THE AMSTRAD CPC CRTC COMPENDIUM V1.4

LOGON SYSTEM

SOMMAIRE

1 PRÉAMBULE

L'objectif de ce document est d'apporter des informations détaillées sur le fonctionnement des différents circuits CRTC 6845 implémentés dans les CPC créés par AMSTRAD. Le CRTC est un circuit contrôleur capable de fournir une interface entre des micro-ordinateurs et des écrans cathodiques gérant un balayage vidéo.

Ce document aborde également le fonctionnement de quelques circuits associés aux CRTC, notamment le GATE ARRAY.

Ces informations peuvent servir à tout programmeur de jeu ou de démo travaillant encore sur ces machines conçues durant les années 1980-1990.
Elles peuvent aussi servir de référence à toute personne souhaitant créer un émulateur autrement qu'en adaptant son code au coup par coup pour des programmes spécifiques.

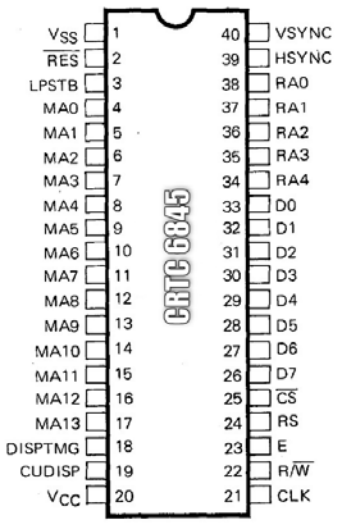
Enfin, il peut apporter des informations utiles aux utilisateurs d'autres machines équipées de CRTC similaires. Il faut cependant considérer que les timings entre le processeur, le CRTC et le circuit vidéo associé d'une autre machine peuvent modifier notablement les comportements décrits ici. Par ailleurs, ce document décrit uniquement les fonctions des CRTC utilisées sur le CPC. Le mode « texte » et le « curseur » ne sont donc pas traités.

Une partie des informations indiquées dans ce document a été vérifiée à partir d'un programme de benchmark appelé SHAKER. Il a été testé pour chaque CRTC sur plusieurs machines. Un document par CRTC existe avec les photos obtenues des résultats de chaque test pour les versions 1.8 et 1.9 de SHAKER. Depuis la version 2.2, un portail existe pour faciliter l'accès, la comparaison et la mise à jour de ces résultats : https://shaker.logonsystem.fr

La vérité tient peu de place, mais l'erreur occupe une infinité de lieux.
Ce document est amené à être corrigé et à évoluer.

Serge Querné
Longshot / Logon System
serge.querne@logonsystem.fr

Remerciements



Arnaud STORQ (*NoRecess*) pour son infinie patience et ses talents cachés de cameraman pour m'envoyer les résultats de SHAKER lorsque mes CPC étaient inaccessibles.

Claire DUPAS (*Cheshirecat*) pour ses tests et son retour d'expérience sur les HSYNC (et ses idées parfois pas si farfelues que ça ;-))

David MANUEL (*DManu78*), auteur de l'excellent émulateur Amspirit, pour nos échanges et tests constructifs afin d'améliorer ce document.

Stéphane SIKORA (*Siko*) pour l'aide apportée sur quelques tests et la réalisation du portail SHAKERLAND.

Marc MAC PHAIDIN (*Lmimmfn*), pour la traduction anglaise de ce document.

2 HISTORIQUE

Version	Date	Mise à jour
1.0	01/01/2021	Création du document Relecture : Arnaud Storq, Claire Dupas, Sébastien Broudin
1.1	15/12/2021	Réimport des schémas. Correctifs §4.3, 4.5, 7.2. Regroupement status sous § 21 avec ajout des registres en lecture et définition des status des CRTC 3 & 4. Correction cpu instructions CPI/CPD/CPIR/CPDR
1.2	30/04/2022	Evolution majeure. Précisions, corrections et clarifications sur quelques chapitres (notamment sur CRTC 0 : VSync, gel C9 sur R0=0, mode « Interlace »). Nouveaux chapitres sur le GATE ARRAY (mode splitting, hsync, interruptions). Descriptif de la RFD. Scroll vertical au 1/64ème de pixel. Ajout de précisions sur le Mode Splitting. Relecture : David Manuel, Claire Dupas, Olivier Antoine.
1.3	14/09/2022	Ajout de précisions 4.4.2, 7.2, 10.3.1, 11.1, 11.2.1, 11.2.3, 11.2.4 ,11.5.2, 11.6, 12.2, 14.2, 15.2, 19.4.1, 19.4.3, 19.6.4 Corrections 7.2, 10.3.3, 11.4, 11.7 12.4, 11.3, 12.4.2 ,13.4, 13.4.1, 15.2.3, 16.4.4, 19.5.3, 19.6.2, 19.6.3, 20.2, 20.5, 25.1 Nouveaux chapitres 16.6, 25.7.2
1.4	11/01/2023	Corrections chapitre 25 (merci à Salvaren). Corrections orthographiques (merci à Sylvestre Campin). Ajout de précisions et corrections majeures sur le mode « Interlace » chapitres 11.7, 16.1, 27.1.9, 11.3.1, 15.2.1, 15.2.3, 15.2.4, 18.2.3, 19. Ajouts importants dans le chapitre 23. Création des chapitres 11.8 et 24 sur les traitements en temps fixe.

3 GÉNÉRALITÉS

3.1TERMINOLOGIE

Le CRTC est un circuit qui passe son temps à compter et à comparer.

Il contient donc principalement des compteurs dont la valeur limite est en général définie dans les registres programmables du circuit.

Dans l'objectif de construire des frames (ou écrans au sens CRTC) composés de plusieurs caractères verticaux, eux-mêmes constitués de plusieurs lignes verticales, elles-mêmes composées de plusieurs caractères horizontaux, certaines terminologies sont associées à ces registres et compteurs.

Il est ainsi question de « *Nombre total horizontal de caractères* », « *Nombre total vertical de caractères* » ou encore « *Raster Maximum* » pour définir la valeur de certains registres, avec parfois quelques différences entre les documentations techniques. Certains émulateurs utilisent des acronymes « non officiels » pour définir le nom des compteurs associés : HCC, VTAC, VLC, VCC,... pour n'en citer que quelques uns.

Ces terminologies ne sont plus du tout adaptées lorsqu'on travaille à un niveau différent de celui prévu pour la construction d'un frame « standard ».

En effet, à un certain niveau, il n'y a plus vraiment de logique de comptage « horizontale » ou « verticale », et il est donc difficile de parler de « caractères » sans se prendre les pieds dans le tapis, car selon la programmation du circuit, un caractère vertical peut aussi être un caractère horizontal.

Cette notion existe tout au plus par rapport à la logique de synchronisation du frame construit.

Étant donné qu'il est possible de réduire la taille d'une ligne à 1 µsec, de réduire la taille d'un caractère vertical à 1 ligne et réduire la taille d'un frame à 1 caractère, les appellations horizontales et verticales perdent quelque peu leur sens.

C'est pourquoi dans ce document je vais en général noter les registres du CRTC sous la forme **Rn** et les compteurs associés par **Cn**. Lorsqu'il sera question de « caractères », il s'agira des mots traités par le GATE ARRAY à partir de l'adresse fournie par le CRTC. J'ai nommé VMA et VMA' les deux pointeurs internes du CRTC.

J'invite les auteurs d'émulateurs se basant sur ce document à adopter ces notations.
Voici une équivalence de quelques termes rencontrés :

Noms identifiés dans certains émulateurs	Compteur
HCC (Horizontal Char Counter)	C0
VLC (Vertical Line Counter)	C9
VCC (Vertical Character Counter)	C4
VSC (Vertical Sync Counter)	C3h
HSC (Horizontal Sync Counter)	C3l
VTAC (Vertical Total Adjust Counter)	C5 or C9 (on CRTC's 0,3,4)
VMA (byte pointer)	VMA or VMA' (word pointer)

3.2ACRONYMES

ASIC : Application Specific Integrated Circuit : Se dit d'un circuit conçu pour une application spécifique. Le GATE ARRAY est un ASIC.

CRTC : Cathod Ray Tube Controller : Circuit qui permet d'interfacer un ordinateur avec un moniteur cathodique capable de gérer des lignes raster.

HBL : Horizontal BLank : Se dit de la période durant laquelle le canon à électrons du moniteur, qui a atteint le côté droit du moniteur, désactive le faisceau pour revenir à gauche de l'écran.

HSYNC : Horizontal SYNC : Se dit du signal émis par le CRTC qui permet au moniteur (via le GATE ARRAY sur CPC) de synchroniser le frame horizontalement sur l'écran.

IVM : Interlace Video Mode

IS : Interlace Sync

PIXEL-M2 : Définition de la taille d'un pixel affiché en MODE 2 (0.0625 µsec)

RFD : Rupture for Dummies : Rupture facile disponible sur le CRTC 1.

RLAL : Rupture Ligne à Ligne : Se dit d'une rupture horizontale où l'adresse est changée sur chaque ligne raster du frame.

RV : Rupture Verticale : Se dit d'une rupture dont l'axe n'est pas « horizontal », et qui permet de créer des zones horizontalement.

RVI : Rupture Verticale Invisible : Se dit de ruptures verticales ayant lieu durant la Hsync et non visibles.

RVLL : Rupture Verticale Last Line : Se dit de ruptures exploitant le traitement « dernière ligne » de C4 des CRTC 0 et 2.

VBL : Vertical BLank : Se dit de la période durant laquelle le canon à électrons du moniteur, qui a atteint le bas de l'écran, désactive le faisceau pour revenir en haut de l'écran.

VMA : Video Memory Address : Pointeur CRTC sur les mots adressés en mémoire et fournis au GATE ARRAY pour l'affichage.

VSYNC : Vertical SYNC : Se dit du signal émis par le CRTC qui permet au moniteur (via le GATE ARRAY sur CPC) de synchroniser le frame verticalement sur l'écran.

4 CRTC & CPC...

4.1GÉNÉRALITÉS

Sur AMSTRAD CPC, la durée d'un caractère CRTC est de 1 µsec.

Ce caractère CRTC représente 2 octets en mémoire.

Le pointeur mémoire est communiqué par le CRTC au GATE ARRAY, **qui va toujours lire la "ram centrale" de 64k**. Le GATE ARRAY ne peut pas lire les données en ROM ou dans la RAM additionnelle des 6128 (ou des extensions mémoire), sauf intervention de Tot0.

Le CRTC est programmé de manière à ce que l'image ainsi créée soit supportée par un moniteur.
Un raccourci de langage consiste à utiliser le terme NOP au lieu de µsec (microseconde) car c'est le temps pris par cette instruction en Z80A.

