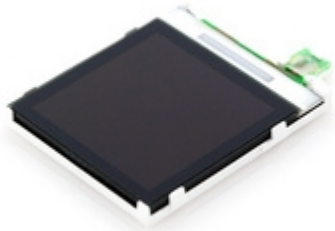
Ecran LCD



1. Présentation

La carte Olimex SAM7-EX256 comporte un écran LCD Nokia 6610. Il s'agit d'un écran à 132x132 pixels codés sur 12 bits (soit 4096 couleurs), avec un éclairage arrière intégré.

Ces écrans ont l'avantage d'être très peu coûteux, cependant il en existe deux types différents :

- Les écrans avec une étiquette “GE8” et un connecteur **vert** sont équipés d'un contrôleur Epson S1D15G00
- Les écrans avec une étiquette “GE12” et un connecteur **orange** sont équipés d'un contrôleur Philips PCF8833

L'initialisation des écrans se fait différemment selon le type de contrôleur, il faut donc bien faire attention à la couleur du connecteur de l'écran (la petite partie rectangulaire qui dépasse vers le haut).
Toutes les cartes utilisées en TP semblent avoir un écran à connecteur vert, donc avec un contrôleur Epson.

2. Initialisation du SPI

L'écran LCD est contrôlé par les ports **PA2** (reset) et **PB20** (backlight), et par l'interface série **SPI0** via la ligne **SPI0_NPCS0**. Les opérations sur l'écran sont effectuées en envoyant des commandes codées sur 9 bits via le SPI. Le bit 8 est mis à 1 lorsque la donnée envoyée n'est pas une commande, mais un paramètre passé à la commande envoyée précédemment.

Pour initialiser l'écran, il faut d'abord initialiser le SPI0.

a. Activation de l'horloge

En premier lieu, il faut initialiser l'horloge du SPI0 dans le registre **PMC_PCER** du PMC (Power Management Controller). Ce registre comporte un bit pour chaque périphérique, écrire 1 dans un bit active l'horloge pour le périphérique correspondant.
Le SPI0 correspond au périphérique 4, il faut donc écrire 1<<4 dans **PMC_PCER**.

L'horloge de certains composants (notamment du PIO) est activée par défaut lorsque la carte est réinitialisée, mais en général, il est plus prudent de s'assurer que l'horloge d'un composant est activée avant de l'utiliser.

b. Désactivation des ports parallèles

Le SPI0 utilise les ports 12, 16, 17 et 18 du PIOA pour communiquer, il faudra donc indiquer au PIO qu'il ne pourra pas utiliser ces ports et que ces derniers seront attribués au SPI0.

Cela est possible en écrivant (1<<12|1<<16|1<<17|1<<18) dans le registre **PIO_PDR** du PIOA, et également dans le registre **PIO_ASR** pour indiquer que le périphérique A attribué à ces ports en prendra le contrôle. (chaque port du PIO est multiplexé et peut être utilisé par 3 périphériques différents, soit le PIO lui-même, soit le périphérique A, soit le périphérique B, la nature de ces périphériques pour chaque port étant détaillée dans la documentation du processeur).

c. Configuration

Une fois l'horloge activée, il faut configurer le SPI (on utilisera comme registre de base **SPI0_BASE**, qui correspond au premier contrôleur SPI).

On commence par réinitialiser le contrôleur en écrivant dans **SPI_CR** (SPI Control Register), dont la structure ressemble à ceci :

31	30	29	28	27	26	25	24
-	-	-	-	-	-	-	LASTXFER
23	22	21	20	19	18	17	16
-	-	-	-	-	-	-	-
15	14	13	12	11	10	9	8
-	-	-	-	-	-	-	-
7	6	5	4	3	2	1	0
SWRST	-	-	-	-	-	SPIDIS	SPIEN

Les bits nous intéressant sont **SWRST** (Software Reset) et **SPIEN** (SPI Enable), correspondant respectivement aux bits 7 et 0. Il faut donc écrire (1<<7|1) dans **SPI_CR**, cela aura pour effet de réinitialiser le SPI et de l'activer par la même occasion.

