

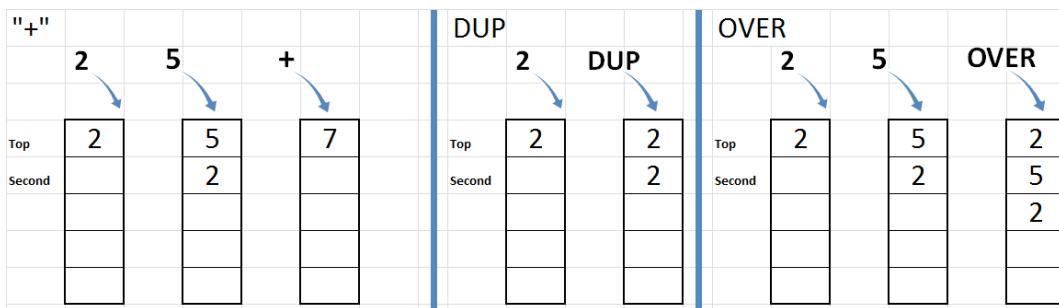
Coup de Forth dans le monde parallèle de l'embarqué !

Emmanuel SAID

Il existe des chips appelés « machine Forth » qui ont la particularité d'avoir le langage implémenté directement dans le silicium. Ce que je souhaite vous faire partager c'est la mise en pratique du chip GA144 de chez GreenArrays, une taille minimale 1 cm² pour juste 144 processeurs ! Bienvenue dans le monde parallèle !

1. Le langage Forth

Comment introduire le chip GA144 sans commencer à introduire le langage Forth ? En effet le langage Forth inventé par Chuck Moore dans les années 70, a subi différentes évolutions pour aboutir à une maturité et permettre d'implémenter le langage dans le silicium. Toute la logique de Forth repose sur la manipulation de données sur une « pile »; une pile comme une pile d'assiette ! Pour effectuer une opération d'addition, il suffit alors d'empiler les 2 nombres ainsi que l'opérande d'addition « + » et récupérer le résultat au sommet de la pile.



/// Image : pile.png ///

Fig. 1 : Exemple d'addition de 2 nombres, on récupère le résultat au sommet de la pile. Une autre fonction DUP, copier le sommet de la pile. OVER est intéressant il permet de copier le second élément sur le haut de la pile.

/// Fin légende ///

C'est un langage surprenant, un peu déroutant au début mais qui a les qualités de pouvoir être facilement implémenté (que ce soit en hard ou soft).

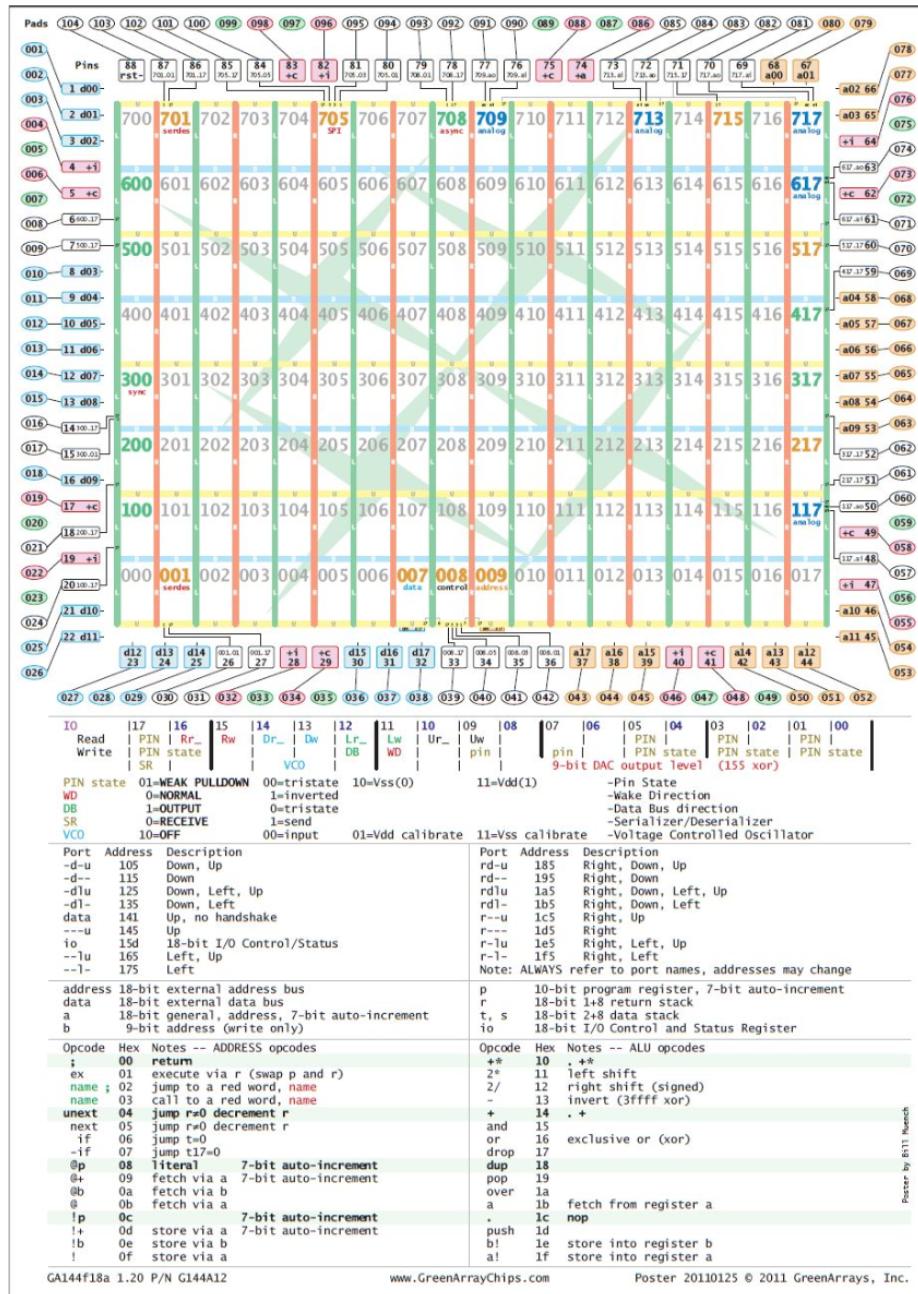
2. Le chip GA144

Nous pouvons aborder sereinement et découvrir le chip GA144. Il renferme 144 processeurs F18A repartis en lignes et colonnes de 8*18, et chacun des processeurs ont 32 instructions plus ou moins complexe qui recouvrent les différentes familles dites « arithmétiques, d'accès mémoire, de contrôle ».

Ce multiprocesseur a la possibilité de fonctionner en mode parallèle, c'est à dire que chacun de ses

processeurs (F18A) sont indépendant les uns des autres, ou en mode pipeline (communication interne entre les processeurs).

Il consomme très peu (7 pico joules / instruction) dû au fait que les processeurs sont du type asynchrone (pas d'horloge, oui ça existe ...), et nous arrivons à une vitesse d'exécution de l'ordre de 666 MIPS * 144 (rien que ça !). Il est doté d'entrées sorties numériques, de convertisseurs ADC et DAC sur 9 bits, de bus SPI, de communication série, de l'interface vers une mémoire extérieure (SRAM). Cerise sur le gâteau les primitives du langage sont implémentées directement au cœur du F18A. Le F18A est un ordinateur à part entière, c'est une machine 18 bits, avec sa propre RAM et ROM (64 mots de 18 bits), piles et registres, entrées sorties.



/// Image : poster.png ///

Fig. 1 : Le GA144 vu de l'intérieur, 8*18 processeurs, 5 convertisseurs ADC, DAC, des entrées sorties logiques, accessibles sur ses 88 pins, pour 32 instructions Forth, et une consommation si faible pour une vitesse si grande.....

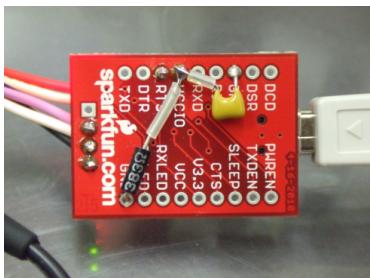
/// Fin légende ///

3. Communiquer avec le GA144

La communication avec le GA144 se concrétise par le biais d'une liaison série. Pour rappel les niveaux de tensions acceptés par le chip sont de 1.8v. De même, le GA144 a la particularité d'avoir les lignes Rx et Tx en logique positive (au repos, à l'état logique '0' correspond à un niveau de tension 0v, et l'état logique '1' correspond à 1.8v). La ligne RTS servira au reset du GA144, elle sera active au niveau bas.

Pour la mise en pratique, le PC communique par le biais du module USB série (à base du célèbre composant FT232RL), celui-ci peut être reprogrammé et configuré pour devenir compatible autant au niveau logique, qu'au niveau de tension de 1.8v (il est important de respecter cette contrainte pour ne pas endommager le GA144). Pour ma part, je préconise d'insérer une résistance (environ 400 ohms) et une capacité de découplage (100nF) sur le module FT232 entre la broche VCCIO et la broche GND.

Ici le module FT232 choisi est le module de chez Sparkfun (<http://www.sparkfun.com/products/718>), mais tout autre module à base de FT232 devrait faire l'affaire.