Sur CPC, les instructions Z80A sont alignées en arrondissant les cycles M d'une instruction sur un multiple de 4 cycles T. Cet alignement est lié à la nécessité pour le GATE ARRAY d'interrompre le Z80A pour accéder à la ram dont l'adresse est fournie par le CRTC. Ce fonctionnement ralentit certaines instructions par rapport à la fréquence d'horloge. Réaliser un code précis nécessite de connaitre le temps exact pris par chaque instruction. Voir chapitre 25, page 255, pour le détail de ces durées pour chaque instruction.

Un peu d'histoire

Les moniteurs et téléviseurs cathodiques sont contraints d'aligner leur fréquence d'affichage au réseau de distribution d'électricité du pays de commercialisation.

En Europe notamment, les téléviseurs construits pour supporter les formats SECAM et PAL fonctionnent avec une fréquence horizontale de 15625 Hz, pour une fréquence verticale de 50Hz.

À noter que le format Américain (et Japonais) était le NTSC (National Television System Committee) avec une fréquence horizontale de 15734 Hz et verticale de 60Hz.

La fréquence horizontale de 15625 Hz est issue de l'utilisation de ligne à retard de 64 µs, inventée pour la conception du format Secam par l'ingénieur Henri de France.
Pour rappel, 64µs = 0.000064 seconde (juste le temps de prendre un café serré).

Le CRTC suit cette logique et traite 1 million de caractères par seconde environ car sa fréquence est de 1 Mhz.

Sans intervention diabolique sur la position de la HSYNC, il est en principe programmé pour générer des lignes complètes de 64 µsec.

Pour se rapprocher de la fréquence voulue, le CRTC est en général programmé pour afficher 312 lignes de 64µs, soit très exactement 19968 µsec (0.019968 secondes).

Cette fréquence étant liée à une horloge, on peut constater des dérives entre 2 machines au bout d'un certain temps.

Le format standard initialisé par la ROM CPC dans un pays Européen est :

- Lignes de 64 caractères "CRTC" horizontaux de 1µs chacun (composées de 40 caractères affichés et 24 non affichés).
- 312 lignes verticales, sous divisées en caractères verticaux de 8 lignes, soit 39 caractères.
- Formule : ((R4+1) x (R9+1)) + R5
- Sur ces 39 lignes de caractères, 25 sont affichées.
- Adresse en ROM de la table CRTC : **&5C5**

Le format standard initialisé par la ROM CPC aux Etats-Unis est :

- Lignes de 64 caractères "CRTC" horizontaux de 1µs chacun (composées de 40 caractères affichés et 24 non affichés).
- 262 lignes verticales, sub divisées en caractères verticaux de 8 lignes, soit 32 caractères + 6 lignes d'ajustement
- Formule : ((R4+1) x (R9+1)) + R5
- Sur ces 32.75 lignes de caractères, 25 sont affichées.
- Adresse en ROM de la table CRTC : **&5D5**

Note 1: C'est le bit 4 du port B du PPI (LK4) qui permet de tester quelle table utiliser.

Note 2 : L'initialisation des registres du CRTC est une des premières choses que la ROM BASSE du CPC effectue à l'allumage et au reset de la machine. L'initialisation des registres commence 64 µsec après la lecture de la première instruction en ROM par le Z80A. Les registres sont initialisés en partant du registre 15 jusqu'au registre 0.

Les moniteurs CTM utilisent les circuits PC1031 (moniteur monochrome vert GT65) et PC1378 (moniteur couleur CTM 644). Ces circuits servent à piloter le déflecteur horizontal pour produire le signal dont le canon a besoin pour le balayage.

Le GATE ARRAY du CPC produit, via son pin 5, un signal de synchronisation composite combinant la VSYNC et la HSYNC, qui arrive au connecteur DIN prévu pour le moniteur (via une résistance 220 Ohm (R137)).

Ce signal composite est décodé par les circuits (PC1031 ou PC1378) du moniteur.
[Le moniteur n'étant pas un téléviseur, les signaux pour l'image sont toujours du RGB]

Le CRTC dispose de registres pour gérer un curseur et lire les données envoyées par un "stylo optique". Les registres relatifs au curseur ne servent pas sur CPC, qui ne gère pas de curseur hardware, en général prévu lorsqu'un mode texte est géré.

Ils peuvent cependant présenter un intérêt car la mise à jour de registres dans ou hors d'une période synchronisation, avec des petites valeurs, pourrait entrainer des conséquences sur d'autres registres.

Le fonctionnement de ces registres n'est pas abordé (pour le moment) dans ce document.

4.2 NUMÉROTATION DES CRTC

AMSTRAD a eu la brillante idée d'utiliser des circuits CRTC 6845 fabriqués par différents constructeurs dans ses machines. Ils ont même conçu des ASIC capables d'émuler son fonctionnement.

Un peu d'histoire

Alors que la société AMSTRAD voulait attaquer le marché américain avec les CPC 6128, un problème a été identifié en ROM avec la valeur du registre 5 (ajustement vertical).

Cette valeur a été fixée à 6 en conformité avec la fréquence souhaitée de 60Hz aux USA (262 lignes), mais faisait (à tort) l'impasse sur le système d'interruption géré par le GATE ARRAY. En effet, sur un CPC Européen, les interruptions débutent 2 lignes (on considèrera qu'on a 1 HSYNC par ligne) après la survenue du signal VSYNC par le CRTC.

Ces interruptions ont une période de 52 "lignes", ce qui donne exactement 6 périodes durant les 312 lignes (voir chapitre 26.6 sur les INTERRUPTIONS). Pour caler 5 périodes de 52 lignes, il aurait fallu programmer 260 lignes et non 262 comme cela a été fait, et donc que R5 soit programmé avec 4 au lieu de 6.

Ces 2 lignes de plus provoquent l'arrivée de l'interruption sur la même ligne que celle sur laquelle le CRTC signale le début de la VSYNC, MAIS avant lui.

Autrement dit, sur un CPC américain, un programme dont le code principal teste l'attente VSYNC via le PPI peut être interrompu pendant cette attente. Si l'interruption dure trop longtemps, à son retour, la boucle d'attente de la VSYNC a loupé le signal (le bit est revenu à 0), et un risque de "deadlock" existe.

Cela pouvait engendrer des problèmes de compatibilité pour des programmes produits en Europe. AMSTRAD a alors décidé de remédier au problème en ….modifiant la table 60 Hz en ROM…. Non…je plaisante…trop simple.

Pour éviter ce « deadlock » sans modifier la ROM, les ingénieurs Amstrad se sont dit qu'ils pouvaient "limiter le problème" en augmentant la durée de la VSYNC. C'était parfaitement possible en utilisant le registre 3 du CRTC, qui avait été programmé avec 8 lignes ….. dans la ROM.

Il est probable qu'ils se soient dit qu'il existait des modèles de CRTC dépourvus de la fonction utilisée pour fixer le nombre de lignes de la VSYNC (ces modèles fixent le nombre de lignes à 16) et décidé ainsi que les CPC américains ne seraient équipés que de CRTC 1 et 2, dépourvus de cette fonction.

Si les ingénieurs AMSTRAD étaient conscients de l'existence de ces différences, on peut supposer que les premiers CRTC utilisés étaient des types 0, puisque cette fonction est utilisée et programmée par la ROM.

De là à pouvoir affirmer que sans un bug lié à un 4 à la place d'un 6 dans la ROM, il n'y aurait qu'un seul type de CRTC utilisé dans tous les CPC, c'est faire fi des considérations commerciales sur le marché des composants par rapport au succès de la machine en Europe…

Plusieurs entreprises ont créé des versions différentes du circuit, implémentant des fonctions complémentaires, comme la programmation du nombre de lignes de la VSYNC que je viens d'évoquer.

On peut cependant déduire que les concepteurs de la ROM BASIC :

- travaillaient originellement avec un CRTC disposant de la possibilité de programmer ce nombre de lignes pour la VSYNC.
- ont estimé que 8 lignes de VSYNC étaient suffisantes pour synchroniser verticalement l'image sur un moniteur CTM.

Au-delà des différences fonctionnelles et techniques documentées dans les guides constructeurs des circuits (appelés « datasheet » avec 2 « e »), ces CRTC ont tendance à se comporter différemment lorsqu'on commence à modifier des registres :

- plusieurs fois au cours du balayage.
- durant ou hors des périodes HSYNC/VSYNC.
- avec la valeur 0, qui est un cas particulier pour la gestion de plusieurs registres.

Ces différences impactent la compatibilité des programmes, notamment lorsqu'il est question de modifier l'adresse du pointeur vidéo en cours de balayage.

Cette technique est encore appelée communément "*Rupture*" car… elle est plus simple et générique à dire que "*Offset Split Screen*".

Les différences de gestion des compteurs (qui peuvent déborder dans quelques situations) induisent généralement un défaut de synchronisation horizontale (R2) et/ou verticale (R7).

Si un code Z80A "attend" le signal VSYNC ou qu'il utilise les interruptions (qui dépendent de la HSYNC), le bordel est accentué.

Un peu d'histoire

À ma connaissance, le premier programme à avoir réalisé un test de CRTC est le jeu "Crafton & Xunk" (Get Dexter), écrit par Rémi Herbulot et Michel RHO en 1986.
Dans ce jeu, le changement d'image a lieu grâce à un scrolling horizontal, qui utilise le registre 2 (positionnement de la HSYNC sur un autre caractère).
Cette méthode, sur un CRTC MOTOROLA, provoque une perte de synchro verticale lorsque la VSYNC se produit durant une HSYNC (Ghost Vsync).
Autrement dit, dans cette situation, le CRTC croit faussement générer un signal VSYNC pour le moniteur.
Je suppose que Rémi Herbulot devait avoir accès, chez Ere Informatique, à un CPC avec un CRTC HITACHI (comme le mien à l'époque) et un autre avec un CRTC MOTOROLA.
Ayant fait ce constat, il a créé un test basé sur la lecture en &BF00 du registre 12, qui permet de distinguer le CRTC HITACHI du CRTC MOTOROLA.
Et il a donc géré un affichage avec et sans scrolling de l'écran.
C'est pourquoi il n'y a pas de scrolling également sur les CPC équipés de CRTC UMC, alors que ce CRTC le permet sans problème, car comme pour le CRTC MOTOROLA, son registre 12 n'est pas lisible.
Pour l'anecdote, j'avais retiré le test "anti scroll" sur le CPC d'un ami pour voir ce que ça donnait, et comme ça fonctionnait (il avait sans doute un CRTC 1), je me suis dit que ce test était prévu pour autre chose. J'étais alors loin de me douter que les CRTC étaient différents.

Un peu d'histoire (encore !)