Une fois le SPI activé, il faut s'occuper de la configuration. Pour cela, on écrira dans le registre **SPI_MR** (SPI Mode Register), dont la structure est la suivante :

31	30	29	28	27	26	25	24
DLYBCS							
23	22	21	20	19	18	17	16
-	-	-	-	PCS			
15	14	13	12	11	10	9	8
-	-	-	-	-	-	-	-
7	6	5	4	3	2	1	0
LLB	-	-	MODFDIS	-	PCSDEC	PS	MSTR

Les parties intéressantes ici sont **MSTR** (Master), **MODFDIS** (Mode Fault Disable) et **PCS** (Peripheral Chip Select).

Dans le cas de l'écran, on veut avoir le contrôle complet sur l'envoi des données, on va donc opérer en mode maître, il faut donc écrire 1 dans le bit 0 (MSTR).

Le « mode fault detection » permet de désactiver automatiquement le SPI lorsque certaines erreurs se produisent. Cette fonctionnalité est rarement utilisée, on écrira donc 1 dans le bit 4 (MODFDIS) pour la désactiver.

Enfin, PCS contient un masque sur 4 bits permettant d'indiquer sur quelle ligne les données seront envoyées, un bit à 0 correspondant à une ligne ouverte. L'écran LCD est relié à la ligne **NPCS0** et correspond donc au bit 0 du masque, on écrira donc la valeur binaire 1110 (0xE en hexadécimal) dans les bits 16 à 19 (PCS).

La valeur à écrire dans **SPI_MR** est donc égale à (0xE<<16|1<<4|1).

Enfin, pour terminer la configuration du SPI, il faut configurer la ligne utilisée, ici la ligne 0. On écrira donc dans le registre **SPI_CSR0** (SPI Chip Select Register 0) dont la structure ressemble à ceci :

31	30	29	28	27	26	25	24
DLYBCT							
23	22	21	20	19	18	17	16
DLYBS							
15	14	13	12	11	10	9	8
SCBR							
7	6	5	4	3	2	1	0
BITS				CSAAT	-	NCPHA	CPOL

On voudra ici mettre **CPOL** (Clock Polarity, bit 0) à 1 pour configurer l'horloge de la ligne en logique inverse.
Les bits 4 à 7 (**BITS**) correspondent au nombre de bits par transfert moins 8. Les commandes de l'écran LCD étant codées sur 9 bits (soit 8+1), on écrira 1 dans cette partie.

SCBR (Serial Clock Baud Rate, bits 8 à 15) est un diviseur donnant le débit de transfert sur la ligne. La fréquence du contrôleur se calcule en divisant la fréquence du processeur (MCK) par la valeur contenue dans SCBR.
La fréquence maximale conseillée pour le contrôleur série est 6 MHz (une fréquence plus importante entraînera notamment une perte de données). La fréquence du processeur étant d'environ 48 MHz, on écrira la valeur 8 dans SCBR (48/8 = 6).

DLYBS (Delay Before SPCK, bits 16 à 23) correspond au nombre minimum de tics de l'horloge processeur entre l'activation de la ligne et le premier transfert. On mettra la valeur la plus petite possible, soit 1.

DLYBCT (Delay Between Consecutive Transfers, bits 24 à 31) correspond au nombre minimum de tics de l'horloge processeur entre deux transferts consécutifs, divisé par 32. On mettra ici aussi la valeur la plus petite possible, soit 1.

Au final, il faudra donc écrire la valeur (1<<24|1<<16|8<<8|1<<4|1) dans **SPI_CSR0**.

Le code d'initialisation du SPI0 devrait ressembler à ceci :

```
      ldr   r12, =PIOA_BASE
      ldr   r11, =PIOB_BASE
      ldr   r10, =PMC_BASE
      ldr   r9,  =SPI0_BASE

      @ On active l'horloge du SPI0
      mov   r1, #PID_SPI0
      str   r1, [r10, #PMC_PCER]

      @ On active le périphérique A des ports 12, 16, 17 et 18 du PIOA
      @ (SPI0_NPCS0, SPI0_MISO, SPI0_MOSI, SPI0_SPCK)
      mov   r1, #(1<<12)|(1<<16)|(1<<17)|(1<<18)
      str   r1, [r12, #PIO_PDR]
      str   r1, [r12, #PIO_ASR]

      @ On réinitialise le SPI0
      mov   r1, #(1<<7|1)
      str   r1, [r9, #SPI_CR]

      @ On configure le SPI0
      ldr   r1, =(0xE<<16|1<<4|1)
      str   r1, [r9, #SPI_MR]

      @ On configure la ligne 0 du SPI0
      ldr   r1, =(1<<24|1<<16|8<<8|1<<4|1)
      str   r1, [r9, #SPI_CSR0]
```

3. Configuration de l'écran

Une fois le SPI initialisé, on peut réinitialiser l'écran en mettant le port PA02 à l'état bas, puis en le remettant à l'état haut un peu plus tard (une boucle s'exécutant 1024 fois suffit).
L'écriture d'une fonction de délai s'avérera utile pour la suite :

```
delay_lcd:
      subs  r0, r0, #1
      bne   delay_lcd
      mov   pc, lr
```

On pourra également allumer l'éclairage arrière en mettant le port PB20 à l'état haut.