/// Image : resistance_vccio.png ///

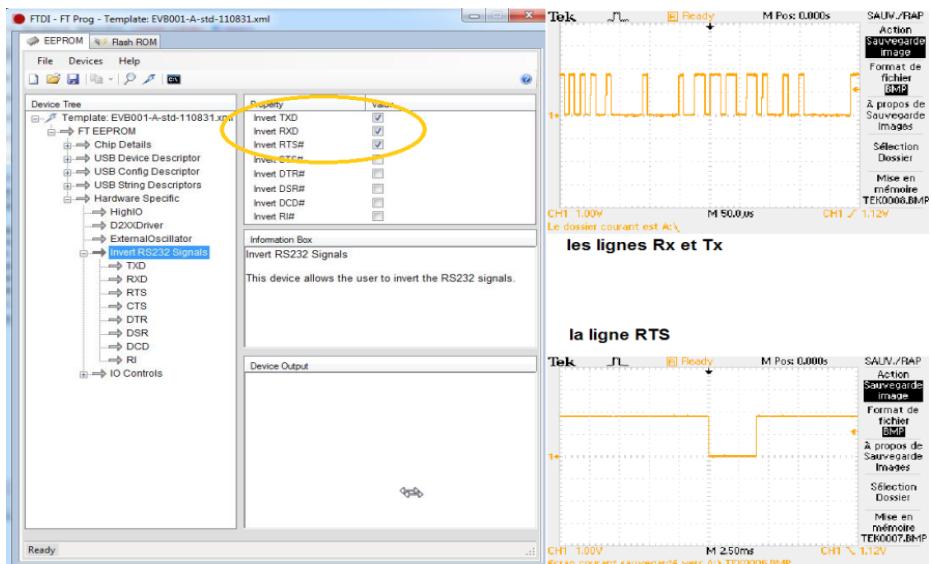
Fig. 1 : Le module FT232 est compatible avec les niveaux de tensions de 1.8v du GA144, mais il est souhaitable d'insérer une résistance (environ 400 ohms) entre la broche VCCIO et GND et une capacité de découplage de 100nF.

/// Fin légende ///

Le module FT232 est presque opérationnel :

Il faut le reprogrammer avec les paramètres du fichier de GreenArrayForth : « C:\Program Files\GreenArrays\EVB001\FTDI\EVB001-A-std-110831.xml ».

Pour reprogrammer l'EEPROM du module, il faut utiliser le programme fourni par le constructeur ftdchip : « FT_PROG 2.6.8 » (http://www.ftdichip.com/Support/Documents/AppNotes/AN_124_User_Guide_For_FT_PROG.pdf)



/// Image : Rx_Tx_RTS_Config.png ///

Fig. 1 : Une fois l'EEPROM programmée, le module FT232 est compatible avec les niveaux logique du GA144, Rx et Tx et RTS sont inversés (la case est cochée).

/// Fin légende ///

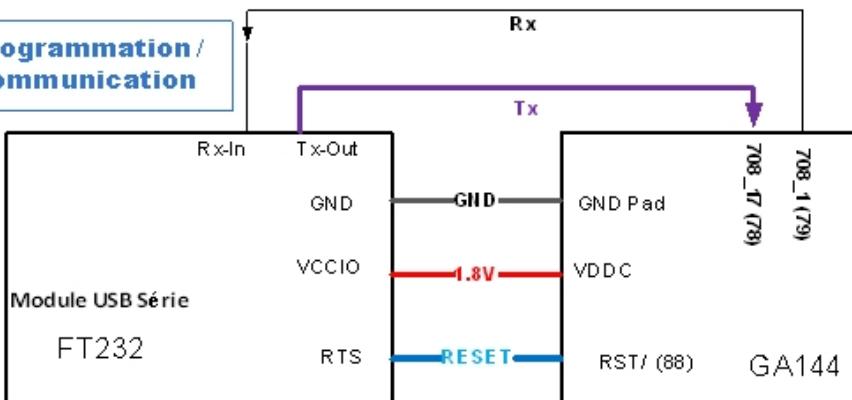
4. Le kit GA144

J'ai hâte de pouvoir mettre en pratique ce chip, mais comment ? Il est possible d'acheter un lot de 10 chips GA144 auprès du fabricant, ou alors leur kit d'évaluation (qui renferme 2 GA144). Pour ma part, j'ai choisi la solution de développer une carte d'expérimentation, bien moins onéreuse, et ayant les mêmes fonctionnalités. Ce kit a la particularité d'exploiter 2 modes de fonctionnement :

- en mode de communication directe avec les processeurs du GA144 (permets de programmer et exécuter le code via l'IDE ArrayForth). Ce mode permet de profiter pleinement de la puissance du chip.
- en mode de communication eForth : on utilise le programme résident dans la mémoire Flash, qui exploite la SRAM, et communique via une liaison série vers un terminal. Nous avons dans ce cas un système Forth à part entière.

START GA144 KIT

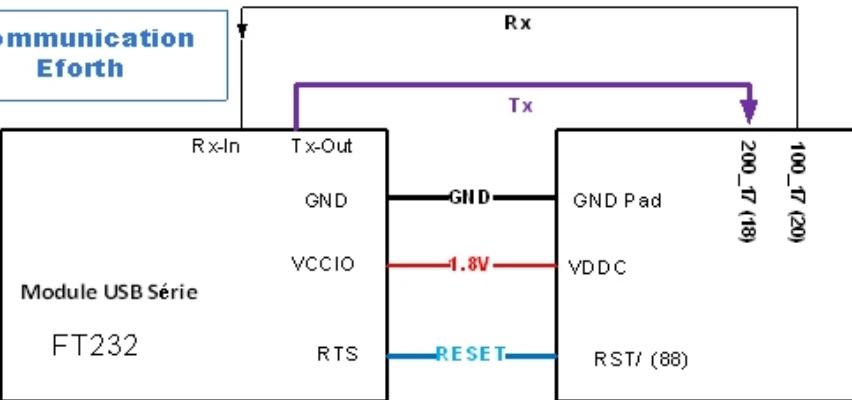
Programmation / communication



PLUG FLASH JUMPER



Communication Eforth



UNPLUG FLASH JUMPER

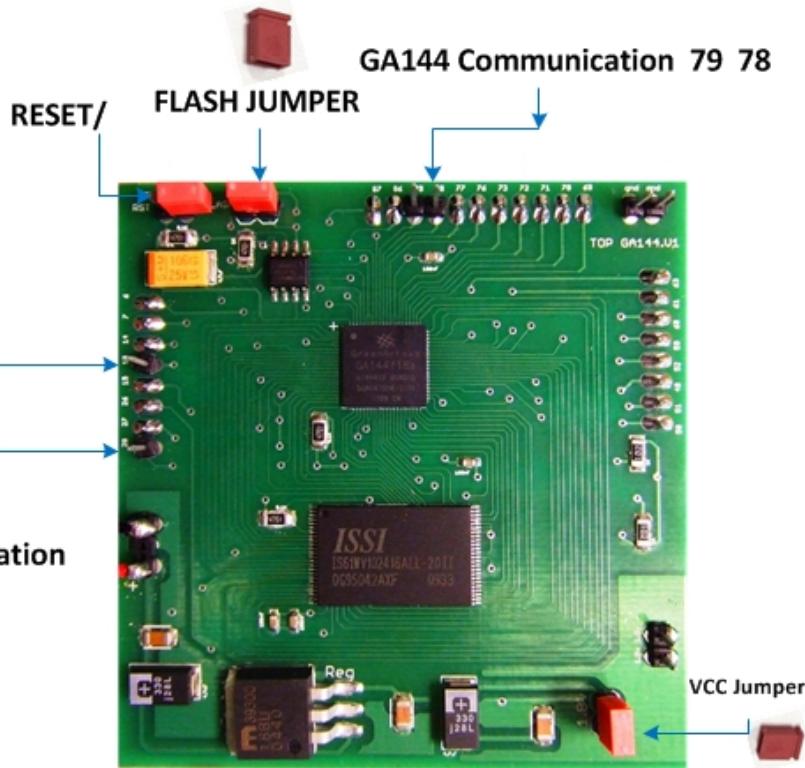


Name Pin

GPIO-600.17	6
GPIO-500.17	7
300.17-CLK	14
300.1-DATA	15
200.17-RX	18
100.17-TX	20
001.1-DATA	26
001.17CLK	27
117.AI	48
117.AO	50
GPIO-217.17	51
GPIO-317.17	52
GPIO-417.7	59
GPIO-517.17	60
617.AI	61
617.AO	63
717.AI	69
717.AO	70
GPIO-715.17	71
713.AO	72
713.AI	73
709.AI	76
709.AO	77
708.17-RX	78
708.1-TX	79
701.17CLK	86
701.1-DATA	87

Eforth Communication
20 18

Alimentation
9V



/// Image : configuration_selftest.png ///

Fig. 1 : 2 modes de fonctionnement, eForth pour utiliser le GA144 avec une mémoire externe, permet d'apprendre le langage Forth et le mode direct avec le GA144 pour programmer et utiliser pleinement les processeurs.

/// Fin légende ///

Le plus simple pour démarrer et d'utiliser eForth, il est juste nécessaire d'avoir un terminal.

Pour ma part, pour tester les quelques exemples fournis par GreenArrays, je conseille d'utiliser le soft SwiftForth. Après avoir téléchargé le fichier son installation et sa configuration sous Linux ne nécessite pas de commentaire particulier.

```
tar xzf SwiftForth-linux-osx-eval.tgz
cd /bin
ln -s ~/SwiftForth/bin/linux/sf
```

Pour utiliser les exemples de eForth, il est conseillé d'installer SwiftForth (évaluation) :
<http://www.forth.com/swiftforth/dl.html> et créer les liens symboliques.

Maintenant on lance le script « term.sf.sh » (merci SwiftForth), le menu suivant apparait.

Je fais manuellement un reset de la carte GA144 (le pont RESET/ est inséré puis enlevé), un appui sur la touche Escape et S pour afficher le « status » de la communication et ensuite la touche « Espace », et le GA144 synchronise la communication, c'est gagné le « message eForth » s'affiche !

```
~/arrayforth_12_9_2011/GreenArrays/EVB001/eF/term$ ./term.sf.sh
*-----+
| Terminal           Escape and...   |
+-----+
| H Help            || S Status      |
| P Pacer load     || R Re-load    |
| E Edit [file]    || L List files  |
| D Display path   || C Change path |
| U Uppercase       || V Verbosity   |
| X eXit           || A App load    |
+-----+
| Host              T Terminal     |
+-----* ok
ok
t 115200 comport /dev/ttyUSB0 CharDelay=3 LineDelay=111 upper dots
eForth btc:20110826 ga144:20110826
ok
```

Les tours de Hanoi, vous connaissez ?