Lorsque les premières techniques de "rupture" ont commencé à être utilisées massivement dans les démos, les différences entre CRTC ont vite commencé à poser problème.
Et notamment lorsque les programmeurs indépendants (souvent lycéens ou étudiants) ont commencé à constituer la scène « démo » et que leurs démos ont commencé à circuler de manière moins discrétionnaire. Au départ, ces premières démos étaient surtout des introductions pour des jeux craqués qui circulaient dans les cours d'école.
[Je crois qu'il existe un jeu des années 80 qui a rencontré ce type de problème avec le CRTC (à vérifier)]

Les premiers programmeurs de démo n'avaient pas encore constitué de "réseau" (la demoscene) et disposaient en général d'une seule machine sous la main.
Les premières intros et démos circulaient au niveau d'un cercle restreint et très régional. Le "réseau" consistait, dans les années 80, à des échanges postaux, minitels (un ancêtre français d'internet, comme prestel en UK) et des échanges téléphoniques ruineux hors des départements, avec très peu ou pas de relations avec les autres pays.

La communication avec des personnes d'autres régions est d'abord passée par la consultation de petites annonces dans les rares magazines informatiques de l'époque à en disposer, car il était difficile de trouver des coordonnées dans une introduction précédant un jeu craqué. Les programmeurs pouvaient difficilement se rendre compte du résultat produit par leur code sur d'autres machines, et l'inertie était énorme.
Il était difficile d'adapter le code via des échanges postaux, sans disposer de la machine (ou même de vouloir le faire, tout simplement).
Les problématiques rencontrées pouvaient aller de la désynchronisation d'image au plantage pur et simple de la machine.

Il était difficile d'être catégorique sur l'origine effective de ces problèmes, et surtout sur leur étendue. Néanmoins, avec le regroupement des demomakers et l'agrandissement de la demoscene, il a été possible de confronter le code sur différentes machines.

La présence de quelques électroniciens dans les rangs des demomakers a permis d'identifier les coupables… (M. SUGAR est toujours en fuite).

Quelques règles "universelles" issues de démarches empiriques ont été décrites dans des documentations secrètes qui ne le sont pas restées longtemps ("Il ne faut pas faire cette opération ici pour que ça fonctionne partout", par exemple).

À noter que certaines techniques sont toujours à l'heure actuelle considérées impossibles à réaliser d'un CRTC à l'autre (jusqu'à ce document...).

Le CRTC 2 (MOTOROLA) s'est avéré être celui posant problème pour une des techniques les plus utilisées, qui consiste à placer R4=R9=0.

Le code nécessaire pour assurer la compatibilité entre CRTC peut néanmoins être bien plus complexe que de juste savoir quand modifier un registre 9 ou 4, nous allons le voir...

Afin de pouvoir exploiter les règles permettant la compatibilité, il était nécessaire dans un premier temps de pouvoir identifier les différents CRTC.

De nombreuses méthodes existent aujourd'hui. Voir chapitres 27, page 266, et 28, page 268.

La numérotation est restée à ce jour celle fixée en fonction de l'ordre ou j'ai découvert ces circuits avec l'aide de quelques membres du groupe de demomakers Logon System.

Nous avons découvert le CRTC émulé "Pré ASIC" après la sortie de l'AMSTRAD PLUS, et c'est pourquoi son numéro est supérieur à 3, bien qu'il soit sorti plus tôt chronologiquement.

Type	Marque	Modèle
0	HITACHI	HD6845S(P)
0	U.M.C. (United Microcircuits Corporation)	UM6845
1-A	U.M.C. (United Microcircuits Corporation)	UM6845R
1-B	U.M.C. (United Microcircuits Corporation)	UM6845R
2	MOTOROLA	MC6845(P)
3	AMSTRAD	ASIC 40489
4	AMSTRAD	ASIC 40226

Note 1 : Les CRTC 3 et 4 sont des CRTC émulés par des ASIC, mais sont néanmoins des CRTC! Sans activer les fonctions complémentaires de l'ASIC 40489, le comportement de ces deux CRTC ne peut pas être différencié actuellement.

Note 2 : Le "Pré-ASIC" (CRTC 4) avait très certainement été conçu dans l'optique de l'AMSTRAD PLUS, car son compteur C9 est prévu pour être interrompu sur n'importe quelle ligne.
C'est pourquoi positionner R9 à 0 peut se faire sur la dernière ligne d'un caractère (compatibilité CRTC 0) ou sur la première ligne d'un caractère (compatibilité CRTC 1).

Note 3 : Il n'est pas exclu, encore à l'heure actuelle, que certaines séries de CPC, soient équipés de modèles de CRTC différents, disponibles chez les 3 constructeurs. (Merci de me le rapporter si vous découvrez un autre modèle, et que vous êtes sûr que ce n'est pas votre petite sœur qui l'a remplacé pour vous faire une blague).
À priori, le CRTC HD6845S de HITACHI, qui est identifié comme un type 0, se comporte exactement comme le CRTC UM6845 de UMC. Sans doute un échange de bons procédés entre les firmes où c'est juste le marquage qui a changé...
La documentation UMC le précise dans sa table de comparaisons avec les autres circuits.
Il n'y a pas à ce jour de test qui permette la distinction de ces deux circuits, mais c'est peut-être possible via le mode « Interlace » ou la gestion particulière de C0 par le HD6845S.

Note 4 : Programmer R3 avec &8x est une très mauvaise habitude, puisque les CRTC 1 et 2 existent aussi dans les machines Européennes, et qu'ils ne respectent pas cette valeur...

Note 5 : Programmer une modification précise d'un registre CRTC dans une routine d'interruption qui n'interrompt pas un code parfaitement synchronisé est une très mauvaise idée.
(et peut laisser croire (à tort) à une différence de CRTC).

Note 6 : La différence constatée entre deux CRTC UM6845R n'est peut-être pas liée au CRTC. Néanmoins, cette différence notable est constante. (voir chapitre 11.5).

4.3 VUE GENERALE DES REGISTRES

			CRTC 0									CRTC 1, 2									CRTC 3, 4								
Register	**Definition**	**Unit**	**r/w**	7	6	5	4	3	2	1	0	**r/w**	7	6	5	4	3	2	1	0	**r/w**	7	6	5	4	3	2	1	0
R0	Horizontal total character number	Char	w									w									w								
R1	Horizontal displayed character number	Char	w									w									w								
R2	Position of horizontal sync. pulse	Char	w									w									w								
R3	Pulse width of horizontal sync. pulse	Function	w	v	v	v	v	h	h	h	h	w					h	h	h	h	w	v	v	v	v	h	h	h	h
R4	Vertical total character number	Char Row	w									w									w								
R5	Total raster adjust	Scan Line	w									w									w								
R6	Vertical displayed character number	Char Row	w									w									w								
R7	Position of vertical sync. pulse	Char Row	w									w									w								
R8	Interlace Mode and Skew	Function	w	c	c	d	d			i	i	w							i	i	w	c	c	d	d			i	i
R9	Max Scan Line Address	Scan Line	w									w									w								
R10	Cursor start	Scan Line	w		b	p						w		b	p						r	s	1	s	s	s	s	s	s
R11	Cursor end	Scan Line	w									w									r	s	0	s	1	s	s	s	s
R12	Display start address (High)	Pointer	r/w									w									r/w								
R13	Display start address (Low)	Pointer	r/w									w									r/w								
R14	Cursor address (High)	Pointer	r/w									r/w									r/w								
R15	Cursor address (Low)	Pointer	r/w									r/w									r/w								
R16	Light Pen (High)	Pointer	r									r									r								
R17	Light Pen (Low)	Pointer	r									r									r								
Access ports to the CRTC on CPC			**r/w**	7	6	5	4	3	2	1	0	**r/w**	7	6	5	4	3	2	1	0	**r/w**	7	6	5	4	3	2	1	0
&BC00	Register selection	Number	w									w									w								
&BD00	Register write	Value	w									w									w								
&BE00	Register status	Function										r		L	b						r								
&BF00	Register read	Value	r									r									r								

Voir chapitres 21, 27.1.8 et 27.1.9 pour obtenir plus d'informations sur le contenu du registre de status et du registre de lecture selon les différents CRTC.

4.4 ACCÈS AU CRTC

4.4.1 GÉNÉRALITÉS

L'accès aux entrées/sorties avec un Z80A nécessite en général d'utiliser des instructions spécifiques.

Ces instructions (OUT, OUTI, INI, IND...) sont en principe prévues pour utiliser des périphériques dont les adresses sont définies sur les 8 bits de poids faible du bus d'adresse 16 bits.

Le bus d'adresse 16 bits est spécifié dans le registre **BC**, mais certaines instructions (OUTD, OUTI, INI, IND, ...) utilisent aussi **B** comme compteur.

Dans l'optique de conception d'un ordinateur, il n'est pas conseillé de placer l'adresse des périphériques pouvant utiliser ces instructions sur les 8 bits de poids fort du bus d'adresse.

Ce sage conseil de M. ZILOG n'a pas été écouté par M. SUGAR.

Aussi l'accès aux périphériques sur CPC passe principalement par les bits **A8..A15 du bus d'adresse**. (le FDC 765 faisant partiellement exception)

Les bits de sélection des périphériques doivent en conséquence être positionnés sur B.

Bienvenue sur le CPC!

Exemples		
Selection registre 12		
	BASIC	OUT &BC00,12
	Z80A	LD BC,&BC00+12:OUT(C),C
Envoi valeur &30 sur registre 12 (sélectionné précédemment)		
	BASIC	OUT &BD00,&30
	Z80A	LD BC,&BD30:OUT (C),C
Lecture registre 12 (sélectionné ou mis à jour précédemment)		
	BASIC	PRINT INP(&BF00)
	Z80A	LD BC,&BF00 : IN A,(C)
Lecture registre de status		
	BASIC	PRINT INP(&BE00)
	Z80A	LD BC,&BE00 : IN A,(C)

L'utilisation de B condamne l'utilisation d'instructions intéressantes permettant d'envoyer ou lire des séries de valeurs successives sur un port, comme OTIR, OTDR, INIR, INDR.

En effet ces instructions utilisent B comme compteur du nombre de valeurs à lire dans une table, et décrémentent ce compteur jusqu'à atteindre la valeur 0.

Remarque digressive :

Ces instructions répétitives peuvent être expérimentalement utilisées, pour traiter plus d'une valeur, sur un périphérique dont les bits de sélection ne participent pas au compteur (bits "haut"), comme le GATE ARRAY.
Il est possible de lancer une de ces instructions et de l'interrompre sauvagement en la positionnant judicieusement avant qu'une interruption survienne.
Cette interruption devra néanmoins "oublier" l'adresse de retour placée sur la pile avant de ré-autoriser d'autres interruptions.
L'intérêt est cependant très limité (et surtout ludique) et pourra activer d'autres périphériques selon le nombre de valeurs lues dans la table.

