *Av_chip* en la finestra principal i prémer la tecla intro.

Abans d'executar és necessari tenir definit els inputs a les funcions fitxer *getInputs.m* *getstim.m*.

B.2. Creant inputs

Els inputs es creen en les funcions fitxer *getInputs.m* i *getstim.m*, en la primera es defineix la configuració tant espacial com de components del xip. A la segona es defineixen totes les dades relacionades amb els estímuls i la discretització temporal.

- Com definir un xip

Per definir un xip és necessari modificar la funció fitxer getInputs.m

```
      function [ c ] = getInputs
%Definition of chip. Creation of chip data structure.
```

El xip objectiu que es vulgui analitzar ha d'estar dividit en blocs, stripes verticals i stripes horitzontals (figura A.4.1). Un stripe vertical és un stripe o una part de l'anella d'alimentació (ring) de direcció perpendicular a les files dels blocs. Un stripe horitzontal es un stripe o una part de la anella d'alimentació que segueix la direcció de les files dels blocs.

La connexió amb el mon exterior es fa a través de pads, els pads es amb els stripes.

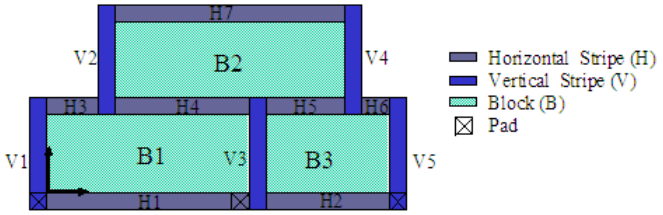

Figura B.2.1: Exemple de configuració de xip

Per connectar dos components d'un xip només és necessari situar-los en l'espai un al costat de l'altre, les connexions es fan en la direcció horitzontal no en la vertical.

Dos passos fan falta per crear un xip. Primer inicialitzar una nova estructura *chip* i segon afegir-hi components. Pel xip de la figura A.4.1 la creació podria ser:

```
% B1, B2, B3 are blocks, V1, V2, V3, V4, V5, H1, H2, H3, H4, H5, %
H6, H7 are stripes

c=inichip(0);  %inicialize the chip (put to 0 fields)

% number of cells per row of the blocks
colsB1=numcells(B1);
colsB2=numcells(B2);
colsB3=numcells(B3);

% number of cell divisions of horizontal stripes
colsH3=numcells(H3);
colsH4=numcells(H4);
colsH5=numcells(H5);
colsH6=numcells(H6);

% number of rows of B1 and B2
rowsB1=numrows(B1);
rowsB2=numrows(B2);

% addition of the first block B1
% because it is the first we need to define the reference of
% coordinates
x0=0; %origin of coordinates
```

```
y0=0;
c=addBlock(c,B1,x0,y0); %this will be the block 1

% addition of the vertical 'V' stripe V1 in x,y coordinates
% inside chip its left down corner it is placed one row down and %
one column left from the left down corner of B1 origin of
% coordinates
x0=x0-1;
y0=y0-1;
c=addStripe(c, V1, x, y,'V');

% addition of the horizontal 'H' stripe H1
% it is placed below B1
x=x0;
y=y0-1;
c=addStripe(c,H1,x,y,'H');

% addition of the third vertical stripe V3
% it is placed next to B1, its down corner is one row below y0
x=x0+colsB1;
y=y0-1;
c=addStripe(c, V3, x, y,'V');

% addition of the horizontal 'H' stripe H2
x=x0+colsB1+1;    %a vertical stripe has one column width
y=y0-1;
c=addStripe(c,H2,x,y,'H');

% addition of the third block
x=x0+colsB1+1;
y=y0;
c=addBlock(c, B3, x, y);

% addition of the vertical stripe V5
x=x0+colsB1+1+colsB3;
y=y0-1;
c=addStripe(c, V5, x, y,'V');

% addition of the vertical stripes V2 and V4
% all placed above B1 and B3
```

```
xV2=x0+colsH3;
xV4=x0+colsH3+1+colsB3;
y=y0+rowsB1;

c=addStripe(c, V2, xV2, y,'V');
c=addStripe(c, V4, xV4, y,'V');

% addition of the second block
x=x0+colsH3+1;
y=y0+rowsB1+1;    %a horzontal stripe has 1 row width
c=addBlock(c, B3, x, y);

% addition of the horizontal stripes H3, H4, H5 and H6
% all are placed above B1 and B3
y=y0+rowsB1;
xH3=x0;
xH4=x0+colsH3;
xH5=y0+colsH3+1+colsH4;
xH6=y0+colsH3+1+colsH4+1+colsH5;

c=addStripe(c,H3,xH3,y,'H');
c=addStripe(c,H4,xH4,y,'H');
c=addStripe(c,H5,xH5,y,'H');
c=addStripe(c,H6,xH6,y,'H');

% addition of H7
% Placed above B2
x=x0+colsH3+1;
y=y0+rowsB1+1+rowsB2;
```