« ESC A », pour charger les applications, et HANOI pour lancer le programme dans le GA144.



/// Image : hanoi.png ///

Fig. 1 : le célèbre jeu de réflexion, les tours de Hanoi.

/// Fin légende ///

Qui veut jouer à Tétris ?



/// Image : tetris.png ///

Fig. 1 : le célèbre jeu Tetris, par la simple commande TETRIS

/// Fin légende ///

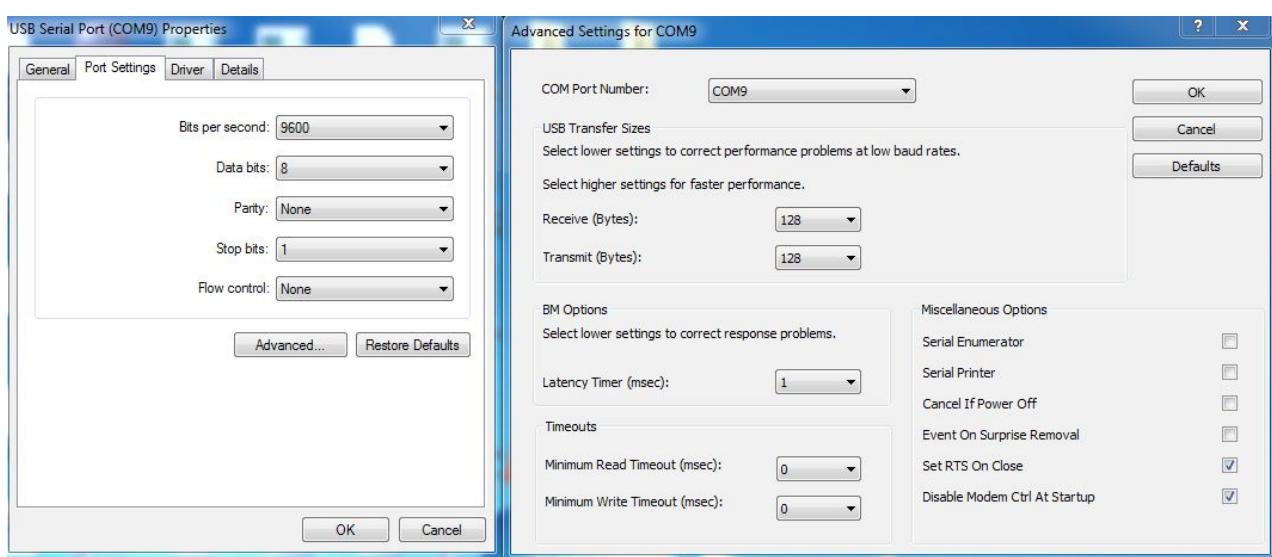
5. L'installation de l'IDE :

L'application ArrayForth est à l'image du chip GA144, si différente de ce qui existe, mais demande une gymnastique intellectuelle pour maîtriser l'outil. L'IDE permet d'écrire du code, simuler chacun des processeurs (très utile pour étudier le code), programmer le chip... c'est une application complète. ArrayForth fonctionne sur différentes plateformes : windows, Linux (via Wine) et Mac. Une documentation complète est disponible <http://www.greenarraychips.com/home/support/index.html>.

Installation d'ArrayForth :

Après avoir téléchargé et installé « arrayForth Release 01g », il suffit de suivre simplement les différentes étapes selon une installation sous Windows ou Linux, Mac. Sous Linux il vous faut lancer l'application par le biais de « Wine » (émulateur Windows).

Sous Windows il faut configurer la communication USB série. Il faut noter le numéro du port (ici COM9).



/// Image : configuration_portUSB.png ///

Fig. 1 : Les réglages de la liaison USB série avec un setting de 128 octets (USB Transfer sizes), et un temps de latence de 1ms permettent une vitesse accrue avec le chip GA144. Cochez les cases RTS (niveau logique à '1' pour éviter un reset permanent sur le GA144), et la case disable modem.

/// Fin légende ///

Sous Linux, après avoir inséré le module USB série, il faut configurer notre nouvelle liaison : « dmesg » permet de trouver notre module. Et on crée un lien symbolique vers le port com9 dans le répertoire `~/.wine/dosdevices`.

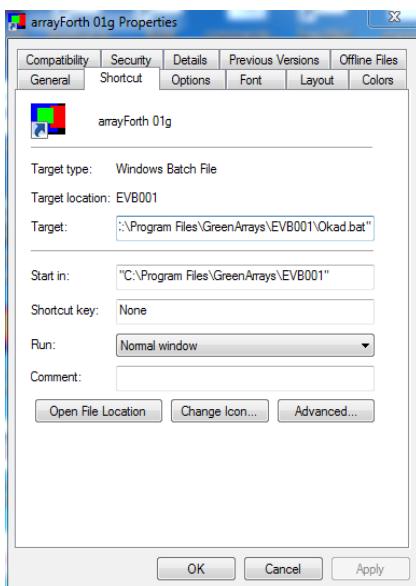
```
dmesg | grep ttyUSB0
```
usb 2-2: FTDI USB Serial Device converter now attached to ttyUSB0
ftdi_sio ttyUSB0: FTDI USB Serial Device converter now disconnected from ttyUSB0
usb 2-2: FTDI USB Serial Device converter now attached to ttyUSB0 ```

cd ~/.wine/dosdevices
ln -s /dev/ttyUSB0 com9
ls -l
```
com9 -> /dev/ttyUSB0 ````
```

Seulement sous Windows (sous Linux les paramètres sont déjà définis), il faut éditer et configurer dans le répertoire « C:\Program Files\GreenArrays\EVB001 » le fichier OKAD.BAT.
On enlève « rem » et on modifie le numéro du port com correspondant au module USB (ici 9) sur la première ligne.

```
mode com9 baud=115200 parity=n data=8 stop=1
rem mode com10 baud=921600 parity=n data=8 stop=1
Okad2-42c-pd.exe
rem howdy
```

Maintenant nous pouvons créer le raccourci vers okad.bat



/// Image : ArrayForth_target.png ///

Fig. 1 : Le raccourci ArrayForth permet de lancer le logiciel via OKAD.BAT pour prendre en compte la liaison série.

/// Fin légende ///

Après le lancement du logiciel ArrayForth, via son raccourci, nous avons encore un réglage à faire dans le logiciel lui-même (le dernier promis !) le numéro du port et de la vitesse (maximum 921600bps).

Pour éditer le bloc 202 et configurer le port de communication :

taper **202** ‘barre Espace’ **edit** ‘barre Espace’

Utiliser les touches **[J][K][L][;]** pour naviguer dans le code (voir rubrique « déplacement du curseur »).

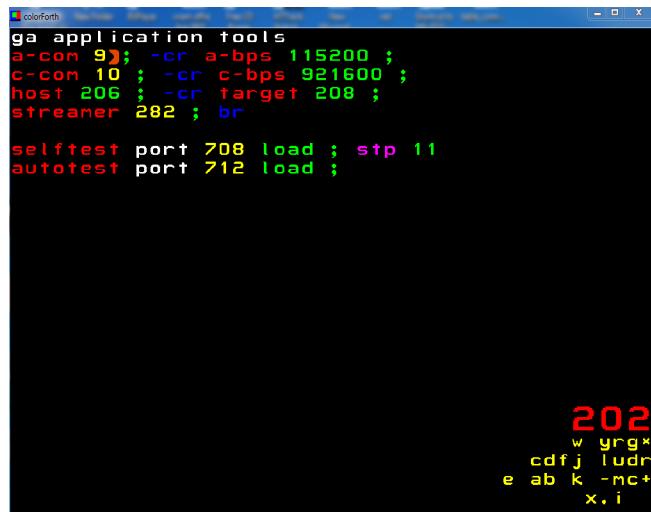
Pour supprimer le caractère avant le curseur, utiliser la touche **[N]**

Pour éditer et écrire en jaune le numéro de port 9 :

utiliser la touche **[U]** taper **9** ‘barre Espace’

Utiliser la touche **[Escape]** pour sortir du mode édition, et ‘barre Espace’ pour valider

taper **save** ‘barre Espace’ et taper **bye** ‘barre Espace’ pour sortir.



/// Image :bloc_202.png ///

Fig. 1 : Il faut configurer le port de communication (ici 9) dans le logiciel ArrayForth (bloc 202).

/// Fin légende ///

Test de la communication avec le GA144 et ses processeurs :

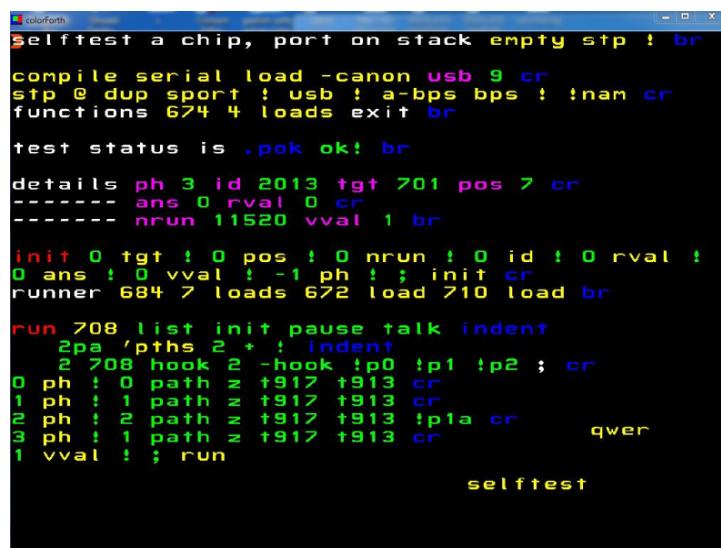
Nous sommes enfin prêt, maintenant nous allons communiquer avec notre chip (un dernier contrôle, on est bien dans le mode communication avec le GA144, module USB configuré et connecté, pont « Flash jumper » inséré), nous pouvons lancer le programme « ArrayForth » en cliquant sur le raccourci.

