Frédéric Sagez
Codeur du groupe NoExtra-Team
fsagez@gmail.com

Le graphisme sur ATARI ST

niveau 200

Nous allons nous intéresser au fonctionnement de l'affichage d'un Atari ST et comprendre comment sont gérées les parties graphiques entre la mémoire et la partie vidéo du ST.

Comment afficher des pixels sur un écran? Pour rappel 1 pixel est égal à 1 point et ce, en utilisant une couleur disponible sur la palette et suivant la résolution désirée. Si on veut utiliser le nombre maximum de couleurs sur notre écran, cela est possible mais uniquement en basse résolution pour le ST avec 16 couleurs au total que l'on pourra choisir sur un échantillon de palette de 512 couleurs pour un Atari STF ou 4096 couleurs pour un Atari STE. Petit rappel : l'Atari ST ne comporte que 3 résolutions d'écran possibles : Basse, Moyenne et Haute résolution spécifique au moniteur monochrome de type SM124.

La théorie selon Atari

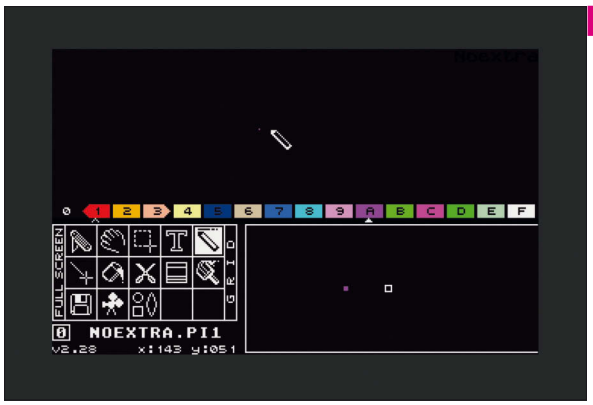
Ce qu'il faut retenir pour un écran en résolution ST-LOW (basse résolution), soit un affichage en 320 par 200 pixels de 4 bits composé de 4 plans (soit 16 couleurs utilisables) que cela représente un total de 320 x 200 x 4 bits = 256000 bits, soit 32000 octets au total de mémoire vidéo utilisée. Et cette zone de mémoire vidéo de 32 ko se trouve exactement à partir de la limite supérieure de la mémoire. Elle est initialisée en tant que telle dès la mise sous tension du ST. Donc si je calcule bien pour un Atari STF faisant 512 ko de mémoire d'origine, notre adresse mémoire au format hexadécimal devrait être de $80000 - $0FD00, soit l'adresse $78300. Eh bien non! Le calcul est faux !!!??? La bonne réponse est l'adresse $78000 car la taille finale de notre écran physique fait au total 32768 octets soit $08000. Alors comment faire pour ne pas se tromper, surtout si l'on veut utiliser aussi un autre écran de 32 ko comme écran physique ? On utilisera généralement les fonctions étendues appelées XBIOS du ST en passant par un appel du Trap #14 et en passant le numéro de la fonction à appeler en paramètre, XBIOS(2) pour obtenir l'adresse physique et XBIOS(3) pour obtenir l'adresse logique (un deuxième écran) quel que soit le nombre de RAM d'un ST.

```
move.w #0,-(a7)          ; set ST Low Resolution
move.l #$78000,-(a7)     ; set physical screen
move.l #$78000,-(a7)     ; set logical screen
move.w #5,-(a7)          ; function Setscreen()
trap   #14               ; XBIOS function called
lea.l  12(a7),a7         ; for Atari 520 STF
```

Pour rappel l'adresse physique de l'écran est la zone de mémoire qui est affichée sur le moniteur par le composant vidéo appelé SHIFTER. L'adresse logique est utilisée comme base de l'écran par toutes les fonctions de sorties du ST. Pour la suite de cet article, nous utiliserons la même adresse vidéo car pour notre démonstration, seul un écran est nécessaire à l'affichage vidéo.