Il reste néanmoins possible d'utiliser

- Les instructions OUTI ou OUTD pour envoyer une par une des données issues d'une table (pointée par HL) au CRTC.
- Les instructions INI ou IND pour lire une par une des données issues d'une table (pointée par HL) du CRTC.

L'intérêt est que ces instructions sont "rapides" au regard du nombre d'opérations effectuées.

L'utilisation de l'instruction OUTI/OUTD est possible en incrémentant préalablement B entre chaque instruction pour OUTI/OUTD car B est décrémenté avant l'accès au port.
Pour le CRTC, qui utilise les bits 0 et 1 de B comme index, cela implique que B soit pré-chargé avec l'adresse du port + 1 pour que les instructions OUTI/OUTD fonctionnent.

Pour les instructions de lecture, le périphérique adressé est défini par BC avant la décrémentation de B, qui doit donc contenir l'adresse normale du port avant lecture.

Exemple en Z80A :

```
                LD BC,&BC02         ; Sélection registre position Hsync
                OUT (C),C
                LD HL,TABHSYNC      ; Pointeur sur la table des positions Hsync
                LD B,&BD+1          ; Port envoi donnée + 1, soit &BE
                OUTI                ; OUTI décrémente B, envoie TABHSYNC[0] sur le port &BD
                                    ; et incrémente le pointeur
                INC B               ; B est remis sur &BE
                OUTI                ; OUTI décrémente B, envoie TABHSYNC[1] sur le port &BD
                                    ; et incrémente le pointeur
TABHSYNC        DB 50, 10
```

4.4.2 INSTRUCTIONS Z80A

Instructions en Z80A permettant de traiter une entrée sortie :

INSTRUCTIONS	DUREE	DESCRIPTION
OUT (C),r8	4 µsec	r8=[A ou B ou C ou D ou E ou H ou L] Écriture sur les périphériques définis dans BC de la valeur contenue dans le registre r8.
OUT (n),A	3 µsec	L'adresse d'entrée sortie est définie par le couple An, et la donnée envoyée au périphérique est A. Cette double contrainte limite drastiquement le nombre de valeurs disponibles pour un périphérique sans provoquer un effet collatéral sur un autre périphérique (sélection+valeur).
OUT (C),0	4 µsec	Écriture sur les périphériques définis dans BC de la valeur 0. Une instruction intéressante pour un demomaker! Mais pas que…
OUTI	5 µsec	Décrémentation de B, puis lecture de la valeur pointée par HL, incrémentation de HL et envoi de la valeur lue sur le port adressé par BC
OUTD	5 µsec	Décrémentation de B, puis lecture de la valeur pointée par HL, décrémentation de HL et envoi de la valeur lue sur le port adressé par BC
IN r8,(C)	4 µsec	r8=[A ou B ou C ou D ou E ou H ou L] Lecture dans le registre r8 de la valeur envoyée par le périphérique défini dans BC. Si plusieurs périphériques sont sélectionnés, nul doute qu'un jeu des chaises musicales va avoir lieu…
IN A,(n)	3 µsec	L'adresse d'entrée sortie est définie par le couple An, et la donnée lue va modifier A.
IN (C)	4 µsec	C'est une instruction « non officielle » La valeur présente sur le bus de données est lue et son évaluation affecte F
INI	5 µsec	Lecture de la valeur sur le port adressé par BC et écriture de cette valeur à l'adresse pointée par HL, incrémentation de HL, décrémentation de B
IND	5 µsec	Lecture de la valeur sur le port adressé par BC et écriture de cette valeur à l'adresse pointée par HL, décrémentation de HL, décrémentation de B

Lorsqu'une I/O a lieu différents signaux du Z80A peuvent être activés.
Le signal MREQ est placé à l'état haut et IORQ à l'état bas. Les broches RD/WR servent en principe à indiquer si le Z80A doit lire ou écrire une donnée.

Le GATE ARRAY n'est accessible qu'en écriture, et la broche RD est à l'état inactif, ce qui implique qu'une lecture sur ce circuit n'est pas prise en compte. Au mieux on récupère un état de haute impédance disponible sur le bus de données.

Cependant, les CRTC ne sont pas connectés aux broches RD et WR du Z80A, et il n'y a donc aucune détection du sens de l'entrée-sortie. En conséquence, si une instruction de lecture est utilisée sur un registre en écriture du CRTC, alors une donnée est envoyée au CRTC.

Avec une instruction IN A,(C), cela revient à envoyer la donnée présente sur le bus de données au CRTC. Mais avec l'instruction IN A,(n), il est possible d'envoyer le contenu de A sur le port An, tout comme on peut le faire avec l'instruction OUT (n),A. Il est donc possible d'envoyer une valeur au CRTC tout en modifiant A. Il serait cependant hasardeux de se fier à la valeur renvoyée.

Exemple en Z80A:

```
LD A,%00011001   ; Les bits 0 et 1 sélectionnent le port Write du CRTC
IN A,(#ff)       ; Envoie la valeur de A sur le dernier registre CRTC sélectionné
```

4.4.3 DELAIS D'ACCES

Le tableau suivant indique pour quelques instructions d'écriture le moment ou le registre est effectivement mis à jour dans le circuit et pour quelques instructions de lecture le moment ou la valeur dans le circuit est effectivement mise à jour dans le registre ou la ram.

INSTRUCTIONS	DUREE	PRISE EN COMPTE I/O	
		CRTC 0, 1, 2	CRTC 3, 4
OUT (C),r8	4 µsec	**3ème µsec**	**4ème µsec**
OUT (C),0	4 µsec	**3ème µsec**	**4ème µsec**
OUT (n),A	3 µsec	3ème µsec	3ème µsec
OUTI	5 µsec	5ème µsec	5ème µsec
OUTD	5 µsec	5ème µsec	5ème µsec
IN r8,(C)	4 µsec	4ème µsec	4ème µsec
INI	5 µsec	4ème µsec	
IND	5 µsec	4ème µsec	
IN A,(n)	3 µsec	3ème µsec	3ème µsec

Il est important de noter que la prise en compte d'une écriture durant la microseconde de mise à jour n'a pas lieu au même « moment » selon les instructions utilisées, et cela peut donc avoir une incidence sur la gestion de la valeur par le circuit.

Une manière de mesurer cette différence est d'utiliser des processus non « ralentis » par le CRTC et le GATE ARRAY, comme l'affichage de la HSYNC par exemple. Voir chapitre 16, page 140.

On peut aussi mesurer cette différence, par exemple, entre ce qui se produit avec une I/O sur le 3ème NOP d'un OUT(C),R8 et le 5ème NOP d'un OUTI sur un CRTC 1. Voir chapitre 13.6.2, page 115.

Il faut aussi noter que sur les CRTC 3 et 4, le CRTC loupe l'I/O présente sur la 3ème µsec d'un OUT(C),r8 et la récupère sur la 4ème µsec de l'instruction. Ceci retarde la mise à jour des registres de 1 µsec si cette instruction est utilisée. **Ce décalage ne se produit pas si l'instruction OUTI est utilisée**. Voir chapitre suivant.

La majorité des schémas qui font référence à des entrées-sorties dans ce document sont réalisés sur la base de l'instruction OUT(C),r8. Pour les CRTC 3 et 4, l'entrée-sortie est positionnée sur la 4ème µseconde.

En pratique, pour un code réalisé sur CRTC 3 ou 4, il faut positionner une instruction OUT(C),r8 1 µsec avant celle qu'on aurait placée pour un CRTC 0, 1 ou 2.

Ceci est dit !

4.4.4 DISSECTION DE OUTs

Ce chapitre a pour objectif de tenter d'expliquer pourquoi une entrée sortie avec un OUT(C),r8 se produit sur le 3ème NOP pour un CRTC équipé d'un GATE ARRAY, et sur le 4ème NOP pour un ASIC qui émule un CRTC (CRTC 3 & 4), mais également pourquoi il existe une différence (que ce soit sur un CRTC ou un ASIC) entre une entrée sortie réalisée avec un OUT(C),r8 et celle réalisée avec un OUTI.

Le GATE ARRAY, au sein du CPC, est LE chef d'orchestre pour de nombreux composants. Il est cadencé à la vitesse faramineuse de 16 Mhz (ce qui lui permet d'afficher des pixels mode 2 de 0.0625 µsec). Il donne une cadence de 1 MHz au AY-3-8912 (générateur sonore) et au CRTC, et cadence le Z80A à 4 MHz.

Un des objectifs des concepteurs du GATE ARRAY était d'utiliser la capacité du Z80A à ralentir son exécution afin de récupérer la priorité d'accès à la RAM pour les adresses pointées par le CRTC.

Les instructions d'un Z80A sont constituées de périodes d'exécution (appelées Cycles M) dans lesquelles se produisent plusieurs sous-périodes (appelées Cycles T). Chaque sous-période « T » ayant une durée de 0.25 µsec (La taille de 4 pixels en mode 2).

Le point commun de toutes les instructions du Z80A est la nécessité d'accéder à la RAM pour y lire le(s) code(s) de l'instruction à exécuter (appelés « **Opcode** » pour code opération). Cette lecture, appelée « **Opcode Fetch** », est effectuée durant un premier cycle appelé M1. Chaque cycle M effectue des types d'instruction basiques :

- Lecture d'un opcode en ram (« Opcode Fetch »).
- Lecture ou écriture en ram d'un octet par le code interne du Z80A qui exécute l'opcode.
- Lecture ou écriture d'un port d'entrées/sorties (« IO Req »).
- Bus : Requête ou acquittement.
- Interruption : Requête ou acquittement.

Un cycle M est composé de plusieurs cycles T, dont certains ont la particularité de placer un signal WAIT à l'état actif afin d'indiquer à un autre circuit que le Z80A accepte d'être ralenti. Ce cycle WAIT est communément nommé Tw. C'est notamment le cas pour les instructions :

- Opcode Fetch, durant le **2ème cycle T**.
- Lecture ou Écriture mémoire, durant le **2ème cycle T**.
- IO Req, durant le **3ème cycle T**.

Lorsque le Z80A « exécute » un cycle Tw, il regarde sa ligne WAIT (en l'occurrence celle reliée au GATE ARRAY) et si elle est active, alors il va générer un autre cycle Tw. Ceci permet de bloquer le processeur indéfiniment si le circuit qui pilote la ligne WAIT le décide.

Ce blocage est possible uniquement si le processeur l'a permis dans un cycle Tw et que le GATE ARRAY faisait la demande de blocage à ce moment-là.