L'utilitaire « selftest » permet de s'assurer que la communication avec le chip est valide, et de tester les 144 processeurs.

Le lancement de « selftest » :

taper **a-com 'barre Espace'** **selftest 'barre Espace'** **run 'barre Espace'**

Au bout de quelques minutes (dépendant de la vitesse du port USB série), on a le message ok !



/// Image :selftest_ok.png ///

Fig. 1 : « selftest » permet de s'assurer que la communication avec le chip est correcte, et que les processeurs sont opérationnels.

/// Fin légende ///

6. L'IDE ArrayForth :

Commandes de l'éditeur ArrayForth

Au premier abord, l'utilisation paraît assez déroutante, mais après quelques manipulations, une fois que l'on maîtrise l'outil, la navigation (d'un bloc à l'autre), l'édition du code est assez efficace. Les couleurs dans le code ont une signification (voir les exemples de code dans les blocs). Pour utiliser efficacement l'IDE, il est recommandé de posséder un clavier QUERTY, en effet la disposition des touches est optimisée dans ce sens (sinon il faut dans le cas d'un clavier AZERTY retrouver la correspondance des touches, voir utiliser un clavier virtuel par exemple « kvkbd » sous linux). Une astuce qu'il est utile de connaître pour passer d'un clavier azerty et vice-versa vers qwerty, sous linux la commande « setxkbmap fr » ou « setxkbmap us » respectivement.

```
setxkbmap fr  
setxkbmap us
```

Il est nécessaire de comprendre la disposition des commandes par rapport au clavier. Entre crochets [] correspond la touche réelle du clavier QWERTY et entre guillemet « » la touche virtuelle. Découvrons la signification des touches.

En mode édition :

« **w** » [R] entrer le texte couleur blanc (commentaire) comme « White »
« **y** » [U] entrer un texte couleur **Jaune** (interprété) comme « Yellow »
« **r** » [I] entrer un mot couleur **Rouge** (définition d'un mot) comme « Red »
« **g** » [O] saisir du texte couleur **Vert** (compilé) comme « Green »
« **a** » [Z] entrer un mot couleur **Grise** comme « Gray »
« **b** » [X] entrer un mot couleur **Bleu** comme « Blue »
« **m** » [<] saisir du texte couleur **Magenta** (variables) « Magenta »
« **c** » [>] saisir du texte couleur **Cyan** « Cyan »

« **c** » [A] change la couleur du mot avant le curseur : **blanc - Jaune - Vert**
« **X** » [N] supprimer le mot courant (couper pour coller)
« **i** » [Alt] Insérer mot (coller)
« **k** » [V] copie (similaires à "x" couper mais ne supprime pas le mot)

La navigation d'un bloc à l'autre :

864

« **x** » [P] basculement entre bloc pair (code) et impair (commentaires)
« **-** » [M] aller au bloc précédent
« **+** » [?] aller au bloc suivant

Le déplacement du curseur dans le code :



« **l** » [J] déplacer le curseur à gauche comme « Left »
« **u** » [K] déplacer le curseur vers le haut comme « Up »
« **d** » [L] déplacer curseur vers le bas comme « Down »

« **r** » [:] déplacer le curseur à droite comme « Right »

« . » [barre Espace] **Valider la commande.**

Touche « **Esc** » : **sortir** du mode actuel.

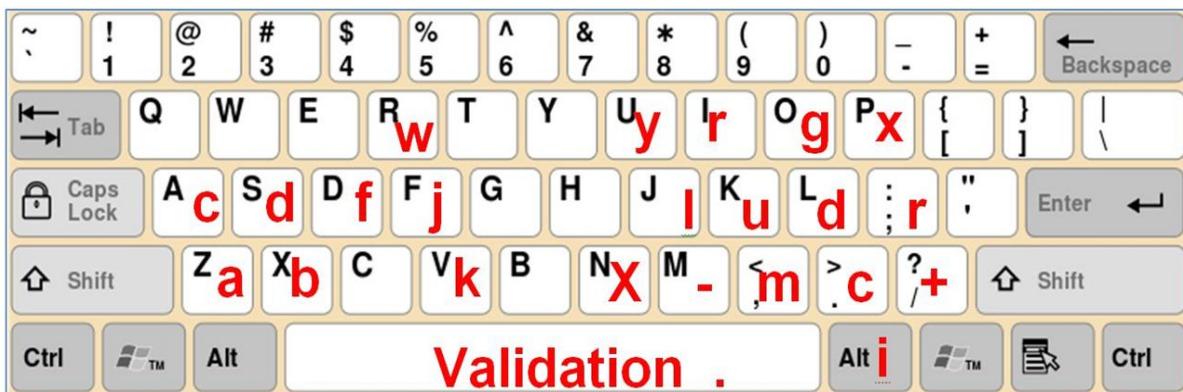
Touche « **F1** » : passage du mode hexadécimal (**Vert Foncé**) et décimal (**Vert clair**).

Le mot « **edit** » ou « **e** » pour **éditer** le code.

Le mot « **save** » pour **sauver** le code.

Le mot « **bye** » pour **sortir** du programme.

Le mot « **find** » pour chercher une définition (**exemple find poll**), suivi de la touche « **d** » [S], permet de trouver la prochaine définition, très pratique pour se déplacer d'un bloc à un autre bloc.



/// Image : editeur_clavier.png ///

Fig. 1 : Correspondance des touches du clavier réel (en noir) avec le clavier virtuel (en rouge). Exemple : la touche du clavier [R] correspond à la touche « **w** » qui a pour signification « entrer le texte couleur blanc » (commentaire).

/// Fin légende ///

7. Mon premier programme :

Pour démarrer en douceur ça vous dit un simple clignotant ? Nous allons piloter la ‘pin’ 7 du chip en sortie, qui correspond au processeur 500. Découvrons ensemble la structure du code dans le bloc 858. La définition du mot en rouge **demohilo** permet d’obtenir sur la pin 7 un niveau haut puis un niveau bas. L’écriture de 30000 en hexa dans le registre b, permet d’obtenir un niveau haut en sortie, tandis que 20000 en hexa permet un niveau bas. Une pause entre ces 2 états est codée par une boucle 100 **for next**. Le mot boucle attend le paramètre n, et permet de générer n fois le nombre d’impulsions.

Pour rappel, Escape permet de sortir du mode actuel, [N] permet d’effacer le mot précédent le curseur.

taper **858** ‘barre Espace’ **edit** ‘barre Espace’

[R] **demo hilow** ‘barre Espace’ ‘touche ESC’

[U] **500** ‘barre Espace’ **node** ‘barre Espace’ **10** ‘barre Espace’ **org** ‘barre Espace’ ‘touche ESC’

[X] **cr** ‘barre Espace’ ‘touche ESC’

[I] **demohilo** ‘barre Espace’ ‘touche ESC’

[Z] **a** ‘barre Espace’ ‘touche ESC’ (va afficher un code C000, le compilateur modifiera de lui-même le code pour afficher l’adresse exacte).

[O] Appuyer la touche F1 pour passer en mode Hexadécimal **30000** ‘barre Espace’ **!b** ‘barre Espace’ Appuyer la touche F1 pour passer en mode Décimal **100** ‘barre Espace’ **for** ‘barre Espace’ **next** ‘barre Espace’ ‘touche ESC’

[X] **cr** ‘barre Espace’ ‘touche ESC’

[O] Appuyer la touche F1 pour passer en mode Hexadécimal **20000** ‘barre Espace’ **!b** ‘barre Espace’ Appuyer la touche F1 pour passer en mode Décimal **100** ‘barre Espace’ **for** ‘barre Espace’ **next** ‘barre Espace’ ; ‘barre Espace’ ‘touche ESC’

[X] **cr** ‘barre Espace’ ‘touche ESC’

[U] **0** ‘barre Espace’ **org** ‘barre Espace’ ‘touche ESC’

[I] **boucle** ‘barre Espace’ ‘touche ESC’

[Z] **a** ‘barre Espace’ ‘touche ESC’ (va afficher un code C000, le compilateur modifiera de lui-même le code pour afficher l’adresse exacte).

[R] **n** ‘barre Espace’ ‘touche ESC’

[O] **for** ‘barre Espace’ **demohilo** ‘barre Espace’ **next** ‘barre Espace’ ; ‘barre Espace’

enfin pour sortir du mode actuel : ‘touche ESC’, et taper **save** ‘barre Espace’

```

colorForth
demo hilow 500 node 10 org cr
demohilo 00a 30000 !b 100 for next cr
20000 !b 100 for next ; cr
0 org
boucle 000 n for demohilo next ; cr

```

858
w yrgx
cdfj ludr
edit ab k -mc+
x.i

/// Image : bloc_858.png ///

Fig. 1 : Exemple du code pour le processeur 500, un simple clignotant sur la sortie 500.17, pin 7 du chip, réglable par le mot boucle.

/// Fin légende ///

8. La simulation :

Ca y est nous avons écrit notre premier programme mais comment tester le code du processeur 500 ? Pas de problème pour ArrayForth la commande « so » lance le simulateur. Mais avant il suffit de lui préciser où se trouve le code à simuler.