Avant d'utiliser l'écran, il faut cependant le configurer. La configuration s'effectue en envoyant des commandes, il vaut mieux donc écrire deux fonctions qui seront souvent utilisées, une pour envoyer une commande, une autre pour envoyer un argument de commande. La première enverra simplement un entier sur 9 bits dont le bit 8 est mis à 0, la seconde fera la même chose mais en mettant ce bit 8 à 1 (pour indiquer qu'il ne s'agit pas d'une commande).

Lors de l'envoi de données, il faut aussi s'assurer que tous les envois sont terminés en attendant que le bit 9 de **SPI_SR** (nommé TXEMPTY dans la documentation du processeur) passe à 1.

```
lcd_write_command:
                  stmfd       sp!, {r1, r9}
                  ldr         r9,  =SPI0_BASE
write_lcd_wait:
                  @ On attend la fin de tous les envois en attente
                  ldr         r1, [r9, #SPI_SR]
                  tst         r1, #1<<9
                  beq         write_lcd_wait

                  @ On envoie les données au SPI
                  str         r0, [r9, #SPI_TDR]

                  ldmfd sp!, {r1, r9}
                  mov         pc, lr
```

L'envoi d'un argument de commande se déroule exactement de la même manière, il suffit de mettre le bit 8 à 1 au lieu de le mettre à 0 en utilisant l'instruction `orr`.

```
lcd_write_data:
                  @ Le bit 8 est mis à 1 si la donnée envoyée
                  @ n'est pas une commande
                  orr         r0, r0, #0x100
                  stmfd       sp!, {r1, r9}
                  ldr         r9,  =SPI0_BASE
write_lcd_wait:
                  @ On attend la fin de tous les envois en attente
                  ldr         r1, [r9, #SPI_SR]
                  tst         r1, #1<<9
                  beq         write_lcd_wait

                  @ On envoie les données au SPI
                  str         r0, [r9, #SPI_TDR]

                  ldmfd sp!, {r1, r9}
                  mov         pc, lr
```

Les commandes à envoyer diffèrent selon le type de contrôleur. On s'intéressera ici à l'initialisation du contrôleur Epson, étant donné que toutes les cartes en salle de TP disposent de ce contrôleur.

Il faudra d'abord définir le code de chaque commande comme ceci :

```
.equ  DISON       ,0xAF
.equ  DISOFF      ,0xAE
.equ  DISNOR      ,0xA6
.equ  DISINV      ,0xA7
.equ  COMSCN      ,0xBB
.equ  DISCTL      ,0xCA
.equ  SLPIN       ,0x95
.equ  SLPOUT      ,0x94
.equ  PASET       ,0x75
.equ  CASET       ,0x15
.equ  DATCTL      ,0xBC
.equ  RGBSET8     ,0xCE
.equ  RAMWR       ,0x5C
.equ  RAMRD       ,0x5D
.equ  PTLIN       ,0xA8
.equ  PTLOUT      ,0xA9
.equ  RMWIN       ,0xE0
.equ  RMWOUT      ,0xEE
.equ  ASCSET      ,0xAA
.equ  SCSTART     ,0xAB
.equ  OSCON       ,0xD1
.equ  OSCOFF      ,0xD2
.equ  PWRCTR      ,0x20
.equ  VOLCTR      ,0x81
.equ  VOLUP       ,0xD6
.equ  VOLDOWN     ,0xD7
.equ  TMPGRD      ,0x82
.equ  EPCTIN      ,0xCD
.equ  EPCOUT      ,0xCC
.equ  EPMWR       ,0xFC
.equ  EPMRD       ,0xFD
.equ  EPSRRD1     ,0x7C
.equ  EPSRRD2     ,0x7D
.equ  NOP         ,0x25
```