- Com definir un bloc

De la mateixa manera que en el cas del xip, són necessaris dos passos per crear un bloc, primer inicialtizar un nou i, segon afegir-hi cel·les.

Una manera d'inicialitzar un bloc és a través de conèixer la seva longitud i utilitzar la funció *createBlockLength*

```
rows=50;    %number of rows
```

```
fu=0.9;   % reference gates space occupation percentage
length=240;    %[um] length of the rows
b=createBlockLength(rows,length,fu); %b is a block structure
```

una altra manera és a través de conèixer el nombre de cel·les de referència per fila amb la funció createBlockCell,

```
rows=50;    %number of rows
fu=0.9;   % reference gates space occupation percentage
cells=250;    %[um] length of the rows
b=createBlockCell(rows,cells,fu); %b is a block structure
```

en ambdós casos és necessari conèixer el factor d'utilització, percentatge de la fila utilitzat per les cel·les de referència. La traducció a d'una fila a cel·les de referència es fa en (resistència, capacitància, …) mantenint la longitud standard d'una cel·la de referència (mitjana entre una cel·la nor i una nand), cel·les complexes poden estar més optimitzades en l'espai que una simple cel·la nand-nor. El factor d'utilització corregeix aquests efectes sobre la longitud total de la fila.

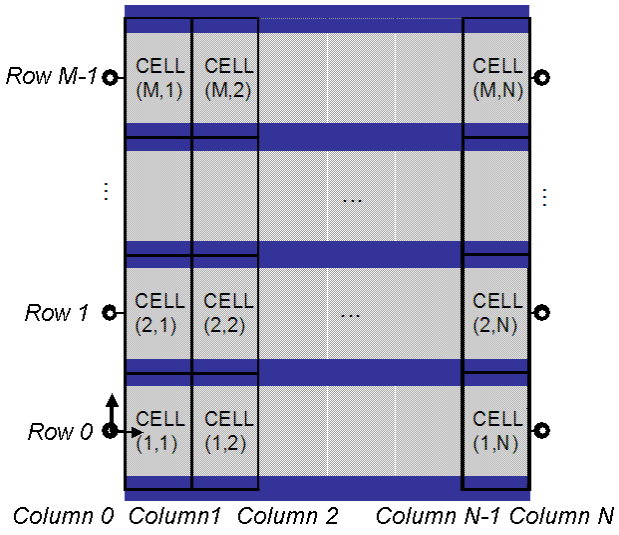

Figura B.2.2: Exemple de la configuració d'un bloc.

La segona part de la definició de bloc és l'addició de cel·les. Per afegir una cel·la primer s'ha de crear una mitjançant les seves propietats i el seu tipus de corrent de commutació. Les propietats elèctriques per a diferent tipus de cel·les estan definides dins les funcions

getTrackParam(), *getWell()*, *getInterconnect()* i *getGate()* que retornen variables amb propietats.

El següent exemple mostra com s'afegeixen cel·les de tipus 'reference' amb valors de propietats per a la tecnologia Philips CMOS090. S'utilitzen tres tipus de corrent de commutació (tipus 0, 1 i 2) distribuïts homogèniament (els tipus de corrent de commutació es defineixen a la funció getStim.m, però el tipus 0 sempre correspon a la cel·la que no commuta).

```
activity=0.1 %percentage of switching cells

%cells construction
%parameters
tracks=getTrackParam(length,numCells(b));
well=getWell(length,numCells(b));
interc=getInterconnect('philips');
gate=getGate('reference');

%switching cells
i=0; %activity counter
swtype=0;
for rb=1:rows
  for cb=1:numCells(b)
      switch i
         case  activity*100
              swtype=1;   %switching cell type 1
         case  2*activity*100
              swtype=2;   %switching cell type 2
              i=0;
      end
      i=i+1;
%creation of cell
      cell=createCell(tracks,well,interc,gate,swtype);
% addition of cell to block b in column cb row rb
      b=addCell(b,cell,rb,cb);
      swtype=0; %non switching
  end
end
```

- Com definir stripes

Es pot crear un stripe usant la funció *createStripe(props, size, type)*. Abans de fer-ho es necessari tenir una variable amb les propietats de cel·la (*props*) saber la longitud (en columnes de cel·les per al cas horitzontal o files de cel·les en el cas vertical). *Type* és el caràcter 'V' o 'H' depenent si és un stripe horitzontal o vertical.