L'astuce des concepteurs du **GATE ARRAY** a été de **générer continuellement 3 Tw suivi d'un cycle « non Tw »**. Lorsque le Z80A, pour un cycle T qui effectue un WAIT, tombe sur un de ces cycles Tw, il va exécuter la séquence « restante » de Tw, ce qui a pour effet de « **linéariser** » les instructions sur 4 cycles T.

Autrement dit, si une instruction a débuté sur un T-Cycle « aligné » et se termine avec un nombre de T-Cycle non multiple de 4, alors la prochaine instruction sera « allongée » durant son premier cycle « M ».

Pour illustrer ces propos, le schéma A sur les pages suivantes indique comment fonctionne une instruction **OUT(#nn),A** (1 opcode #D3) qui a besoin de 11 cycles T pour s'exécuter. Cette instruction comporte un nombre impair de cycles T (donc non multiple de 4) et est immédiatement suivi d'un deuxième **OUT(#nn),A**.

Lors de l'exécution du **premier OUT(#nn),A**, aucun des signaux WAIT du Z80A ne tombe en même temps qu'un signal WAIT du GATE ARRAY. L'instruction est donc exécutée en 11 T-Cycles, soit 0.25 x 11=2.75 µsec.

Lorsque le **second OUT(#nn),A** est exécuté, le signal WAIT du Z80A lors du cycle M1 survient au même moment que le signal WAIT émis par le GATE ARRAY. Cela provoque la génération d'un 2ème T-Cycle d'attente par le Z80A (voir « **Wait Extent** » sur le schéma). Durant ce 2ème cycle WAIT, le GATE ARRAY n'envoie pas de signal WAIT, ce qui stoppe la génération de cycles WAIT par le Z80A. Ainsi, ce 2ème OUT(#nn),A se retrouve « allongé » de 0.25 µsec pour coller au motif d'accès à la mémoire défini par le GATE ARRAY. Son exécution a alors duré 3 µsec (soit 12 T-Cycles).

Et ainsi de suite. Tant que l'instruction « rectifiée » contient elle-même un nombre de T-Cycles « non aligné », la durée de l'instruction suivante sera aussi rectifiée. Ainsi dans l'exemple précédent, si un NOP (4 T-Cycles) est ajouté derrière le 2ème OUT(#nn),A, alors le cycle T2 du cycle M1 effectuera un WAIT en même temps que le GATE ARRAY, qui allongera la durée du NOP de 0.25 µsec (donc 1.25µsec au total). Un autre NOP après ce premier NOP sera aligné. En effet, le cycle T2 de son cycle M1 sera aligné sur le « cycle non wait » du GATE ARRAY, et l'instruction s'exécutera alors en 4 T-Cycles (soit 1 µsec).

Il en va de même pour toutes les instructions dont le cycle M1 s'allonge en fonction du cadencement opéré par le GATE ARRAY pour maintenir la priorité sur les accès en ram à l'adresse fournie par le CRTC tous les 4 T-Cycles.

Dans le cadre d'une opération d'écriture I/O, ce mécanisme permet de caler, par rapport au début d'une µseconde, le moment ou l'I/O débute. En l'occurrence sur **le cycle T2 du cycle M d'I/O**. Le Z80A met l'adresse I/O sur le bus de données (&BD00 pour l'écriture dans un registre du CRTC pour rappel).

Le GATE ARRAY cadence le CRTC à 1 Mhz (mais pas tout à fait au même moment que pour le AY-3-8912). Le CRTC vient périodiquement voir si le signal IORQ du Z80A est actif pour vérifier si il est concerné par l'I/O. Si c'est le cas lors d'une écriture, il peut récupérer la valeur pour sélectionner un de ses registres ou mettre à jour celui qui est sélectionné.

Selon l'instruction qui génère l'I/O, les données ne sont pas présentes immédiatement lorsque le CRTC est en état de mettre à jour ses registres, ce qui peut avoir pour conséquence de différer l'écriture des registres entre 2 instructions différentes au sein d'une même micro-seconde. C'est notamment le cas entre l'instruction OUT(C),reg et OUTI, dont la différence peut être mise en évidence avec des techniques R2.JIT ou R3.JIT, par exemple. Le schéma B sur les pages suivantes met en parallèle ces deux instructions.

À priori, les ASIC ne cadencent pas le CRTC exactement comme le GATE ARRAY (cas des « CRTC » 3 et 4). Il y a sans doute un décalage du type que celui que j'ai représenté sur le schéma C, et qui a pour conséquence d'inverser le moment de mise à jour des registres par rapport à ce qui se produit avec le GATE ARRAY.

Cela permet également d'expliquer pourquoi la mise à jour d'un registre CRTC a lieu sur la 5ème µseconde de l'instruction OUTI, quel que soit le type de CRTC, alors qu'il existe une différence de 1 µseconde lorsque la mise à jour a lieu avec l'instruction OUT(C),r8.

Diagram A : OUT(#NN),a + OUT (#nn),A + NOP + NOP

OUT(#nn),A	1																																			
	1 µSec																1 µsec																1 µSec					
GA	WAIT								WAIT				WAIT				WAIT				WAIT				WAIT								WAIT					
Z80 Cycle M	M1 - FETCH 0xD3																M2 -READ CYCLE												M3 - OUT							
	8	9	10	11	12	13	14	15	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0	1	2	3	8	9	10	11	12	13	14	15	0	1	2	3	4	5	6	7
No Tcycle	T1				T2				T3				T4				T5				T6				T7				T8				T9				T10				T11			
Z80A Oper	--				MREQ RD				RFSH				RFSH MREQ				--				MREQ RD				MREQ RD				--				WR IORQ				WR IORQ				WR IORQ			
Z80A Tw					WAIT																WAIT																WAIT							

OUT(#nn),A	2																																			
	1 µsec																1 µSec												1 µsec							
GA	WAIT				WAIT								WAIT				WAIT				WAIT				WAIT				WAIT							
Z80 Cycle M	M1 - FETCH 0xD3																M2 -READ CYCLE												M3 - OUT						
	8	9	10	11	12	13	14	15	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0	1	2	3	8	9	10	11	12	13	14	15	0	1	2	3	4	5	6	7
No Tcycle	T1				T2				Tw				T3				T4				T5				T6				T7				T8				T9				T10			
Z80A Oper	--				MREQ RD								RFSH				RFSH MREQ				--				MREQ RD				MREQ RD				--				WR IORQ				WR IORQ			
Z80A Tw					WAIT				WAIT extent																WAIT																WAIT			

NOP	3																																			
	1 µsec																1 µSec																			
GA	WAIT				WAIT								WAIT				WAIT				WAIT				WAIT											
Z80 Cycle M	M1 - FETCH 0x00																M1 - FETCH 0x00																			
	8	9	10	11	12	13	14	15	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0	1	2	3	4	5	6	7	8	9	10	11
No Tcycle	T1				T2				Tw				T3				T4				T1				T2				T3				T4			
Z80A Oper	--				MREQ RD								RFSH				RFSH MREQ				--				MREQ RD				RFSH				RFSH MREQ			
Z80A Tw					WAIT				WAIT extent																WAIT											

Diagram B : OUT(C),H vs OUTI on CRTC 0, 1, 2

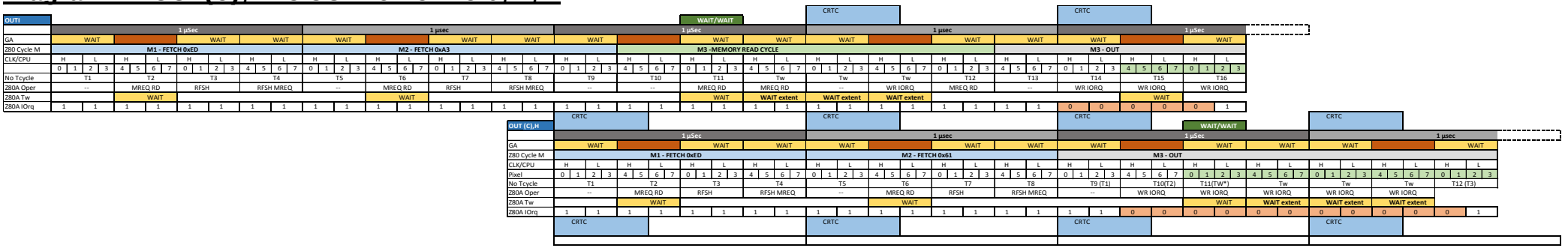
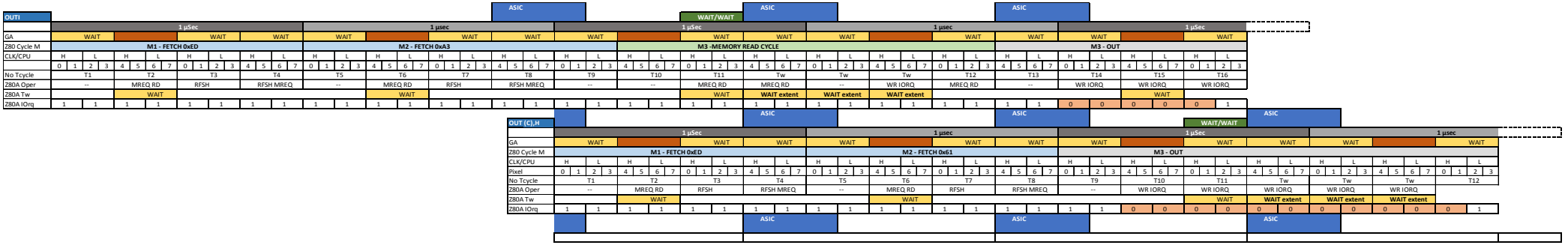


Diagram C : OUT(C),H vs OUTI on CRTC 3, 4



5 LES AUTRES CIRCUITS

5.1 ACCÈS

Les périphériques, sur CPC, ont été câblés de manière à décoder partiellement l'adresse utilisée pour effectuer l'entrée/sortie. (ai-je déjà écrit « Bienvenue sur CPC » ?).

Un périphérique est concerné par une opération d'entrée/sortie dès que quelques bits précis du bus d'adresse sont positionnés à 0 et/ou 1. Ce qui implique que si d'autres bits relatifs à d'autres périphériques sont également positionnés, alors l'opération d'entrée/sortie va également les concerner.

Il est donc possible d'envoyer une même valeur à différents périphériques **simultanément**.

L'intérêt peut sembler relatif car les valeurs communes utiles à plusieurs périphériques ne sont pas énormes. Cependant, si on mesure bien les conséquences d'envoyer des valeurs imprévues à un périphérique, cela permet de faire des « économies » de registre(s) en modifiant judicieusement la valeur des registres Z80A B et/ou C.

Dans un environnement hautement contraint par le temps machine, cela peut avoir l'intérêt d'affecter d'autres usages à ces registres.