On notera l'envoi d'une commande **COMMANDE(arg1, arg2, ...)**, où COMMANDE est le nom de la commande à envoyer, et arg1, arg2, ... les arguments à envoyer avec la commande. Le code assembleur correspondant s'écrira donc :

```
mov         r0, #COMMANDE
bl          lcd_write_command
mov         r0, #arg1
bl          lcd_write_data
mov         r0, #arg2
bl          lcd_write_data
...
```

La plupart des commandes suivantes servent à initialiser des paramètres internes à l'écran, on n'a généralement pas besoin de connaître leur signification exacte. Pour plus d'informations sur le contrôleur Epson, voir le lien suivant :
http://www.sparkfun.com/datasheets/LCD/S1D15G10D08BE_TM_MF1493_03.pdf

Tout d'abord, on utilise la commande DISCTL (Display Control) pour initialiser les paramètres de l'horloge du contrôleur. Cette commande prend 3 paramètres, le premier détermine le nombre de divisions de l'horloge interne (on utilisera la valeur par défaut 0), le second doit être égal au nombre de lignes de l'écran divisé par 4 plus 1 (soit 132/4 + 1 = 32), et le dernier correspond à une fonctionnalité peu utilisée, on la désactivera donc en y mettant 0.
On enverra donc d'abord **DISCTL(0, 32, 0)**.

Ensuite, on passe à la commande COMSCN (Common Scan direction) pour régler la direction de balayage de l'écran. La seule valeur produisant un affichage correct est 1, on enverra donc **COMSCN(1)**.

Il faut aussi activer l'oscillateur interne de l'écran, simplement en envoyant la commande **OSCON** (Oscillator On, sans argument).

Avant de poursuivre, on pensera aussi a « réveiller » l'écran (qui est à l'état « sommeil » lorsqu'il est réinitialisé) avec la commande **SLPOUT** (Sleep Out, également sans argument).

La commande PWRCTR (Power Control) permet d'activer ou de désactiver les régulateurs de tension. On les activera tous en passant 0xF en paramètre, on enverra donc **PWRCTR(0xF)**.

Par défaut, l'affichage de l'écran se fait de bas en haut en lisant les composantes de couleur dans l'ordre bleu-vert-rouge. En inversant l'affichage avec DISINV (Display Invert), l'affichage de l'écran se fera de haut en bas dans l'ordre rouge-vert-bleu, ce qui est exactement ce dont on a besoin. Il suffit d'envoyer la commande **DISINV** (sans argument).

Il faut à présent configurer le mode d'affichage de l'écran avec la commande DATCTL (Data Control), prenant trois arguments. Le premier contient des bits permettant d'activer ou de désactiver certains paramètres d'adressage de l'écran, on mettra ici la valeur par défaut 0. Le second argument permet de régler l'ordre des composantes rouge-vert-bleu, on laissera également la valeur par défaut 0. Le troisième argument permet de modifier le nombre de bits par pixel, on choisira la valeur 2, qui correspond à 12 bits par pixel.
On enverra donc **DATCTL(0, 0, 2)**.

Il ne reste plus qu'à régler le contraste avec la commande VOLCTR (Volume Control). Le premier argument doit être compris entre 0 et 63 et correspond au contraste, une valeur plus élevée correspondant à un contraste plus élevée. On choisira la valeur produisant l'affichage le plus satisfaisant, 36 étant une bonne valeur de départ. Le second argument doit toujours être égal 3.
On enverra donc **VOLCTR(36, 3)** (la valeur 36 étant à modifier éventuellement, chaque écran pouvant avoir un contraste différent pour une même valeur donnée).

Une fois la configuration terminée, il faut attendre environ 10000 cycles d'horloge pour que l'alimentation de l'écran se stabilise. On pourra appeler la fonction `lcd_delay` précédemment déclarée avec un paramètre égal à 10240 (une valeur immédiate codable pouvant être utilisée avec l'instruction `mov`). Ensuite, on peut enfin activer l'affichage de l'écran avec la commande **DISON** (Display On, sans argument).