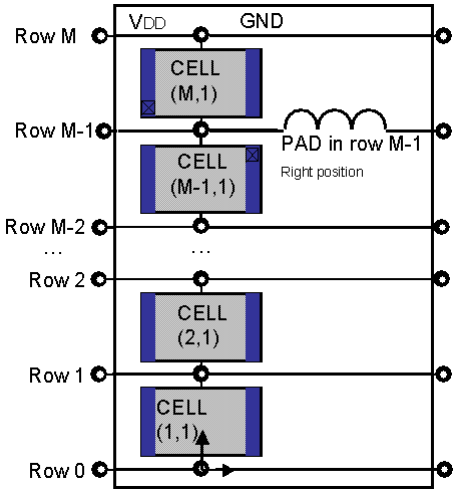

Figura B.2.3: Exemple de la configuració d'un stripe vertical.

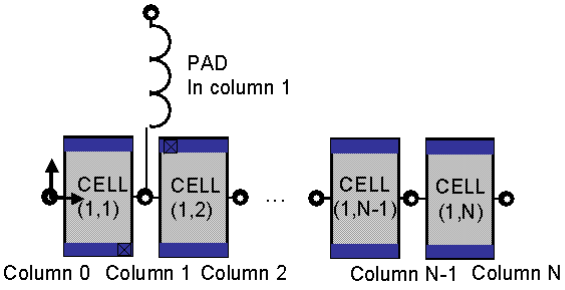

Figura B.2.4: Exemple de configuració d'un stripe horitzontal

Les propietats d'una cel·la d'stripe es poden crear amb les funcions *getStripVprops* per a un stripe vertical o *getstripHprops* per l'horitzontal.

Un stripe pot tenir localitzat un pad que connecti amb l'exterior. Per crear un pad es pot fer servir les funcions *makePADV* pels stripes verticals o *makePADH* pels horitzontals.

És important notar que quan es crea un pad de fet s'estan creant simultàniament el pad de V_{DD} i el de *GND* suficientment a prop per considerar simètriques les línies de V_{DD} i de *GND*.

El següent exemple mostra com es crea un stripe vertical amb un pad situat a la meitat en el costat esquerre.

```
%VERTICAL STRIPE
%left down corner
y=-1; %y coordinate (in rows unit)
x=-1; %x coordinate (in cells unit)

%stripe properties
Props=getStripVProps(0); %Properties are resitance, inductance,
%capacitance and reactance of the stripes

rows=12; %number of rows that the stripe will connect
SV=createStripe(Props,rows,'V'); %creation stripe

%Creation of the pad
%pad properties
Rpad=0.01;  %[ohm]
Lpad=0*0.1e-9;    %[H]

P=makePadV(6,Lpad,Rpad,'l'); %pad in row 6, in 'l' left side
SV=addPad(SV,P);    %add pad to the stripe
```

- Com definir estímuls

Per definir un conjunt de tipus de corrent de commutació és necessari modificar la funció fitxer getStim.m

```
function [ stim ] = getstim
%GETSIM will create the types of switching currents in frequency
%domain and will store all frequency data in the stim struc
```

L' objectiu d'aquesta funció es la de definir la resolució temporal de la solució així com guardar les formes d'ona dels corrents de commutació en el domini de la freqüència. El

següent exemple mostra com crear una forma d'ona de corrent per cel·les de tipus 1, a partir d'un graó en el domini del temps.

```
Tend=3e-9;      %[s] Period (maximum time of all transitions)
Ts=10e-12;      %[s] Sample time
% Initialization and creation of a stimuli structure
stim=createStim(Tend,Ts);   %create the struct stim of Tend %lenght
with a sample of Ts

%creation of a step in frequency domain
type=1;     %NAME (associated to a kind of switching cell)
%                      ___________
% ____________________|           |______________
%0                  t1          t2           Tend
t1=250e-12; %[s]
t2=350e-12; %[s]
sw1=gen_step(t1,t2,Ts,Tend); %sw1 has the frequency components %of
the step. F is the frequency axis
%sw1 is a vector row with the frequency components of the step

stim=addType(stim,type,sw1);     %add the current description in
%frequency domain to for a type of cell.
```

B.3. Entenent l'output

Després d'executar el fitxer *Av_chip.m* queden unes variables en el *workspace*, els resultats en el domini del temps estan en la variable *RT*.