Il faut éditer le bloc 148, ajouter la ligne : demohilo 0 500 enter
Pour rappel, il faut utiliser les touches [J] [K] [L] [:] pour naviguer dans le code du bloc 148.

taper 148 ‘barre Espace’ edit ‘barre Espace’ ‘touche ESC’

[R] demohilo ‘barre Espace’ ‘touche ESC’

[U] 0 ‘barre Espace’ 500 ‘barre Espace’ enter ‘barre Espace’

Enfin pour sortir du mode actuel : ‘touche ESC’, et taper save

```

f18 software simulator cr
compile bootstrap 1242 load empty cr
prelude 1250 load engine 1252 8 loads cr
opcodes 1268 4 loads cr
display 1276 6 loads 1248 load 1288 2 loads cr
preserve variables nmem 0 n2mem 0
nm2n nod @ !node nmem @ mem ! indent
    nod2 @ !node n2mem @ mem ! ; big 100
puka nn-a nn-n 2* 8000 + block ;
code nn nn puka push puka pop 64 move ; cr
keyboard 1292 2 loads cr
ports and pins 1296 2 loads cr
spi flash testbed 1244 2 loads spi cr
interactive 1238 2 loads cr
smtn 0 0 enter br

delay 0 100 enter cr
demohilo 0 500 enter cr
softsim power first @ 1 + if cr
drop nm2n ; then 0 first ! 100 big ! cr
0 time ! 1 gap ! -1 wind? ! default ; br 148
start /softsim ok h

```

w yrgx
cdfj ludr
edit ab k -mc+
x.i

/// Image :bloc_148.png ///

Fig. 1 : « so » permet de lancer le simulateur, mais avant il faut préciser dans le bloc 148 le code à simuler : ici le 'node' ou processeur 500 qui correspond au code du bloc 858.

/// Fin légende ///

Le lien avec le compilateur pour le code utilisateur se fait dans le bloc 200. Nous allons lui dire que notre code demohilo se trouve au bloc 858.

taper **200** ‘barre Espace’ **edit** ‘barre Espace’ ‘touche ESC’

[R] **practical demo hilo** ‘barre Espace’ ‘touche ESC’

[U] **reclaim** ‘barre Espace’ **858** ‘barre Espace’ **load** ‘barre Espace’ ‘touche ESC’

taper **compile** ‘barre Espace’, les adresses en gris dans le bloc 858 après les mots demohilo et boucle deviennent respectivement **00a** et **000**.

enfin pour sortir du mode actuel : ‘touche ESC’, et taper **save** ‘barre Espace’



/// Image : bloc_200.png ///

Fig. 1 : Il faut indiquer au compilateur le lien avec le code du bloc 858 dans le bloc 200.

/// Fin légende ///

Au lancement du simulateur, graphiquement nous avons 3 parties, en haut à droite une vue des 144 processeurs, avec une zone bleue qui permet de définir le zoom sur une partie d'entre eux (zone de droite), une croix rouge qui permet de pointer sur un processeur ici le node 500. La simulation a la possibilité de s'exécuter en mode pas à pas (touche [F]) ou en mode continu (touche [D]) et ainsi de visualiser les différents registres, mémoire, entrées sorties du processeur. La partie du bas, représente le code dans le processeur, et plus à droite les piles de retour et de données. Dans notre exemple, il est intéressant de comparer le bloc 858 avec le code dans le processeur 500 et de comprendre son exécution. Nous avons bien en sortie l'effet escompté à savoir un clignotant sur la sortie 500.17 représenté par un « 0 » puis un « 1 » cycliquement au sommet du node.

Le node 500 en vert a pour signification que le processeur est actif

ludr [U] [I] [O] [P]
Déplace la zone bleu qui représente la vue des processeurs sur la gauche (200..207,200..500)

ludr [J] [K] [L] [:]
Déplace la croix rouge qui pointe sur le processeur 500

pfgs [A] [S] [D] [F]
[A] : remise à zéro similaire à un reset
[S] : passe le nombre de pas de 1 à 100
[D] : lance la simulation en mode automatique
[F] : lance la simulation pas à pas

Description du processeur 500

0 : état de la sortie 500.17
500 : numéro du node 00b : adresse du bus
0 : numéro du slot [0..3], @p : opcode
05b12 : instruction code machine
1 : compteur , 00b : valeur de P , le compteur programme
15555 : registre A
io : registre B
15555 : registre IO
15402 : sommet pile de Retour
15555 : sommet (Top) pile Data
15555 : Second pile Data
@ : port avec node adjacent

Le code original du bloc 858

```
858 list
demo hilow 500 node 10 org cr
demohilo 00A 30000 lb 100 for next cr
20000 !b 100 for next ; cr
0 org
boucle 000 n for demohilo next ; cr
```

Le code du node 500
Exemple: la définition **demohilo**

0b 30000 2/ and @b +
0a 05b12 @p !b @p .

Colonne 1 : 0a est l'adresse de la mémoire du mot **demohilo**
Colonne2 : 05b12 est le code machine des instructions
@p : est l'instruction en cours, 30000 va être déposé sur la pile

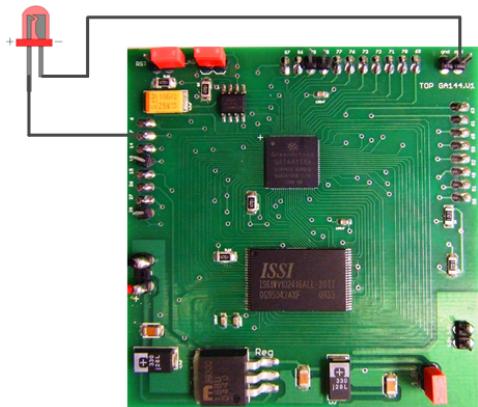
/// Image :simulation.png ///

Fig. 1 : La simulation représente les différents processeurs, le code, les registres, la mémoire.

/// Fin légende ///

9. L'exécution :

Et dans la réalité ? Rien ne vaut un effet visuel qui permet de contrôler simplement ce que l'on fait. Pour ceux qui n'ont pas d'oscilloscope sous la main, je vous propose d'intercaler une LED entre la broche 7 du chip (qui correspond à notre sortie 500.17) et le GND. Étant donné la tension de sortie de 1.8v, nous n'avons pas besoin de mettre une résistance en série avec la LED. Pour pouvoir allonger la durée entre 2 clignotements, nous devons légèrement modifier le code du bloc 858 en insérant le mot pause qui contient 2 boucles imbriquées.



/// Image :kit_led.png ///

Fig. 1 : Une LED à la sortie de la pin 7 du chip permet de visualiser le clignotement.

/// Fin légende ///

Le bloc 858 modifié:

```
858 list
demo hilow 500 node 10 org cr
pause 4000 for 10000 for next next ;
demohilo 011 30000 !b pause 100 for next cr
20000 !b pause 100 for next ; cr
0 org
boucle 000 n for demohilo next ; cr
```

Maintenant nous devons programmer le node 500, et lancer le code. Pour cela il faut éditer le bloc 860 et ajouter le code correspondant. La première partie gère la liaison série pour communiquer avec le GA144, le mot « run » spécifie communiquer avec le « node 500 » (`talk 0 500 hook`), programmer le code dans la RAM (`0 64 500 boot`) et lance le code à l'adresse 0 qui est le mot « boucle » (`0 call`), avec le paramètre 10 sur la pile (`10 lit`). Les instructions suivantes `?ram` et `upd` permettent d'afficher le contenu de la ram du node, et de la pile de données.

Le bloc 860:

```
860 list
demo hilow boot empty compile serial load br

customize -canon 0 fh orgn ! cr
a-com sport ! a-bps bps ! !nam br

run cr
talk 0 500 hook 0 64 500 boot 10 lit 0 call cr
panel ?ram upd ;
```

taper **compile** ‘barre Espace’, les adresses en gris dans le bloc 858 après les mots `demohilo` et `boucle` deviennent respectivement `011` et `000`.

enfin pour sortir du mode actuel : ‘touche ESC’, et taper **save** ‘barre Espace’

taper **860** ‘barre Espace’ **load** ‘barre Espace’ **run** ‘barre Espace’

Ouah ! Miracle la Led clignote ! De plus j'ai une nouvelle interface avec le contenu de la RAM, le numéro du node (**500**) et en haut la pile de données.

colorForth

indicator panel 135 load node stack / upd .s

2ba60 2a2b0 2aaaa 2aaea
2ab2a 22aab 0a2a2 2faba 2aaea 2aaaa br

path, via, hops, tgt - green selected .pth

2 709 1 709
1 608 1 608
0 707 10 500 br

mem dump / ?ram or ?rom>.ram

2e9b2 13411 1f401 15555 134a9 134a9 134a9 134a9
134a9 134a9 048b2 01388 048b2 02710 1f40e 1f40c
15555 05bb2 30000 1340a 05bb2 20000 1140a 134a9
134a9 134a9 134a9 134a9 134a9 134a9 134a9 134a9

qwer

run

/// Image :panel_node_500_led.png ///

Fig. 1 : Le panel permet de visualiser et d'interagir avec le node 500.

/// Fin légende ///

Nous pouvons interagir avec le processeur 500, on dépose sur la pile 5(**5 lit**), qui correspond au nombre de clignotement et on fait un appel au mot boucle (**0 call**).

taper **5** 'barre Espace' **lit** 'barre Espace' **0** 'barre Espace' **call** 'barre Espace'

10.La communication entre processeurs

Mais comment passer les données entre les différents processeurs ? Il faut se reporter au poster « GA144 vu de l'intérieur », entre chacun des nodes, nous avons des liaisons du type Up, Down, Left ou Right. Nous devons indiquer au port de communication « @ » ou doivent transiter les données à envoyer et recevoir. Je crois qu'un exemple permettra de nous éclairer efficacement sur ce mode de fonctionnement.

Supposons que nous devons passer 5 valeurs du node 517 vers le node 417. Entre ces 2 nodes la liaison est du type Down (en bleu). Dans le code « down b ! » va configurer le port, « !b » va écrire les données sur le port. « @p » récupère les données stockées à partir de l'adresse 5.