La séquence d'initialisation de l'écran devrait ressembler à ceci :

```
	ldr   r12, =PIOA_BASE
	ldr   r11, =PIOB_BASE

	@ On allume l'éclairage de l'écran (PB20)
	mov   r1, #1<<20
	str   r1, [r11, #PIO_PER]
	str   r1, [r11, #PIO_OER]
	str   r1, [r11, #PIO_SODR]

	@ On réinitialise l'écran (PA02)
	mov   r1, #1<<2
	str   r1, [r12, #PIO_PER]
	str   r1, [r12, #PIO_OER]
	str   r1, [r12, #PIO_CODR]
	mov   r0, #1024
	bl    delay_lcd
	str   r1, [r12, #PIO_SODR]

	@ On configure les paramètres d'horloge du contrôleur
	mov   r0, #DISCTL
	bl    lcd_write_command
	mov   r0, #0
	bl    lcd_write_data
	mov   r0, #32
	bl    lcd_write_data
	mov   r0, #0
	bl    lcd_write_data
```

```
    @ On paramètre la direction de balayage
    mov   r0, #COMSCN
    bl    lcd_write_command
    mov   r0, #1
    bl    lcd_write_data

    @ On active l'oscillateur interne
    mov   r0, #OSCON
    bl    lcd_write_command

    @ On réveille l'écran
    mov   r0, #SLPOUT
    bl    lcd_write_command

    @ Contrôle de la puissance
    mov   r0, #PWRCTR
    bl    lcd_write_command
    mov   r0, #0xF
    bl    lcd_write_data
    mov   r0, #3
    bl    lcd_write_data

    @ On inverse l'affichage
    mov   r0, #DISINV
    bl    lcd_write_command

    @ Configuration de l'écriture de données
    mov   r0, #DATCTL
    bl    lcd_write_command
    mov   r0, #0
    bl    lcd_write_data
    mov   r0, #0
    bl    lcd_write_data
    mov   r0, #2
    bl    lcd_write_data

    @ Contrôle du contraste
    mov   r0, #VOLCTR
    bl    lcd_write_command
    mov   r0, #36
    bl    lcd_write_data
    mov   r0, #3
    bl    lcd_write_data

    @ On donne le temps à l'alimentation de l'écran de se stabiliser
    mov   r0, #10240
    bl    delay_lcd

    @ On active l'écran
    mov   r0, #DISON
    bl    lcd_write_command
```

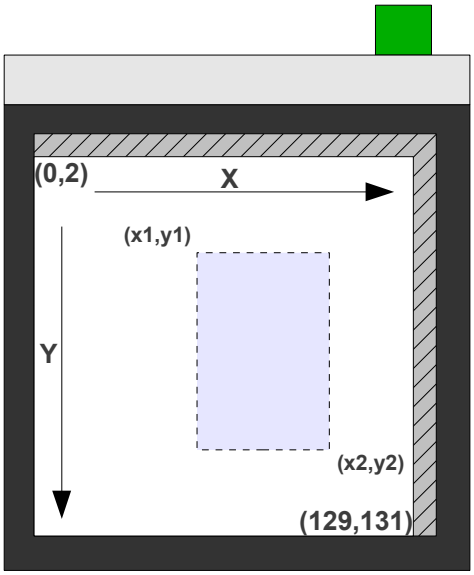
4. Dessin

Avant toute opération de dessin, il est nécessaire de définir un rectangle d'écriture. Il s'agit d'une zone rectangulaire de l'écran faisant office de « sous-écran » dans laquelle l'écriture de pixels se fera.

Pour définir un rectangle d'écriture, on utilisera les commandes **PASET** (Page Set) et **CASET** (Column Set).
Ces deux commandes prennent chacune deux arguments, l'ordonnée de départ et d'arrivée pour **PASET**, l'abscisse de départ et d'arrivée pour **CASET**.
Donc en envoyant **PASET(y1, y2)** puis **CASET(x1, x2)**, on définit un rectangle d'écriture de coordonnées (x1, y1, x2, y2). L'ordre dans lequel les commandes sont envoyées n'est pas important, cependant il faut veiller à ce que y1 <= y2 et x1 <= x2.

Il faut également noter que la surface visible de l'écran est de 130x130 pixels et non 132x132. En effet, l'écran comporte un bandeau invisible de 2 pixels de hauteur dans le bord haut de l'écran, et un autre bandeau invisible de 2 pixels de largeur dans le bord droit.
Les cordonnées du coin supérieur gauche de l'écran sont donc (0, 2), il faut donc ajouter 2 à l'ordonnée lors de chaque opération sur l'écran.