		Register B (or A)								Register C (or n)								
Circuit	**r/w**	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**	**Select Addr**
PAL	w	**0**	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	&7F00
Gate Array	w	**0**	**1**	x	x	x	x	x	x	x	x	x	x	x	x	x	x	&7F00
CRTC Select	w	x	**0**	x	x	x	x	**0**	**0**	x	x	x	x	x	x	x	x	&BC00
CRTC Write	w	x	**0**	x	x	x	x	**0**	**1**	x	x	x	x	x	x	x	x	&BD00
CRTC Status	r	x	**0**	x	x	x	x	**1**	**0**	x	x	x	x	x	x	x	x	&BE00
CRTC Read	r	x	**0**	x	x	x	x	**1**	**1**	x	x	x	x	x	x	x	x	&BF00
ROM Select	w	x	x	**0**	x	x	x	x	x	x	x	x	x	x	x	x	x	&DF00
Printer	w	x	x	x	**0**	x	x	x	x	x	x	x	x	x	x	x	x	&EF00
PPI Port A	r/w	x	x	x	x	**0**	x	**0**	**0**	x	x	x	x	x	x	x	x	&F400
PPI Port B	r/w	x	x	x	x	**0**	x	**0**	**1**	x	x	x	x	x	x	x	x	&F500
PPI Port C	r/w	x	x	x	x	**0**	x	**1**	**0**	x	x	x	x	x	x	x	x	&F600
PPI Control	w	x	x	x	x	**0**	x	**1**	**1**	x	x	x	x	x	x	x	x	&F700
FDC Status	r	x	x	x	x	x	**0**	x	**1**	**0**	x	x	x	x	x	x	**0**	&FB7E
FDC Data	r/w	x	x	x	x	x	**0**	x	**1**	**0**	x	x	x	x	x	x	**1**	&FB7F
FDC Motor	w	x	x	x	x	x	**0**	x	**0**	**0**	x	x	x	x	x	x	x	&FA7F

Enfin, toujours dans la perspective d'environnements contraints, il peut être utile de considérer que tous les bits d'une valeur envoyée à un périphérique ne sont pas utiles.

Pour un CRTC, c'est vrai pour une donnée de mise à jour d'un registre, comme R9 tronqué à 5 bits, tout comme la valeur du numéro de registre.

Autrement dit, on peut sélectionner R9 avec la valeur 9, mais aussi &31 (00110001), et le mettre à jour avec la valeur 7, mais aussi &27 (00100111).

5.2 CPC + / GX 4000

Sur l'AMSTRAD PLUS, il est possible de communiquer directement avec le CRTC émulé, via des registres complémentaires créés spécifiquement pour gérer les fonctions complémentaires "PLUS". Ces fonctions permettent par exemple de faire des masquages de données, d'effectuer des décalages fins, de définir des lignes et adresses de rupture, ou même de caler une interruption raster ou "dma" (non sans bug toutefois).

L'objectif actuel de ce document n'est pas (encore) d'analyser l'interaction entre ces registres et l'émulation faite du CRTC. L'accès à ces registres spécifiques ne passe pas par le système d'entrée/sortie du Z80A décrit précédemment.

L'AMSTRAD PLUS dispose d'une page de registres dit "mappés" sur une adresse.
Chaque registre dispose donc de sa propre adresse.
Le Z80A accède à ces registres avec de simples écritures (ou lecture) à une adresse donnée.

Il est donc possible d'accéder à certaines fonctions (par exemple la modification d'une couleur) de deux manières différentes. Les informations indiquées pour le CRTC 3 dans cette version du document concernent uniquement des I/O réalisées via les instructions OUT du Z80A.

À noter cependant que la page de ces registres, située entre &4000 et &7FFF, est accessible via une fonction jusqu'alors inexploitée du GATE ARRAY.

Cette fonction est cependant elle-même conditionnée à l'utilisation d'une séquence de « déverrouillage » de 17 octets qui doit être envoyée sur le port de sélection (&BC00) du CRTC.

255, 0, 255, 119, 179, 81, 168, 212, 98, 57, 156, 70, 43, 21, 138, 205, 238

Remarque :
Il ne sert à rien d'envoyer cette séquence sur un CRTC 4.
Ils n'ont pas utilisé la même séquence.
Je ne dirais rien de plus sans la présence de mon avocat...

6 CONSTRUCTION D'UN FRAME

6.1 LOGIQUE GENERALE

Les algorithmes suivants décrivent la logique générale de gestion d'affichage avec un CRTC « idéal ». Nous le verrons plus loin, cette logique est parfois très différente selon les situations.

6.1.1 COMPTAGE DES CARACTERES

C0 est incrémenté
Lorsque C0 atteint R0
    C0 passe à 0
    C9 est incrémenté
    Lorsque C9 atteint R9
        C9 passe à 0
        C4 est incrémenté
        Lorsque C4 atteint R4
            Lorsque C5 atteint R5
                C4 passe à 0, C5 passe à 0
                MA est rechargé à partir de R12/R13
            Sinon C5 est incrémenté

6.1.2 SYNCHRONISATIONS

Lorsque C0 atteint R2
    Hsync débute, C3h=0
    C3h est incrémenté si R3h>0
    Lorsque C3h=R3h
        Fin de Hsync

Lorsque C4 atteint R7
    Vsync débute, C3v=0
    C3v est incrémenté
    Lorsque C3v atteint R3v (ou 16)
        Fin de Vsync

6.1.3 AFFICHAGE DES CARACTERES

Lorsque C0 passe à 0
    L'affichage des caractères est activé
Lorsque C0 atteint R1
    L'affichage des caractères est désactivé

Lorsque C4 passe à 0
    L'affichage des lignes de caractères est activé
Lorsque C4 atteint R6
    L'affichage des lignes de caractères est désactivé

6.1.4 POINTEUR VIDEO

À chaque µsec, VMA est incrémenté de 2 tant que l'affichage est actif et C9 intervient dans l'adresse

```
Si C0=R0      alors C0=0                                          vrai pour R0=0
              Si C9=R9      Alors C9=0                            vrai pour R9=0
                            Si C4=R4      alors
                                          Si R5=0       alors     vrai pour R5=0
                                                C4=0              vrai pour R4=0
                                                MA=R12/R13
                                          Sinon C4=C4+1 (crtc 0, 1, 2) et  Gestion C5

                                    Sinon C4=C4+1
                        Sinon C9=C9+1
          sinon C0=C0+1
```

6.1.5 REPRÉSENTATION SCHÉMATIQUE

Les schémas suivants décrivent globalement la construction d'un frame à partir des différents registres, sans que les registres soient modifiés durant le balayage.

Définition Ecran	Horiz. Reg				Vertical Reg					Vid Ptr		Special
	R0	R1	R2	R3	R4	R5	R6	R7	R9	R12/R13		R8
	63	40	46	14	35	24	25	30	7	&30	0	0

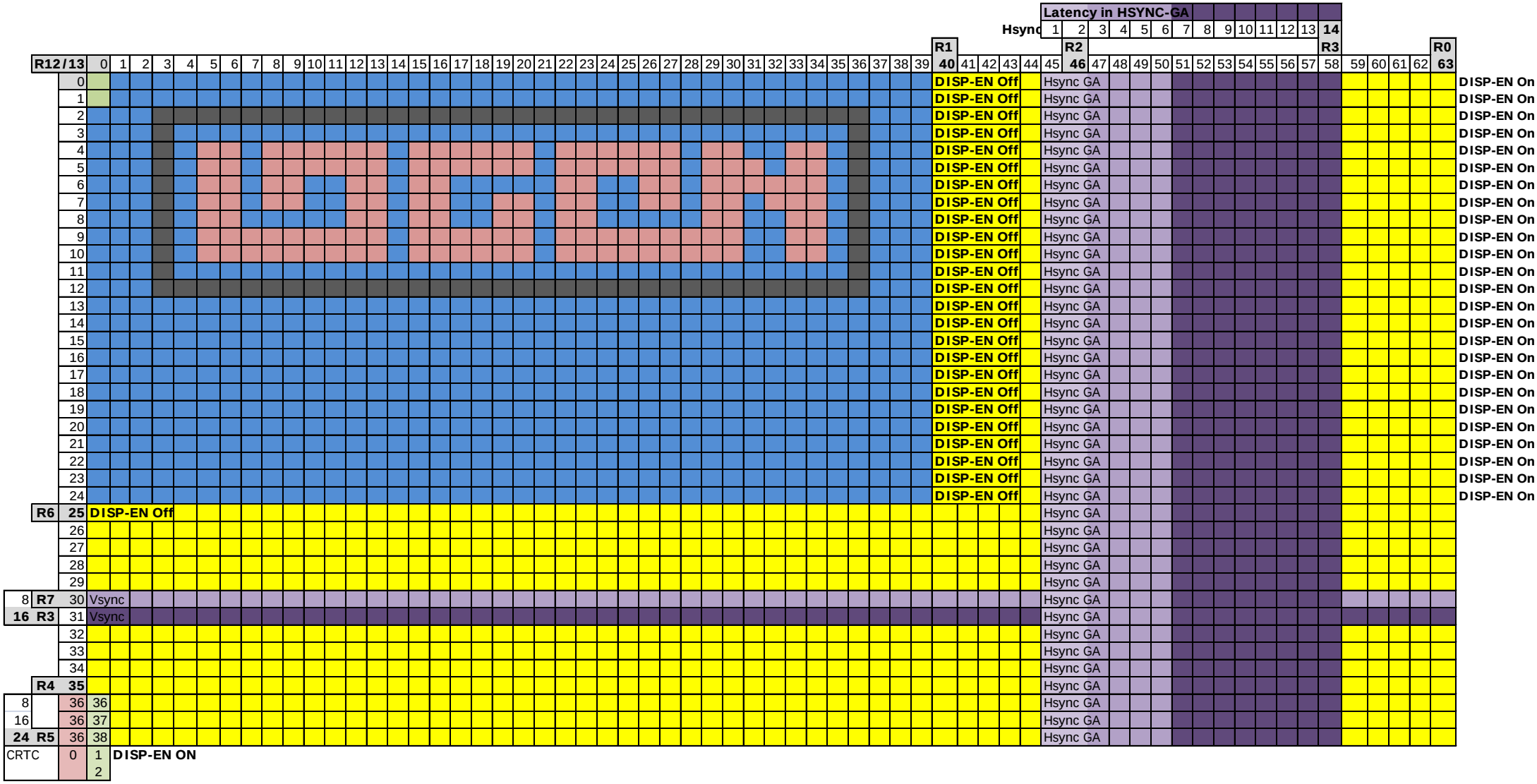
(R0+1) x [((R4+1) x (R9+1)) + R5]
64 x ((36 x 8)+24) = 19968 µs