- Com llegir resultats

Per llegir resultats es pot fer servir la funció *giveResultsTime(RT,char,k)* per als resultats component *k*-èssim del xip descrit a *char* ('b' per a blocs,'H' per a stripes horitzontals i 'V' per a stripes verticals). Aquesta funció retornarà dos variables, una amb l'evolució dels nivells de tensió i una altra amb l'evolució del corrent del component del xip.

Cada variable retornada és una matriu tridimensional, la primera i segona dimensions donen la fila i columna de la cel·la del element demandat a *giveResultsTime*, la tercera dimensió és el temps (per més informació veure com utilitzar la funció *giveResultsTime* en A.2.6).

El següent exemple mostra com dibuixar el gràfic del mínim nivell de tensió registrat en el tercer bloc d'un xip de tres blocs després d'executar *Av_chip.m*.

```
% RT is the result of Av_chip for a chip with 3 blocks
[ v, i ] = giveResultsTime( RT, 'b', 3 );
%v and i are three-dimensional matrices with v(i,j,t), i(i,j,t)) %the
value of voltage for row i, column j at time index t of %the block 3

%We will use min() function (standard of matlab)to obtain the minimum
value of v following the 3 dimension, [] activate this function.

%vmin=abs(min(v,[],3));

Figura      %creation of the Figura windows
surf(vmin); %create of a surface with the values of vmin
colormap(hot)     %definition of the colors of the map
caxis([.87,1.2])  %maximum and minimum values
colorbar;   %show the color legend
view([0,90]);     %projection to the x-y axis
```

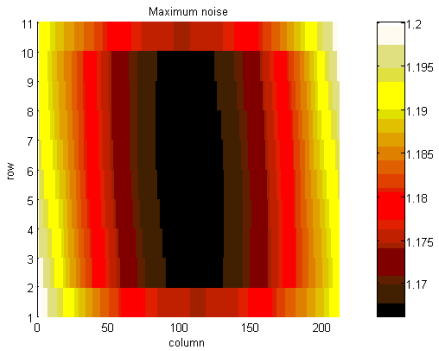

Figura B.3.1: Figura resultant de l'apicació del codi exemple.

- Significat dels resultats

Els resultats finals per a un element (bloc, stripe vertical o stripe horitzontal) són les matrius tridimensionals amb voltatges i corrents. Cada matriu de voltatges conté l'evolució temporal del nivell de tensió entre les alimentacions locals V_{DD} i *GND* de tots els contorns de les cel·les de l'element (figura A.4.6).

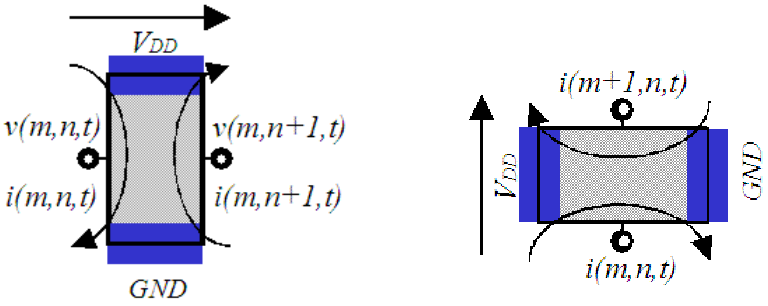

Figura B.3.2: Contorn de les cel·les per a un bloc o stripe horitzontal (esquerra) o stripe vertical (dreta).

La matriu de corrents conté l'evolució temporal del llaç de corrent entre les alimentacions locals V_{DD} i *GND* de tots els contorns de les cel·les de l'element.

El signes per als corrents segueix la convenció de positiu quan el llaç de corrent entre V_{DD} i *GND* segueix les agulles del rellotge. Si es considera només la xarxa de V_{DD}, el signe positiu és equivalent a quan el corrent va d'esquerra a dreta per als blocs i stripes horitzontals; i de dalt a baix per als stripes verticals.

És important notar que si es vol saber el corrent total per un conductor d'alimentació s'ha de sumar els valors dels llaços de corrent de les cel·les que comparteixen conductor.