Le système de coordonnées de l'écran lorsqu'il est configuré comme décrit précédemment ressemble donc à ceci (la position du rectangle d'écriture étant évidemment complètement arbitraire) :

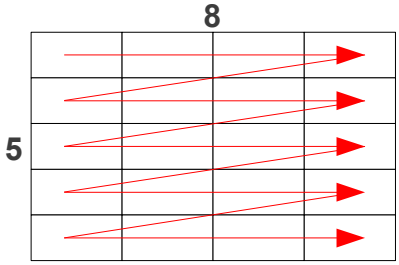


Une fois le rectangle d'écriture établi, on peut commencer à y écrire des données. La commande **RAMWR** (RAM Write) doit être envoyée pour faire passer l'écran en mode écriture. Toutes les données envoyées en tant qu'arguments de commande seront ainsi écrites dans la mémoire de l'écran.

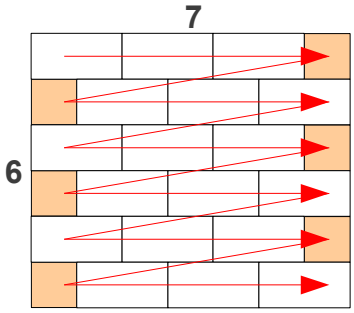
L'écran est configuré en mode 12 bits par pixel, ce qui correspond au nombre de couleurs maximum. Les pixels doivent donc être envoyés par groupes de deux, en envoyant trois octets à la suite (2*12 = 24 = 3*8).
Un pixel est codé selon ses trois composantes rouge, vert et bleu, une composante est donc codée sur 4 bits (12/3 = 4), et peut donc être comprise entre 0 et 15.
Deux pixels consécutifs de couleur (R1, V1, B1) et (R2, V2, B2) seront donc codés par les trois bits suivants :

	7	6	5	4	3	2	1	0
Argument 1	R1				V1			
Argument 2	B1				R2			
Argument 3	V2				B2			

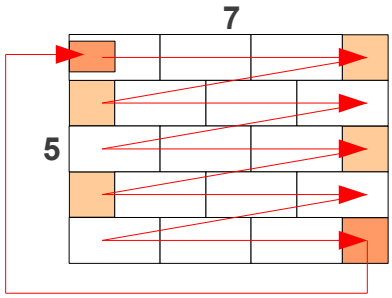
L'intérêt particulier du rectangle d'écriture est principalement le « retour à la ligne » automatique. Autrement dit, une fois le rectangle d'écriture établi, il suffit d'envoyer des paires de pixels les unes à la suite des autres, qui iront remplir ce rectangle de gauche à droite en partant du coin supérieur gauche. Dès que l'écriture atteint le bord droit du rectangle, un retour à la ligne se fait automatiquement et on se retrouve tout à gauche de la ligne suivante, et ainsi de suite jusqu'à remplir le rectangle.



Si la largeur du rectangle d'écriture est impaire, la dernière paire écrite sur une ligne sera coupée en deux et le deuxième pixel sera reporté sur la ligne suivante.



Si le nombre de pixels total du rectangle d'écriture est impair, l'écriture boucle et le dernier pixel se retrouve reporté au tout début, dans le coin supérieur gauche. Dans le cas d'un rectangle plein, cela ne pose pas de problème, cependant il faut faire attention à cette éventualité lorsque l'on écrit des bitmaps sur l'écran car le premier pixel se retrouve écrasé.
Pour éviter ce problème, on peut soit répéter le premier pixel à la fin du bitmap, soit écrire des zéros dans les 4 bits du deuxième octet du dernier pixel et omettre le troisième octet (dans ce cas le second pixel de la paire ne sera pas écrit).



Ces trois commandes permettent d'effectuer de manière très simple le remplissage d'un rectangle avec une couleur donnée. Étant donné une couleur (R, V, B), la première chose à faire est de transformer cette couleur en 3 octets RV, BR et VB, correspondant à une paire de pixels de couleur (R, V, B).
La principale difficulté consiste à calculer le nombre de paires de pixels requis pour remplir un rectangle d'écriture donné.

Soit un rectangle (x1, y1, x2, y2), avec x1 < x2 et y1 < y2.
Le nombre de pixels contenus dans ce rectangle est donc (x2 – x1 + 1) * (y2 – y1 + 1).
On ajoute 1 pixel, ce qui permet d'ajouter une paire supplémentaire si le nombre total de pixels est impair.
Il suffit alors de diviser le tout par 2 pour obtenir le nombre de paires minimum, la formule finale est donc :

((x2 – x1 + 1) * (y2 – y1 + 1) + 1) / 2

Lorsque toutes les données sont écrites, il est recommandé d'envoyer la commande **NOP** (No Operation, sans argument) pour terminer l'opération d'écriture de façon certaine.