Définition d'un écran graphique

Notre écran est composé d'une ligne de 320 pixels en horizontal et de 200 lignes en vertical : une ligne de 320 pixels x 4 bits est égal



1 *Tracé du point sur l'outil Néochrome Master*

à 1280 bits et si on divise par 1 byte cela donnera 160 octets. Une ligne fait 320 pixels en longueur et elle est découpée en 20 colonnes, donc pour afficher un pixel sur l'écran, nous disposons de 20 colonnes de 16 pixels. Un bloc de 16 pixels contient 4 mots de 16 bits pour chaque plan. Dans notre exemple suivant nous allons dessiner un point de couleur numéro 10 sur les 16 couleurs de la palette, il prendra les valeurs suivantes en mot – Word : $0000,$0100,$0000,$0100 (valeurs en hexadécimal).

Donc pour récapituler, si je veux écrire un point sur l'écran qui utilise un numéro de couleur :

- les couleurs de 0 et 1 utilisent un plan : j'utilise un seul mot,
- les couleurs de 0 à 3 utilisent deux plans : j'utilise les deux premiers mots,
- les couleurs de 0 à 7 utilisent trois plans : j'utilise les trois premiers mots,
- les couleurs de 0 à 15 utilisent quatre plans : j'utilise les quatre mots.

Et en tout on peut mettre 16 points de couleurs de 0 à 15 sur un bloc de 16 pixels contenant 4 plans. Bon vous n'avez pas tout compris, mais pour afficher un point sur l'écran il faut le positionner dans un bloc de 16 pixels, hum… pas de panique ! Voici un exemple pour vous rendre compte du premier niveau de complexité.

Un peu de pratique

Pour la démonstration j'utilise l'outil de dessin **Neochrome Master** pour dessiner un pixel, j'utilise la couleur numéro 10 (valeur $A en hexadécimale) sur la palette qui correspond à la couleur violette située dans le quatrième plan comme on peut le voir sur le dessin 1 en basse résolution. Le point se trouve à la position 136 pour l'axe des X et 51 pour l'axe des Y et comme on peut le voir, cela correspond au neuvième bloc de 16 pixels situé sur l'écran. Pour avoir la valeur en mot – soit en WORD – de ma ligne, j'exporte en hexadécimal via l'outil les valeurs de la ligne où se situe mon point pour connaître les valeurs exactes des quatre mots de mon bloc de 16 pixels. 2

Pour dessiner sur notre écran le point, nous allons utiliser l'outil **DEVPAC** et écrire une petite routine en assembleur permettant de

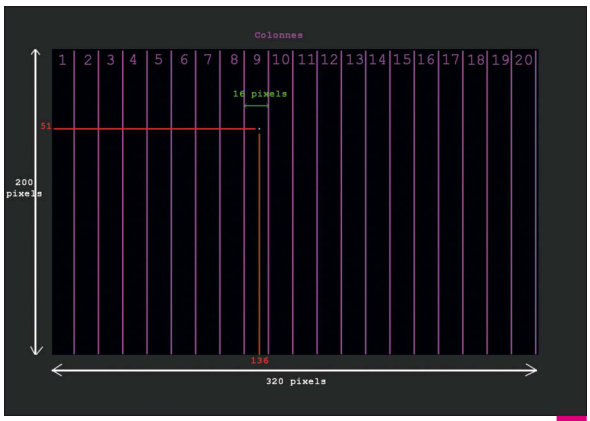
visualiser sur l'écran le résultat final. Dans un premier temps je vais déclarer un seul écran pour afficher mon résultat et je suis les préconisations du constructeur en utilisant l'adresse de la partie mémoire graphique d'un Atari 520 STF (soit 512Ko de mémoire) à l'adresse $78000 tout en utilisant une fonction étendue du BIOS. Je vais écrire une routine appelée « main » que je vais appeler en mode SUPERVISEUR pour accéder à tout l'environnement du ST. Dans la routine je vais sauvegarder la palette système de base utilisée par le Desktop, afficher un fond noir et initialiser la couleur numéro 10 de la palette de destination. Ensuite je remplis mon écran de lignes indiquant les 20 blocs de 16 pixels pour vérifier que je suis au bon endroit. 3