The screenshot shows a terminal window titled "colorForth". The code in the terminal is:

```
517 node 0 org
send 00 down b! emission de 5 donnees cr
4 for lecture donnees @p écriture !b unext ; c
r
05 donnees 1 , 2 , 3 , 4 , 5 , cr
```

Below the code, there is a memory dump with the number 866 at the top, followed by a grid of characters.

w	y	r	g	x					
c	d	f	j	l	u	d	r		
e	d	i	t	a	b	k	-m-	c	+
x	.	i							

/// Image : 866.png ///

Fig. 1 : Le code dans le node 517 transmet les données au node 417.

/// Fin légende ///

Le mot ini par le code « 10 a ! » signifie que le registre 'a' prend comme valeur 10. Cette valeur sera l'adresse de départ pour le stockage des données reçues. Le code « @b » va lire la donnée présente sur le port, « !+ » stocke la donnée à l'adresse pointé par le registre 'a' et l'adresse sera incrémentée.

The screenshot shows a colorForth window with the title 'colorForth'. The code in the window is:

```
417 node 0 org
ini 00 adresse ecriture 10 a!>
get 02 -n down b! reception donnees cr
4 for recuperation @b stockage !+ unext cr
reinitialisation du port io b! ; cr
```

Below the code, the status bar displays the number '864' in red, followed by memory dump information: 'w yrgx', 'cdfj ludr', 'edit ab k -mc+', and 'x.i'.

/// Image : 864.png ///

Fig. 1 : Le code dans le node 417 va recevoir les données du node 517.

/// Fin légende ///

Nous mettons à jour les liens pour le simulateur dans le bloc 148 et dans le bloc 200 pour le compilateur.

The screenshot shows a colorForth window with the title 'colorForth'. The code in the window is:

```
f18 software simulator empty compile,
demo bootstream 1242 loadempty,
prelude 1250 load boot descriptors 1236 load,
engine 1252 8 loads opcodes 1268 4 loads,
boot loader 1238 load,
display 1276 6 loads 1248 load 1288 2 loads,
preserve variables nmem 0 n2mem 0
nn2m nod2 @ !node n2mem @ mem !,
...nod @ !node nmem @ mem !; big 100
puka nn-a nn-n 2x 8000 + block ;
code nn nn puka push puka pop 64 move ;
keyboard 1292 2 loads,
ports and pins 1296 2 loads,
interactive 1240 load,
communication inter node 0 517 enter,
0 417 enter,
/softsim 0 time : power,
first @ 1 + if drop nm2m ; then 0 first !,
100 big ! 1 gap ! -1 wind? !,
100 !node 0 mem ! 100 other,
0 !node 0 mem ! 0 node 0 xo ! 0 yo ! tvis 148
,
init and testbeds 216 load,
start /softsim ok h
```

Below the code, the status bar displays the number '148' in red, followed by memory dump information: 'w yrgx', 'cdfj ludr', 'edit ab k -mc+', and 'x.i'.

/// Image : 148.png ///

Fig. 1 : Le bloc 148 est mis à jour pour le simulateur, ajout du code communication inter node.

/// Fin légende ///

```
colorForth
user f18 code reclaim br

softsim example reclaim 0 node 1342 load cr
practical example pwm code reclaim 842 load cr
communication inter node reclaim 864 2 loads cr
r
adc demo reclaim 868 3 loads cr
```

/// Image : 200.png ///

Fig. 1 : Pour le compilateur dans le bloc 200, nous spécifions dans quel bloc se trouve le nouveau code.

/// Fin légende ///

Le simulateur est lancé par la commande « so », nous avons bien le résultat attendu, visible sur le node 417 les données sont stockées à partir de l'adresse 10 (0a).

```
517 ?  
0:  
95555  
0 ?  
15555  
d  
15555  
15555  
15555  
15555  
  
417 ?  
0:  
95555  
1 ?  
0000f  
io  
15555  
15555  
15555  
15555 - 417 0000f  
io  
e 0 155 95555 ;  
317rdw 0f 134a9 call 0a9  
4fetch 0e 00005 bb and 0b ;  
15555 0e 00004 bb and 0b unext  
- rdu 0c 00003 bb and 0b dup  
815555 0b 00002 bb and 0b .  
15555 0a 00001 bb and 0b *x  
15555 09 134a9 call 0a9  
15555 08 295555 bt ;  
15555 07 00165d bb + ex  
15555 06 016774 bb + unext ep  
0 05 268b2 push .  
217rdw 04 00004 bb and 0b unext  
4fetch 03 00115 bb + ep ;  
15555 02 04b12 bb ! ep .  
- rdu 01 0000a bb and 0b .  
e 15555 00 04ab2 bb a' .  
  
io  
15555  
15555  
15555  
15555  
e  
77  
1  
so -+ ludr  
pfgs ludr  
-+ ohlw  
i.
```

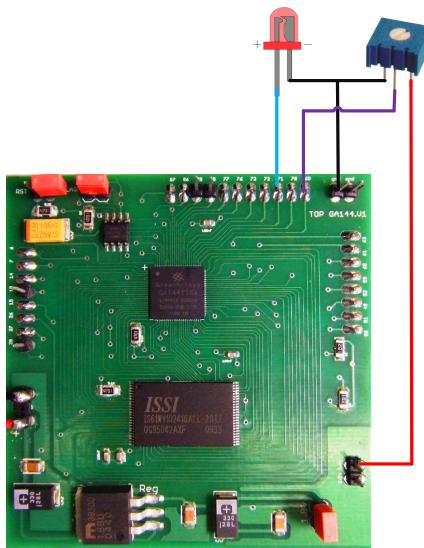
/// Image : communication_interNode.png ///

Fig. 1 : Le node 517 a passé 5 valeurs au node 417, et celle-ci sont stockées dans le node 417 à partir de l'adresse 10 (0a).

/// Fin légende ///

11. La conversion A/D:

Jouons avec le convertisseur analogique digital du GA144. Pour cela il faut intercaler un potentiomètre sur la pin 69 du chip (qui correspond à l'entrée 717.ai) et le Vcc et GND. De même nous ajoutons une LED entre la broche 71 du chip (qui correspond à notre sortie 715.17) et le GND.



/// Image : Led_potar.png ///

Fig. 1 : Une Led sur la sortie 715.17 (pin 71) et une résistance ajustable sur l'entrée 717.ai (pin 69)

/// Fin légende ///

Puisque je suis un peu fainéant, je vais copier le code du bloc 630 vers le bloc 872 et le modifier.

taper **630 'barre Espace' blk 'barre Espace'** **872 'barre Espace' copy 'barre Espace'**

Les modifications sont sommaires il faut mettre en commentaire la virgule à la fin de la définition du mot « mv » et ajouter le mot dup (duplique le sommet de la pile) et insérer une nouvelle définition le mot send (permet d'envoyer la valeur vers le node 716).

```
1900 afs analog cr
reclaim 717 node 0 org
sam n-k 00 io b! data a! @b - 1ff and or cr
!b dup ! @ 1000 for unext dup ! @ - . + ;
vdd Oc 2000 sam ; -cr vss Oe 6000 sam ;
vpin 10 0 sam ;
ux nn-hl 12 dup a! dup or 17 for +* unext cr
push drop pop a ;
m/ dn-q --u/mod push drop pop ;
mv -n 19 vss vdd over - . + push cr
vpin - . + -if dup or then 1800 ux pop m/ ; br
dup
send 24 n right b! !b io b! ;
!mv n 28 io b! 511 ux -1800 m/ 155 or !b ; cr
30 1900 bin
```

/// Image : 872.png ///

Fig. 1 : Le code du bloc 872, inspiré du bloc 630, légèrement modifié.

/// Fin légende ///

Le code du bloc 874 va programmer le processeur 717 et lancer son programme.

A screenshot of a colorForth terminal window titled "colorForth". The code in the window is:

```
demo adc boot empty compile serial load br
customise -canon 0 fh orgn ! cr
a-com sport ! a-bps bps ! !nam br

run cr
talk 0 717 hook 0 64 717 boot 19 call br
panel ?ram upd ;
```

The screen shows the number 874 in red at the top center, followed by a series of lowercase letters: w yrgx cdfj ludr edit ab k -mc+ x.i.

/// Image : 874.png ///

Fig. 1 : Le code du bloc 874 permet de configurer le processeur 717 et de lancer une mesure du convertisseur A/N. La valeur récupérée est empilée au sommet de la pile de données.

/// Fin légende ///

Bien sûr ne pas oublier de spécifier dans le bloc 200 le nouveau code.

A screenshot of a colorForth terminal window titled "colorForth". The code in the window is:

```
user f18 code reclaim br

softsim example reclaim 0 node 1342 load cr
practical example pwm code reclaim 842 load cr
communication inter node reclaim 864 2 loads c
r
adc demo reclaim 868 3 loads cr
```

The screen shows the number 200 in red at the top center, followed by a series of lowercase letters: w yrgx cdfj ludr edit ab k -mc+ x.i.

/// Image : 200.png ///

Fig. 1 : Ajout de la ligne adc demo qui signifie de charger les blocs 868, 870,872.

/// Fin légende ///

taper **compile** 'barre Espace' **save** 'barre Espace'

Comme je suis impatient de jouer avec notre nouveau code :

taper **874** 'barre Espace' **load** 'barre Espace' **run** 'barre Espace'

La commande « run » permet de récupérer la donnée mesurée par le convertisseur. Le fait de tourner le potentiomètre dans un sens ou dans l'autre aura l'effet d'augmenter ou diminuer la valeur. Pour la suite de l'exemple, il faut régler le potentiomètre pour avoir une mesure proche de 10. Et nous pouvons continuer à coder

Ajouter le code suivant dans le bloc 870, celui-ci permet de transmettre les données d'un bloc à l'autre.