Voici un exemple de fonction permettant de dessiner des rectangles pleins. Les paramètres sont passés par la pile l'un après l'autre dans l'ordre (x1, y1, x2, y2, couleur). Ici le paramètre couleur est passé comme un entier 12 bits, qui s'écrit donc avec 3 chiffres en hexadécimal, chaque chiffre correspondant à une composante de couleur dans l'ordre rouge-vert-bleu (0x000=noir, 0xFFF=blanc, 0xF00=rouge, 0x0F0=vert, 0x00F=bleu, et ainsi de suite) :

```
fillrect:
      stmfd       sp!, {r0, r1, r2, r3, r4, r5, lr}

      ldr         r1, [sp, #11<<2]  @ x1
      ldr         r2, [sp, #10<<2]  @ y1
      ldr         r3, [sp, #9<<2]   @ x2
      ldr         r4, [sp, #8<<2]   @ y2
      ldr         r5, [sp, #7<<2]   @ couleur (RRRRVVVVBBBB)

      @ On positionne le rectangle d'écriture
      mov         r0, #PASET
      bl          lcd_write_command
      mov         r0, r2
      bl          lcd_write_data
      mov         r0, r4
      bl          lcd_write_data
      mov         r0, #CASET
      bl          lcd_write_command
      mov         r0, r1
      bl          lcd_write_data
      mov         r0, r3
      bl          lcd_write_data

      @ On calcule le nombre d'octets à écrire
      @ ((((x2 - x1 + 1) * (y2 - y1 + 1)) / 2) + 1)
      sub         r1, r3, r1
      add         r1, r1, #1
      sub         r2, r4, r2
      add         r2, r2, #1)
      mul         r4, r1, r2
      mov         r4, r4, lsr #1
      add         r4, r4, #1

      @ On segmente la couleur dans 3 registres (3 octets pour 2 pixels)
      mov         r1, r5, lsr #4
      and         r1, r1, #0xFF     @ r1 = RRRRVVVV
      mov         r2, r5, lsl #4
      orr         r2, r5, lsr #8
      and         r2, r2, #0xFF     @ r2 = BBBBRRRR
      and         r3, r5, #0xFF     @ r3 = VVVVBBBB

      @ On se préprare pour écrire les données
      mov         r0, #RAMWR
      bl          lcd_write_command

      @ On écrit les données
fillrect_loop:
      mov         r0, r1
      bl          lcd_write_data
      mov         r0, r2
      bl          lcd_write_data
      mov         r0, r3
      bl          lcd_write_data
      subs        r4, r4, #1
      bne         fillrect_loop

      mov         r0, #NOP
      bl          lcd_write_command

      ldmfd       sp!, {r0, r1, r2, r3, r4, r5, pc}
```

La vérification de x1 <= x2 et y1 <= y2 n'est pas effectuée, on pourra éventuellement l'ajouter et inverser les paramètres dans le cas où elle n'est pas vérifiée.

L'appel de la fonction (pour dessiner un rectangle de (0,10) à (40,50) rempli de pixels blancs) se fait comme ceci :

```
	mov		r0, #0
	stmfd		sp!, {r0}
	mov		r0, #10
	stmfd		sp!, {r0}
	mov		r0, #40
	stmfd		sp!, {r0}
	mov		r0, #50
	stmfd		sp!, {r0}
	ldr		r0, =0xFFF
	stmfd		sp!, {r0}
	bl		fillrect
	add		sp, sp, #5<<2
```

Pour dessiner une ligne horizontale, il suffit de dessiner un rectangle de hauteur 1 (y1=y2). De même, pour dessiner une ligne verticale, il suffit de dessiner un rectangle de largeur 1 (x1=x2). Pour dessiner un pixel unique, il suffit donc de dessiner un rectangle 1x1 (x1=x2 et y1=y2).

Pour dessiner des formes plus complexes, on pourra implémenter l'algorithme de tracé de segment de Bresenham pour tracer des lignes, et l'algorithme de tracé d'arc de cercle de Bresenham pour tracer des cercles.