En passant par le registre d'adresse a0, je lui assigne l'adresse de mon écran physique et je place mes 4 mots sur les 4 différents plans. Pour connaître les coordonnées de la ligne c'est très simple : on multiplie le numéro de ligne par 160 octets et pour la colonne, c'est la 8ème colonne multipliée par 8 octets (comme on commence par zéro, on soustrait toujours le numéro de la colonne par un).

```
    SECTION TEXT

move.w #0,-(a7)             ; set low resolution
move.l #$78000,-(a7)        ; set physical screen
move.l #$78000,-(a7)        ; set logical screen
move.w #5,-(a7)             ; setscreen
trap   #14                  ; XBIOS function called
lea.l  12(a7),a7            ; for Atari 520 STF

pea main(pc)                ; exec routs main in super mode
move.w #$26,-(sp)
trap #14
addq.l #6,sp

clr.l -(sp)                 ; exit !
trap #1

main:
movem.l $ffff8240.w,d0-d7
movem.l d0-d7,sav_pal       ; save palette system

move.l #$00000777,$ffff8240.w ; put palette for color 0 and 1
move.l #$00E70FC7,$ffff8250.w ; put palette for color 10 and 11

lea.l  $78000,a0            ; Fill screen by chunck of 16 pixels
move.w  #(160*200)/4-1,d7   ; in 20 parts
move.l  #$00020000,(a0)+    ; of 2 first words
dbf    d7,*-6               ; loop above line instruction

*> calculate position on the screen
lea.l  $78000,a0            ; manage our screen
lea    160*51(a0),a0        ; position Y = 51
add.w#(9-1)*8,a0            ; position X = 136
                            ; 9 chuncks - 1 x 2 x 4 words
*> put 16 pixels with a pixel color number 10
* Rappel: dc.w  $0000,$0100,$0000,$0100
move.w #$0000,0(a0)         ; first bitplane
move.w #$0100,2(a0)         ; second bitplane
```

```
 1 *
 2 *  NEOchrome V2.28 cut buffer contents (left justified):
 3 *  by Chaos, Inc. of the Delta Force (member of The Union)
 4 *
 5 *    pixels/scanline     = $013F (bytes/scanline: $00A0)
 6 *  # scanlines (height) = $0002
 7 *
 8 *  Hardware color pallet  (color 0 to 15):
 9 *
10 *       $0800,$0F00,$0FD0,$0FDC,$0FFC,$0005,$0E65,$083F
11 *       $00E7,$0FC7,$0E07,$01F2,$0E25,$0AD1,$0DED,$0FFF
12 *
13      dc.w   $0000,$0000,$0000,$0000,$0000,$0000,$0000,$0000
14      dc.w   $0000,$0000,$0000,$0000,$0000,$0000,$0000,$0000
15      dc.w   $0000,$0000,$0000,$0000,$0000,$0000,$0000,$0000
16      dc.w   $0000,$0000,$0000,$0000,$0000,$0000,$0000,$0000
17      dc.w   $0000,$0100,$0000,$0100,$0000,$0000,$0000,$0000
18      dc.w   $0000,$0000,$0000,$0000,$0000,$0000,$0000,$0000
19      dc.w   $0000,$0000,$0000,$0000,$0000,$0000,$0000,$0000
20      dc.w   $0000,$0000,$0000,$0000,$0000,$0000,$0000,$0000
21      dc.w   $0000,$0000,$0000,$0000,$0000,$0000,$0000,$0000
22      dc.w   $0000,$0000,$0000,$0000,$0000,$0000,$0000,$0000
23
                                          9ème colonne
```