Description des 16 premières lignes en vert indiquées sur les schémas pages suivantes :

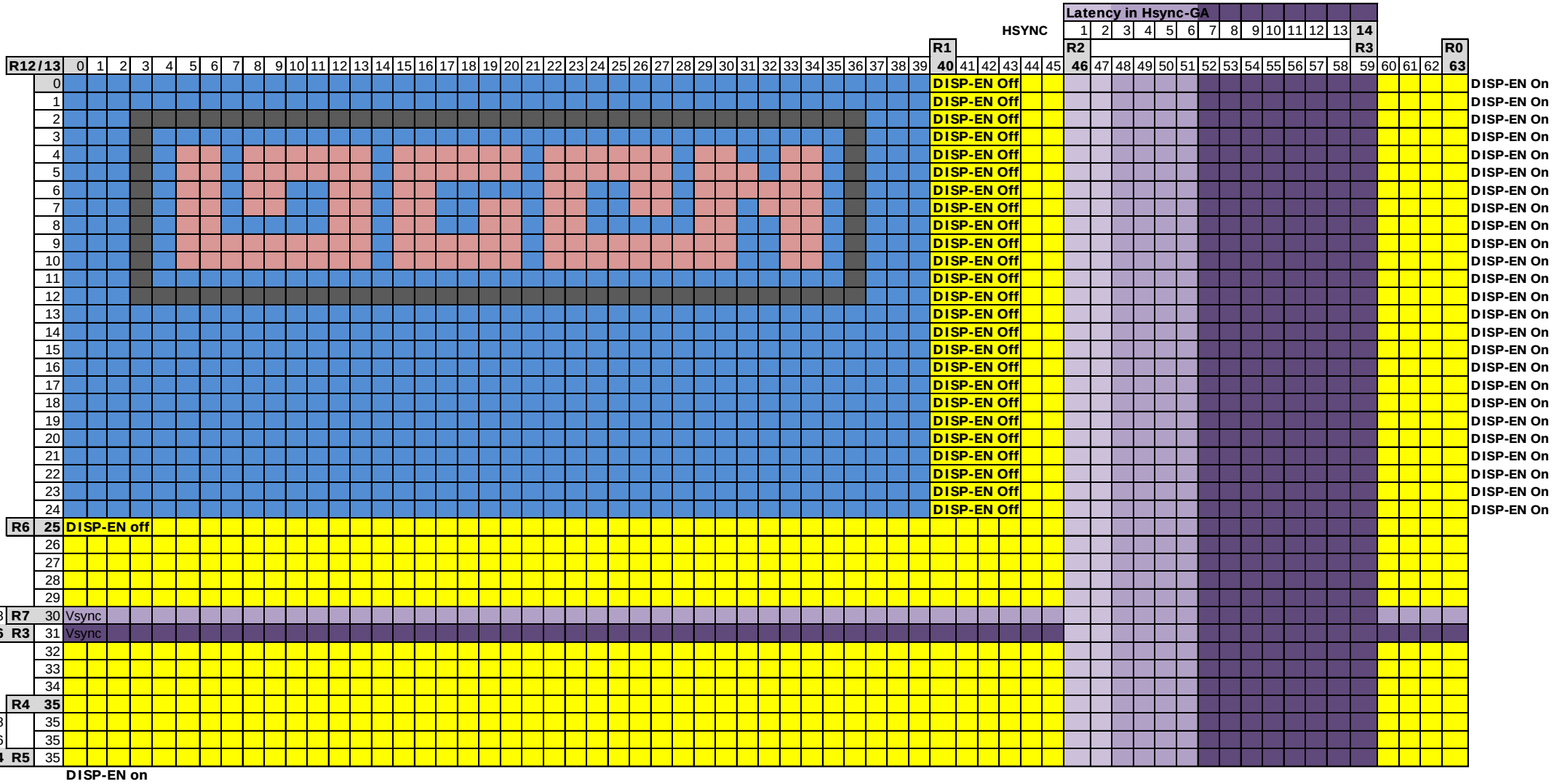
Video Pointer																																		
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0				First byte								Second byte								
R12		C9 (*)			R12/13										0	Hex			7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	
5	4	2	1	0	9	8	7	6	5	4	3	2	1	0	0	Addr	C9																		C4
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C000	0																		0
1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	C800	1																		0
1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	D000	2																		0
1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	D800	3																		0
1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	E000	4																		0
1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	E800	5																		0
1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	F000	6																		0
1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	F800	R9 7																		0
1	1	0	0	0	0	0	0	0	1	0	1	0	0	0	0	C050	0																		1
1	1	0	0	1	0	0	0	0	1	0	1	0	0	0	0	C850	1																		1
1	1	0	1	0	0	0	0	0	1	0	1	0	0	0	0	D050	2																		1
1	1	0	1	1	0	0	0	0	1	0	1	0	0	0	0	D850	3																		1
1	1	1	0	0	0	0	0	0	1	0	1	0	0	0	0	E050	4																		1
1	1	1	0	1	0	0	0	0	1	0	1	0	0	0	0	E850	5																		1
1	1	1	1	0	0	0	0	0	1	0	1	0	0	0	0	F050	6																		1
1	1	1	1	1	0	0	0	0	1	0	1	0	0	0	0	F850	R9 7																		1

(*) C5 on CRTC 3, 4 in vertical adjustment period (R5)

CRTC 0, 1, 2


Latency in HSYNC-GA
Hsync
R12 / 13
R1
R2
R3
R0
R6
R7
R4
R5
CRTC
DISP-EN Off
DISP-EN On
Hsync GA
Vsync
DISP-EN ON

CRTC 3, 4


HSYNC
Latency in Hsync-GA
R12 / 13
R1
R2
R3
R0
R6
R7
R4
R5
DISP-EN Off
DISP-EN On
DISP-EN off
DISP-EN on
Vsync

7 SYNCHRONISATION

7.1 PRINCIPES

Le CRTC est chargé de déterminer les adresses mémoires à afficher, qu'il communique au GATE ARRAY, qui lit les données et les gère selon ses paramètres, pour générer des pixels plus ou moins larges et colorés.

Le CRTC envoie aussi au GATE ARRAY les signaux VSYNC et HSYNC, que ce dernier recombine pour envoyer un signal composite au moniteur.

- Lorsque le GATE ARRAY reçoit un signal HSYNC, il y a 2 µs de latence avant que la "vraie" synchro horizontale du moniteur débute.
- Lorsque le GATE ARRAY reçoit un signal VSYNC, il y a 2 lignes de latence avant que la "vraie" synchro verticale du moniteur débute.

Sur un sujet qui traite des différences entre des modèles de circuits à l'échelle de la microseconde, il est important de spatialiser les instructions par rapport à ce qui est affiché.

Plusieurs difficultés sont liées à cette démarche :

- Situer une instruction Z80A par rapport au fonctionnement interne du CRTC, dans la mesure où des délais peuvent exister entre la mise à jour des registres internes et le moment où les caractères sont affichés.
- Déterminer quand, durant l'instruction d'entrée sortie, la modification d'un registre CRTC est effective et prise en compte.

Le même type de question existe pour les instructions Z80A accédant directement au GATE ARRAY, ou lors de la mise à jour de la RAM lue par le GATE ARRAY.

Pour « synchroniser » les instructions, ce document se base sur un point de référence unique sur lequel sont alignées les instructions en Z80A affectant la vidéo.

Ce point de référence est le moment où on considère que C0=0 au sens CRTC.

Il existe un différé d'affichage entre le moment ou le CRTC fournit un pointeur vidéo, et le moment où le GATE ARRAY affiche le caractère 16 bits correspondant.
Ce différé est de 1 µsec.

Ce décalage temporel d'affichage du GATE ARRAY par rapport au CRTC ne serait pas gênant si l'intégralité de ce qui est envoyé par le CRTC au GATE ARRAY était **toujours** retardé de 1 µsec.

Mais ce n'est pas toujours le cas, en particulier pour la gestion du signal HSYNC pour les machines équipées des CRTC 0, 1 et 2.

Lorsque ce document traite d'instructions Z80A, et que leur synchronisation a une importance au regard des différences de délai, le compteur C0 est présenté sur 2 "time-line".
Celle au regard de C0 à partir du point de référence CRTC et celle au regard de l'affichage (pixels, ...) par le GATE ARRAY.

Il sera donc question de "**C0 from Vsync**" (ou **C0vs**) et/ou "C0 from GA" lorsque l'affectation des registres du CRTC est déterminante par rapport à l'affichage des caractères par le GATE ARRAY.

Le CRTC communique avec le Z80A de deux manières différentes (en dehors des registres en lecture ou du registre de statut du CRTC 1) :

- Soit via la broche VSYNC du CRTC qui est reliée directement au bit 0 du port B du PPI 8255.
- Soit via les interruptions du Z80A produites par le GATE ARRAY à partir des signaux HSYNC du CRTC.

7.2 SYNCHRONISATION VSYNC

Le CRTC génère un signal sur sa broche VSYNC lorsque C4 atteint R7.
C'est un indicateur fiable pour baser ensuite toute autre méthode de synchronisation.

Le bit 0 du port B du PPI (adresse d'entrée sortie "standard" &F5) passe à 1 dès que le signal VSYNC est produit par le CRTC.

En général, les programmes font des "boucles" pour attendre que ce bit passe à 1, ce qui entraine une marge d'erreur liée à la durée des instructions du code d'attente.

Cette marge existe même si le code de « boucles » est "aligné" grâce à une interruption générée à partir d'une instruction HALT située avant la boucle d'attente (la marge est juste "stable").

Il est cependant parfaitement possible d'écrire un code capable de caler du code Z80A sur la microseconde qui correspond à C4=R7 et C0=C9=0.

Toutes les données indiquées dans ce document utiliseront ce point de référence **C0vs**.
Il est bien sûr possible d'utiliser le système d'interruption comme nouveau point de référence.

Le chapitre 26.6.6 sur les interruptions fait une synthèse selon les différents CRTC.

Il ne faut cependant pas perdre de vue que les interruptions sont dépendantes des différences avec les couples CRTC/GATE ARRAY des différentes machines.

Le principe du code de synchronisation **C0vs** est de placer une attente de VSYNC dans une période où l'indicateur au niveau du PPI n'est pas encore positionné à 1.