The screenshot shows two nodes in a colorForth environment. Node 716 contains the following code:

```
716 node 0 org cr
get 00 -n right b! >eb io b!
send 04 n left b! !b io b! ;
```

Node 870 contains the following code:

```
870
w yrgx
cdfj ludr
edit ab k -mc+
x.i
```

/// Image : 870png ///

Fig. 1 : Le code dans le node 716 récupère la donnée envoyée par le node 717, pour la transmettre à son tour vers le node 715.

/// Fin légende ///

Et le code de « dimohilow » est copié et subit une légère retouche.

The screenshot shows two nodes in a colorForth environment. Node 715 contains the following code:

```
demo hilow 715 node 10 org cr
pause 0a 5000 for 10000 for next next ;
demohilo 11 30000 !b pause 20000 !b pause ; cr
0 org
get 00 -n left b! >eb io b! recuperation valeur
du node 600 cr
boucle 04 n for demohilo next ;
```

Node 868 contains the following code:

```
868
w yrgx
cdfj ludr
edit ab k -mc+
x.i
```

/// Image : 868.png ///

Fig. 1 : Le code dans le node 868 récupère la donnée envoyé par le node 716 pour générer « n » clignotements sur la sortie du node 715.

/// Fin légende ///

Une nouveauté, pour pouvoir programmer et exécuter les programmes des différents processeurs, il faut utiliser le code du bloc **846** suivant. La configuration concernant le **processeur 717** permet de pointer sur le code à l'adresse « **19** » (définition du mot **mv**). Les **2 autres nodes** vont pointer à l'adresse 0.

The screenshot shows a terminal window titled "colorForth" with the following code:

```
colorForth
Loader template host load loader load,
using default ide paths,
kill boots 0 708 hook 0 -hook,
,
setup application,
...717 +node 717 /ram 0 1 /stack 19 /p,
...716 +node 716 /ram 0 1 /stack 0 /p ,
...715 +node 715 /ram 0 1 /stack 0 /p,
visit whole chip 2 0 ship panel upd ?ram
```

Below the terminal window is a graphic of the number **846** in red, with smaller text "w yrgx", "cdfj ludr", "edit ab k -mc+", and "x. i" arranged around it.

/// Image : 846.png ///

Fig. 1 : Le code dans le node 846 va configurer les nodes du GA144, et exécuter les programmes des processeurs.

/// Fin légende ///

taper **compile** 'barre Espace' **save** 'barre Espace'

Comme je suis encore plus impatient de jouer avec notre nouveau code :

taper **846** 'barre Espace' **load** 'barre Espace'

Quoi de nouveau ? Cool la Led clignote le nombre de fois réglé par le potentiomètre.

Maintenant nous savons jouer avec les processeurs et même interagir entre eux !

12. Le GA144 en mode sommeil :

Il est possible de mettre en sommeil et de réveiller un node du GA144 par exemple lorsque sur une entrée le signal change d'état (0 ou 1 logique) . Dans la réalité cela viendrait à intercaler un bouton poussoir sur la broche 6 du chip (qui correspond à l'entrée 600,17) et le GND.

Mais dans notre exemple , nous allons dans la simulation, simuler le changement d'état de la broche 6.

The screenshot shows a terminal window with a black background and white text. At the top, there is some configuration code for a softbed wave. Below it, a digital clock simulation is running, displaying the time as 1232. The digital clock has red digits and a green background. To the right of the digits, there is a legend with various symbols: w, y, r, g, x, c, d, f, j, l, u, d, r, e, a, b, k, -m, c, +, x, ., i.

```
colorFont
softbed wave
/wave softbed assign time,
@ 100 / 1 and ?v p17v t ; cr
600 !node /wave
```

/// Image : 1232testbed.png ///

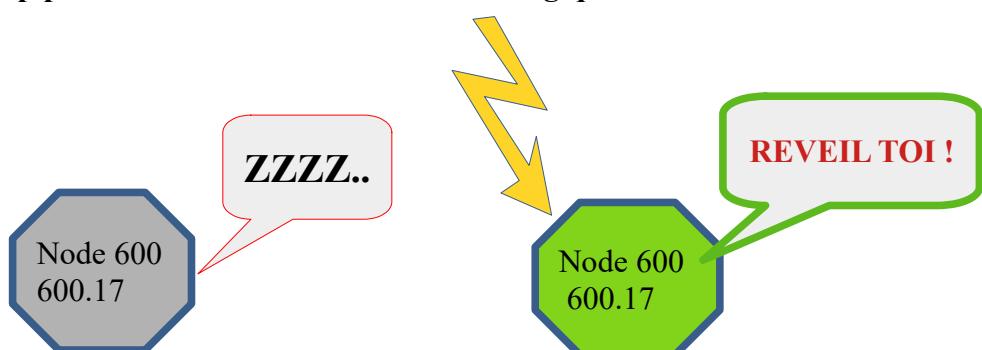
Fig. 1 : Le code dans le node 1232 va simuler un changement d'état 0 puis 1 , tout les 100 pas dans le simulateur

/// Fin légende ///

Pour pouvoir signaler au node, et en particulier **au registre i/o** du node (ici node 600 pin17), il faut programmer le registre io , avec WD = **0** si nous voulons une détection d'un niveau logique à 1 , et inversement WD = **1** si nous voulons un détection d'un niveau logique à 0.

WD : Wakeup pins

niveau logique



La programmation du registre IO , permet de changer la détection du niveau logique

WD = 1 soit le IO Register = \$800 → détection d'un niveau logique à 0

WD = 0 soit le IO Register = \$0 → détection d'un niveau logique à 1

BIT	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reset	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
WRITE	pin 17	ctl					WD						pin 5	ctl	pin 3	ctl	pin 1	ctl
Alt write	SR			vco	ctl	DB				9	bit	D/A	val	(155 xor)				
READ	pin 17	Rr-	Rw	Dr-	Dw	Lr-	Lw	Ur-	Uw				pin 5		pin 3		pin 1	

/// Image : io_register.png ///

Fig. 1 : le registre des entrées/sorties (IO register)

/// Fin légende ///

Et voilà le code du node 600 , dans le bloc 860 :

init permet d'accéder au registre io par le biais de b **io b!** , d'écrire dans **le registre io** à **\$800** ou 0 (**dup or**), pour pouvoir effectuer une détection lorsque le niveau logique est à 0 ou 1 respectivement , et dormir...

left b! Indique sur quelle broche (ici 600.17 à gauche) nous mettons la mise en oeuvre du **wake-up**

?pin @b permet de lire l' état de la broche 17 et ne pas oublier de faire ensuite un **drop**

demo effectue une boucle sans fin.

```
colorForth
pin demo 600 node 10 org br
?pin @b drop wait pin 17 1 ;,
o org
init io b! wd 800 dup or !b pin17 left b!
demo 05 ?pin demo ;>
860
```

/// Image : 860_wakeup.png ///

Fig. 1 : le code du bloc 860 permet de mettre en oeuvre cette gestion du wakeup

/// Fin légende ///

Ne pas oublier de rajouter dans le bloc 216 , le chargement du bloc 1232 pour la simulation du signal /wave



```
colorForth
softsim configuration,
.
spi boot testbed 1244 2 loads,
sync boot testbed 'addr, len' 1230 load,
wave boot testbed 1232 load,
smtm 0 +node 0 /ram 0 /p,
/commands test 400 +node 0 /ram 25 /a 12 /b,
9 8 7 6 5 4 3 2 1 12345 10 /stack a9 /p,
.
rom write test 200 +node 13 /p,
.
0 32 103 break,
0 be 300 break
```

216

/// Image : 216_wave_demo.png///

Fig. 1 : le chargement du bloc 1232 pour la simulation testbed "1232 load"

/// Fin légende ///



```
colorForth
user f18 code reclaim br
softsim example reclaim 0 node 1342 load cr
practical example pwm code reclaim 842 load br

demo 860 load,
sha256 reclaim 900 5 loads,
greg n1 object 950 load,
mango test 894 load,
an012 sensor tag 960 load,
.
hardsim pwr verify 882 load,
random 884 load
```

200

/// Image : 200_wave.png ///

Fig. 1 : Le code en bloc 200 charge notre code du bloc 860 "demo 860 load"

/// Fin légende ///



```
colorForth
f18 software simulator empty compile,
demo bootstream 1242 load empty,
prelude 1250 load boot descriptors 1236 load,
engine 1252 1266 thru opcodes 1268 1274 thru,
boot loader 1238 load,
view 1276 1286 thru 1248 load 1288 1290 thru,
preserve variables nmem 0 n2mem 0
n2m nod2 @ !node n2mem @ mem !,
...nod @ !node nmem @ mem ! ; big 100
puka nn-a nn-n 2x 8000 + block ;
code nn nn puka push puka pop 64 move ;
kbd 1292 1294 thru,
ports/pins 1296 1298 thru,
interactive 1240 load,
demo 0 600 enter,
/softsim 0 time ! power,
first @ 1 + if drop n2m ; then 0 first !,
100 big ! 1 gap ! -1 wind? !,
100 !node 0 mem ! 100 other,
0 !node 0 mem ! 0 node 0 xo ! 0 yo ! !vis ;
init and testbeds 216 load.
```