Fichier de données hexadécimales de la ligne 2



Résultat de l'affichage de notre point en assembleur 3

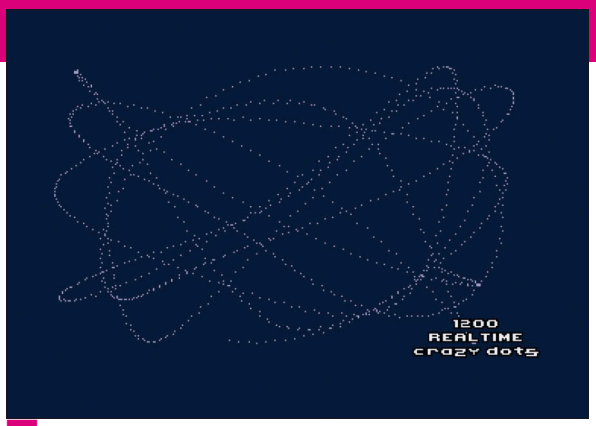
```
move.w #$0000,4(a0)         ; third bitplane
move.w #$0100,6(a0)         ; fourth bitplane

move.w  #7,-(sp)            ; wait for a key press
trap  #1
addq.l  #2,sp

movem.l sav_pal,d0-d7       ; restore palette system
movem.l d0-d7,$ffff8240.w
rts

sav_pal ds.l 8
```

Enfin j'attends qu'une touche du clavier soit appuyée pour rendre la main à l'utilisateur tout en m'assurant de remettre la palette d'origine. Au final j'écris 4 mots aux adresses suivantes : $7A060, $7A062, $7A064 et $7A066 de mon écran physique.

Maintenant que nous avons vu comment dans cet exemple on affiche un point sur l'écran, il serait intéressant de pouvoir placer un point où l'on veut sur l'écran avec simplement ses coordonnées.

Point, pixel et plan

Et pour cet exercice je vais vous montrer comment écrire un point à la position voulue. Mais dans notre cas, le point pourra être affiché uniquement sur l'un des 4 plans et ne pourra donc utiliser que 4 couleurs possibles. Pourquoi ? C'est juste pour une question d'optimisation sur l'utilisation de mes 4 Words car le premier plan utilise uniquement un Word comme on peut le voir sur les 16 pixels : $0080,$0000,$0000,$0000. Donc pour la couleur numéro 1 on pourra utiliser un seul mot à l'adresse écran $7A060 et on pourra mettre la même valeur pour les trois autres plans si on le souhaite et cela donnera ceci :



4 *Infinite live of the Blitter – NoExtra (2011)*

- Plan numéro 2 utilisera la couleur 2 à l'adresse écran $ 7A062,
- Plan numéro 3 utilisera la couleur 4 à l'adresse écran $ 7A064,
- Plan numéro 4 utilisera la couleur 8 à l'adresse écran $ 7A066.

Maintenant le plus difficile est à venir. Pour calculer mes coordonnées sur l'écran physique, on sait que pour placer notre point sur une ligne verticale il faut le multiplier par 160 octets (pour rappel : 320 pixels / 16 pixels x 4 mots x 2), donc on fera une simple multiplication par 160 avec l'instruction mulu. Pour la partie horizontale nous devons trouver où notre point doit se situer et cette fois-ci on enlève la notion de colonne, nous allons diviser la coordonnée par 16 pixels puis la multiplier par 8 octets. (Pour rappel : 4 mots x 2)
Ensuite pour afficher le point final à l'écran, nous utiliserons l'instruction bset pour activer le bit sur l'octet de poids fort du mot et pour cela nous récupérons le reste de notre division que l'on va soustraire à la valeur 15 (16 pixels – 1) tout en sachant que notre bit est compris entre 0 et 7 car l'instruction bset ne gère que des données sur 8 bits. Donc on obtiendra :

- Axe vertical : Offset Y = 51 x 160 = 8160, soit à partir de l'adresse vidéo $78000 + $1FE0 va donner l'adresse $79FE0
- Axe horizontal : Offset X = 136 / 16 x 8 = 64, on additionne $40 à $7A020 qui donnera la position finale à l'adresse $7A060 qui est l'offset de l'écran et on indiquera le bit à afficher sur cette adresse pour « allumer » le point à afficher. Voici le code source qui reprend la même structure que le premier programme, on réutilise seulement la procédure main pour implémenter le code suivant :

```
SECTION TEXT

move.w #0,-(a7)             ; set low resolution
move.l #$78000,-(a7)        ; set physical screen
move.l #$78000,-(a7)        ; set logical screen
move.w #5,-(a7)             ; setscreen
trap   #14                  ; XBIOS function called
lea.l  12(a7),a7            ; for Atari 520 STF

pea main(pc)                ; exec routs main in super mode
move.w #$26,-(sp)
trap #14
addq.l #6,sp

clr.l -(sp)
trap #1

main:
movem.l $ffff8240.w,d0-d7
movem.l d0-d7,sav_pal       ; save palette system

move.l #$00000777,$ffff8240.w; put palette for color 0 and 1

lea.l  $78000,a0            ; Fill screen
move.w #(160*200)/4-1,d7    ; in 20 parts
move.l #$00020000,(a0)+     ; of 8 words
dbf    d7,*-6

*> Display one pixel with the first bitplane
lea.l  $78000,a0            ; adress of the screen
move.w  #136,d0             ; position X is 136
move.w  #51,d1              ; position Y is 51
move.w  d0,d2               ; backup X in the register d2
andi.w  #$fff0,d0           ; which cluster of 16 pixels to display ?
sub.w   d0,d2               ; sub multiple of 16 with the X position
subi.w  #15,d2              ; seek bitnumber between 16 pixels
neg.w   d2                  ; the bitplane for the bset instruction
lsr.w   #1,d0               ; divided by 2 : offset X done !
mulu.w  #160,d1             ; multiplication by 160 : offset Y done !
add.w   d0,d1               ; position in the screen : $7A060
move.w  (a0,d1.l),d0        ; get bitplane word : $0080
bset    d2,d0               ; activate the bit for display
move.w  d0,(a0,d1.l)        ; display pixel on screen offset

move.w  #7,-(sp)            ; wait for a key press
trap  #1
addq.l  #2,sp

movem.l sav_pal,d0-d7       ; restore palette system
movem.l d0-d7,$ffff8240.w
rts

sav_pal   ds.l 8
```

Afficher un point ne prend pas trop de temps au niveau du processeur mais si on veut en afficher plus, on peut aller jusqu'à un maximum d'environ 400 points à l'écran par frame. Et comme vous avez pu le voir dans le code assembleur ci-dessus, tout reste encore à optimiser comme par exemple supprimer la multiplication et créer une table de multiplication sachant que l'on utilise 200 lignes au total.
Et puis utiliser un seul plan nous fait gagner en instructions avec l'utilisation d'un Word, donc peu coûteux en ressources machine ; il existe tellement de « tricks » que les codeurs n'ont pas fini de nous étonner, comme par exemple dans cette démo 4 où le nombre de points à afficher a été multiplié par 3. Et cet effet affiche toutes les dots en temps réel ! Dans un prochain article nous parlerons de l'utilisation de graphismes de tailles diverses tels que des sprites, éléments de zones graphiques qui font partie d'un jeu par exemple. Et bien sûr dans cet article je vous expliquerai comment on gère les déplacements de zone graphique un peu plus volumineux qu'un point sur un Atari ST.
Les sources en assembleur sont disponibles sur :
https://github.com/NoExtra-Team/framework, plus précisément dans le répertoire framework / sources / PIXEL. •