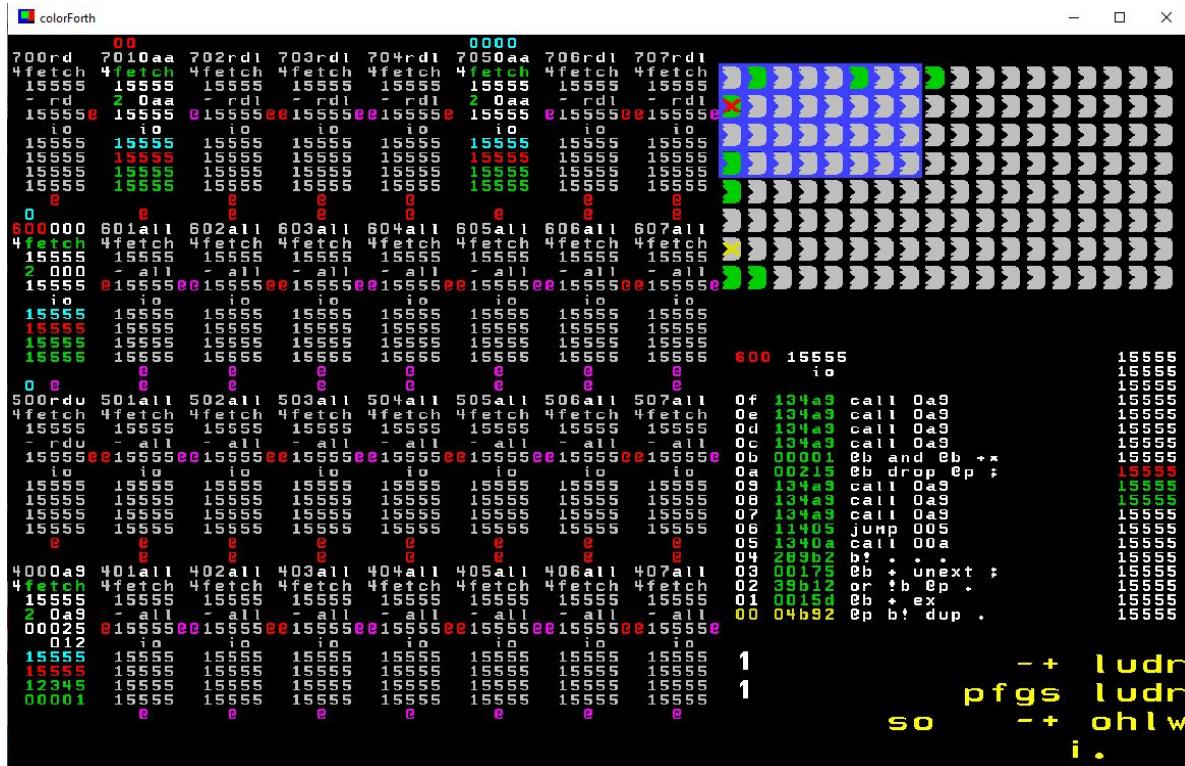
148

/// Image : 148_demo_wave.png ///

Fig. 1 : Le code en bloc 148 va specifier au simulateur le node 600 "demo 0 600 enter"

/// Fin légende ///

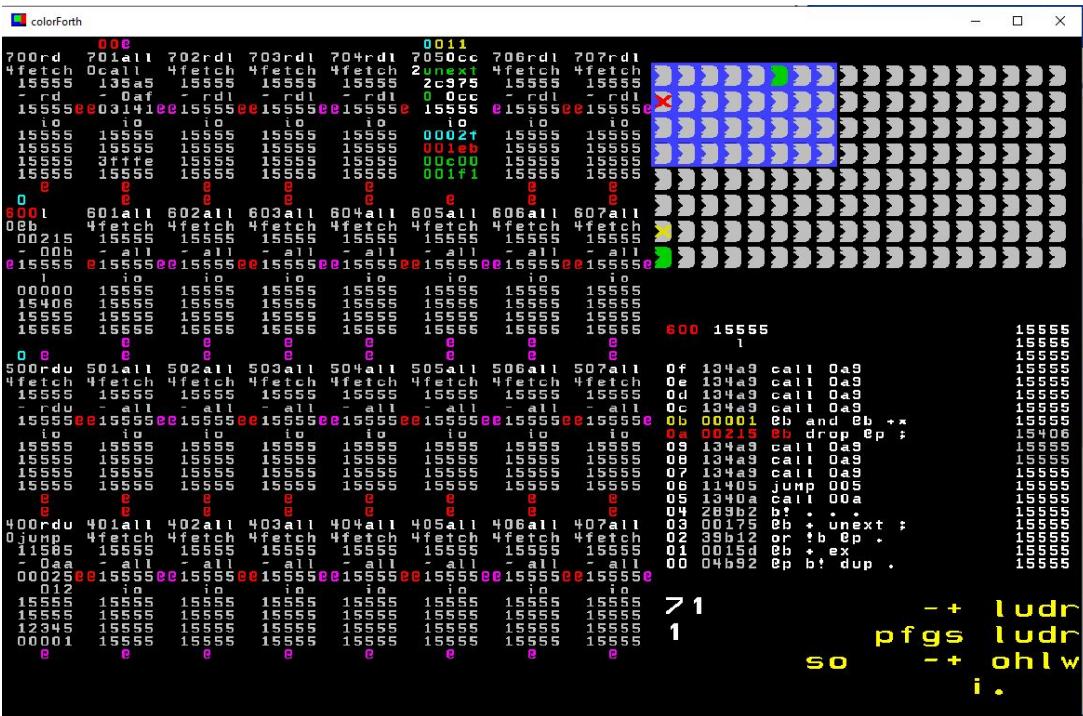
Et en mode simulation , finalement cela donne bien l'effet escompte , a savoir que le node va se mettre en veille, tant que le signal sur la broche 6 (600.17) est a un niveau logique de '0' .



/// Image : simu_600_0.png ///

Fig. 1 : La simulation du node 600 (croix rouge sur fond vert) , avec en bleu clair 0 sur la pin17

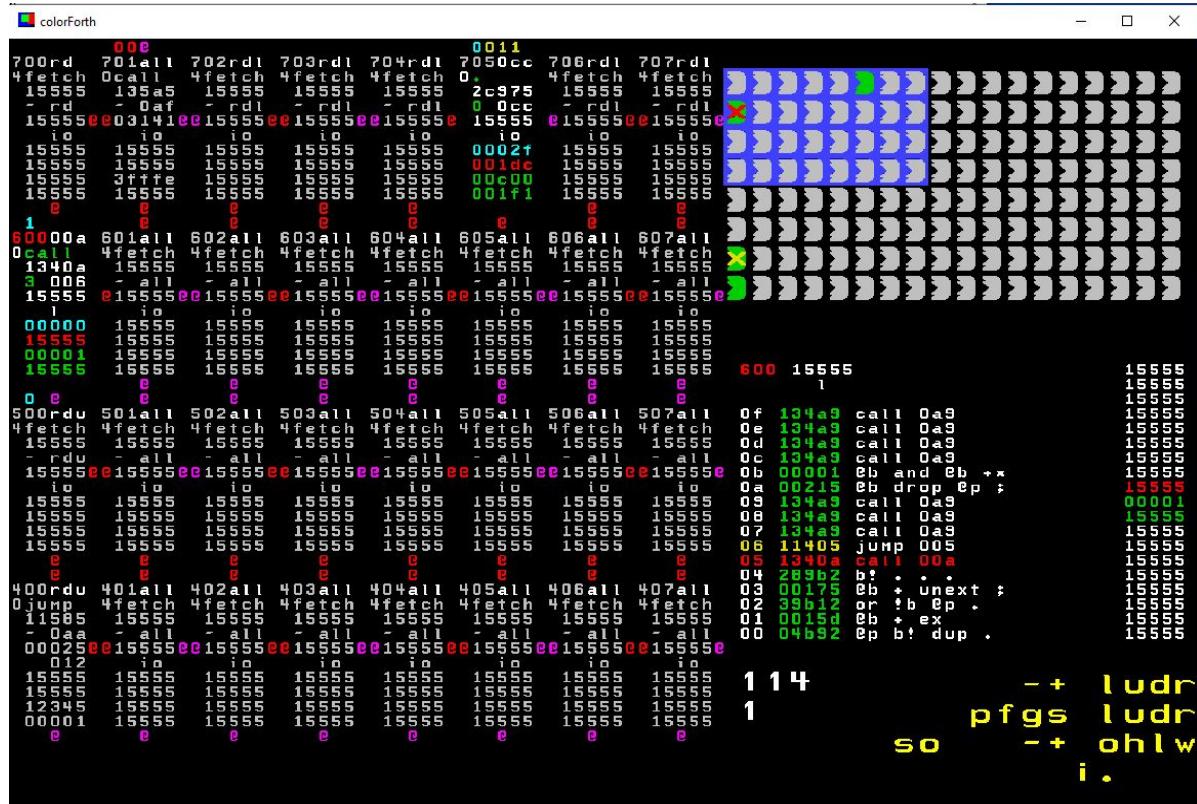
/// Fin légende ///



/// Image : simu_600_sleep.png ///

Fig. 1 : Le node 600 (croix rouge sur fond gris) es en mode sommeil profond.... Et attend que le niveau logique (en bleu) passe a 1
/// Fin légende ///

ET hop , le node 600 vient de se reveiller , et a depose (et deposera) sur la pile la valeur 1 jusqu a ce que la valeur logique sur la pin passe a '0'



/// Image : simu_600_1.png ///

Fig. 1 : La simulation du node 600 (croix rouge sur fond vert avec en bleu clair 1 sur la pin17, le node 600 est bien reveille !

/// Fin legende ///

13. Quelques liens:

<http://esaid.free.fr/>

<http://www.greenarraychips.com/index.html>

<http://www.greenarraychips.com/home/support/code/e422a-readme.start.txt>

<http://www.greenarraychips.com/home/documents/downindex.html>

<http://www.colorforth.com/>

<http://www.forth.com/swiftforth/>

<http://fr.hobbytronics.co.uk/ft232rl-breakout>

http://www.ftdichip.com/Support/Utilities.htm#FT_Prog

http://www.ftdichip.com/Support/Documents/AppNotes/AN_124_User_Guide_For_FT_PROG.pdf

Denis Bodor db@ed-diamond.com lefinnois@lefinnois.net
Rédacteur en chef de GLMF. Utilisateur GNU/Linux depuis 1994. Randonneur du jardin magique.