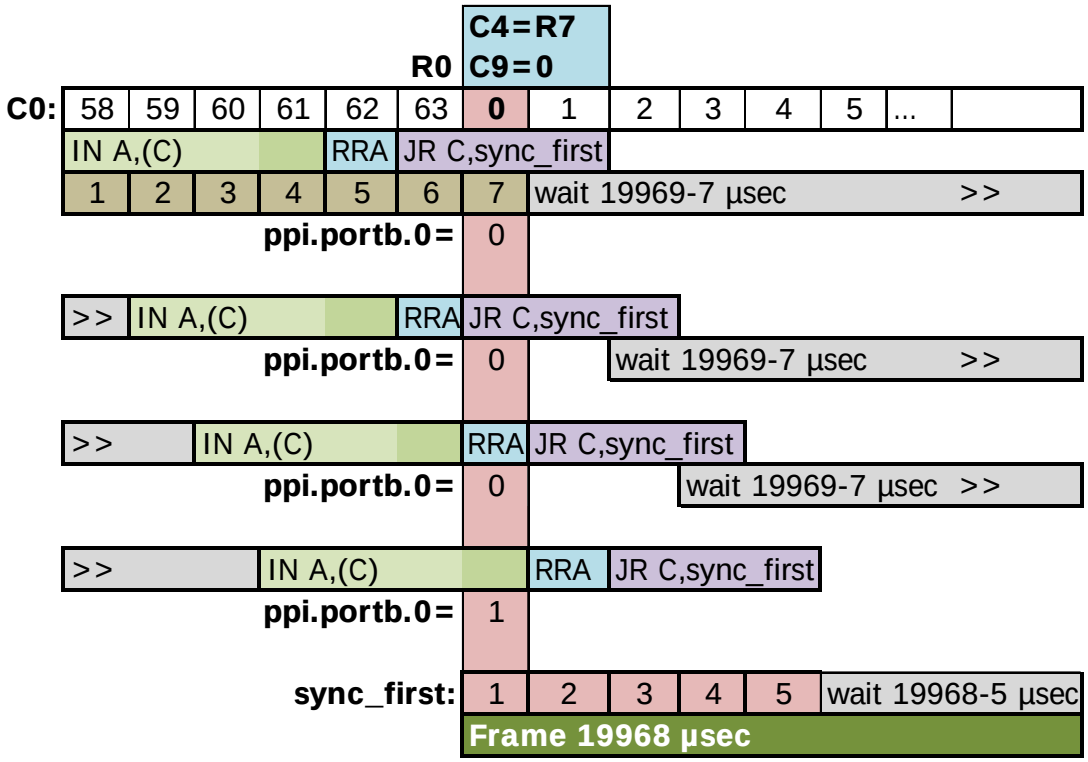
Il s'agit ensuite de faire dériver ce code microseconde par microseconde à chaque frame, grâce à une période de **19969** microsecondes. Ce balayage est plus long de 1 microseconde que celui produit par le CRTC (avec une programmation adéquate et standard de 312 lignes de 64µsec).

Ceci peut être vu comme une "dérive" du code microseconde par microseconde.

L'instruction d'entrée sortie, qui lit le port B du PPI, est l'instruction IN A,(C) qui s'exécute en 4 µsec.

C'est sur la 4ème µsec de l'instruction IN A,(C) que A est mis à jour lorsque le bit 0 du port B passe à 1. Le point de référence **C0vs** est calculé de cette manière et toutes les informations indiquées dans ce document sont corrélées à cette mesure, qui est un point de référence unique.

Description schématique du principe :



Code de synchronisation :

```
;=============================================================================================
;
;       Synchronisation Vsync
;       A l'issue du call de synchronisation, la 1ere usec est celle sur laquelle le CRTC a positionné le signal Vsync
;       Principe
;               - attendre le signal vsync avec la marge d'erreur
;               - attendre que le signal vsync se termine
;               - faire deriver le test de vsync avec une boucle durant 19968+1 us
;               - des que la vsync est detectee par le test, c'est la detection au plus tot, a laquelle il faut soustraire la duree du test
;
;=============================================================================================
sync_vbl
                di
                ld b,#f5            ; Attendre Etat Vsync =1
                ld hl,19968-21      ; Compteur de nop (moins les marges et la gestion de l'attente)
                ld de,-11
sync_wvblon1                        ; Ici on attend le debut de la periode Vsync (ou on attend pas si on y etait deja)
                in a,(c)            ;
                rra                 ;
                jr nc,sync_wvblon1
sync_wvbloff1                       ; Flag Vsync CRT passe a 1 (ou etait deja a 1)
                in a,(c)            ; Attendre que le flag repasse a 0 (Fin de Vsync)
                rra
                jr c,sync_wvbloff1
sync_wvblon2                        ; On est certain maintenant que le signal Vsync n'etait pas deja en cours
                in a,(c)            ;
                rra                 ; marge1 de 7us
                jr nc,sync_wvblon2
sync_wvbloff2                       ; Attendre que le signal Vsync repasse a 0 en comptant le temps ecoule
                add hl,de           ; 3     On nop2         On nop3
                in a,(c)            ; 4     2               1
                rra                 ; 1     1               1
                jr c,sync_wvbloff2  ; 3/2   2               3       (bcl)+3+4+1+2=15 / marge 15-5=10
                ex de,hl            ; 1
                call wait_usec      ; 5 >> 6 + 10(marge2)
                ;
                ; Zone de  derive pour attendre de nouveau la premiere manifestation du flag
                ; le in a,(c) va "descendre" nop par nop (frame par frame) jusqu'a ce que le in recupere le flag actif
sync_derive_bcl
                ld b,#f5                ; 2
```

```
                in a,(c)            ; 4 usec. 0.1.2.[3] (+1)
                rra                 ; 1 usec (+1)
                jr c,sync_first     ; 2/3 (+3)
                ld de,19969-20      ; 3
                call wait_usec      ; 5+(19969-20)
                jr sync_derive_bcl  ; 3 >> 20
sync_first                          ; 6 Le flag a ete détecté au plus tôt, et ce depuis 5 usec (1+1+3)
                ld de,19968-11      ; 3
                jp wait_usec        ; 3 >> 11 >> de=19968-11



;================================================================================
; wait "de" usec
; 40+(((de/8)-5) x 8)+(de and 7) nop
; nb - le call de la fonction n'est pas compte
;================================================================================
wait_usec:
                ld hl,sync_adjust   ; 3
                ld b,0              ; 2
                ld a,e              ; 1
                and %111                    ; 2>8
                ld c,a              ; 1
                sbc hl,bc           ; 4
                srl d               ; 2
                rr e                ; 2>17
                srl d               ; 2
                rr e                ; 2
                srl d               ; 2
                rr e                ; 2>25
                dec de              ; 2>27 8
                dec de              ; 2>29 16
                dec de              ; 2>31 24
                dec de              ; 2>33 32
                dec de              ; 2>35 40 *
                nop                 ; 1>36
wait_usec_01
                dec de              ; 2 -
                ld a,d              ; 1 -
                or e                ; 1 -
                nop                 ; 1 -
                jp nz,wait_usec_01  ; 3 - v=(8 x DE)
                jp (hl)             ; 1>37
                nop                 ; 1 * v=0--7
                nop                 ; 1
                nop                 ; 1
                nop                 ; 1
                nop                 ; 1
                nop                 ; 1
                nop                 ; 1
sync_adjust
                ret                 ; 3>40 *
```

7.3 FAKE VSYNC

Le signal VSYNC est au sein d'une relation tripartite, entre le CRTC, le GATE ARRAY et le PPI.
Le CRTC communique l'état de sa broche VSYNC au PPI et au GATE ARRAY.

Le PPI 8255 est un circuit générique programmable d'interfaçage de périphériques.
AMSTRAD a utilisé 3 versions différentes de ce composant :

- NEC D8255AC-5
- NEC D8255AC-2
- TOSHIBA TMP8255AP-5

Le dernier chiffre indiqué est la fréquence maximum du circuit.
À ma connaissance il n'existe pas de test pertinent pour différencier ces modèles.

Il est possible de faire fonctionner les broches de communication de ce circuit en entrée ou en sortie selon une programmation adéquate.

Le port B du PPI est conçu sur CPC pour fonctionner en entrée, et donc recevoir des informations des périphériques auquel il est connecté. Il est possible de reprogrammer ce port en sortie, en mettant à 0 le bit 1 du registre de contrôle du PPI situé à l'adresse d'I/O #F700. Une écriture sur le port B (#F500) d'une valeur avec le bit 0=1 va donc mettre la broche 25 du circuit à l'état haut. Cela signifie que le PPI va envoyer son signal vers le CRTC, qui par voie de fait, va le renvoyer vers le GATE ARRAY, puisqu'il y est également connecté.

Cependant, j'ai constaté, comme Kevin Thacker (ArnoldEmu) avant moi, que cette FAKE VSYNC ne fonctionne pas sur certains CPC. Sur d'autres CPC, le résultat est incorrect.

Sur un CRTC 2, programmé avec R2=50 et R3=14, la VSYNC n'est en principe plus générée.

Nous verrons plus loin qu'un des problèmes du CRTC 2 est qu'une VSYNC FANTÔME est générée si la condition VSYNC a lieu durant la HSYNC. Il devrait donc en théorie être possible de faire ce travail à la place du CRTC en plaçant le bit 0 du port B à 1 au bon endroit et pendant la bonne durée.

Cependant, si on active pendant 1024 µsec une VSYNC PPI en lieu et place de celle qu'aurait généré le CRTC, le signal produit par le PPI n'est à priori pas assez fort pour contrer le signal bas généré par une VSYNC FANTÔME. Mais lorsque le CRTC refait passer sa broche à l'état bas (alors qu'elle y est déjà), alors la VSYNC du PPI est alors prise en compte, et l'image se retrouve stabilisée 16 lignes plus haut que prévu. Un électronicien m'aiderait certainement à y voir plus clair.

Il reste néanmoins possible de contourner plus simplement la VSYNC FANTÔME du CRTC 2, en évitant que la condition de démarrage ait lieu durant la HSYNC et en mettant à jour R7 avec C4 juste après la fin de la HSYNC (voir chapitre 15.2, page 133).

ATTENTION : Je ne connais pas vraiment les conséquences « techniques » potentielles relatives à l'envoi d'un signal dans le sens opposé à celui prévu. Ceci étant dit, le risque parait assez faible dans la mesure où, depuis plusieurs années, de nombreux CPC contiennent plusieurs CRTC soudés l'un sur l'autre dans une orgie d'étain. Le ou les CRTC passifs prenant alors les signaux du CRTC actif. Dans le pire des cas, un CRTC vaut (encore) moins de 10€.

Un peu d'histoire

Ce n'est pas la première fois qu'un port prévu en entrée est utilisé en sortie sur le CPC. Les plus anciens se souviendront sans doute du kit de téléchargement qui permettait de récupérer des programmes (payants) via une connexion téléphonique reliée au minitel (je ne sais pas si ce système a été utilisé avec Prestel en U.K.). Le câble d'un de ces kits était connecté à partir de la prise minitel sur le CPC avec sa prise JOYSTICK, et afin de pouvoir gérer un échange bidirectionnel, le port clavier était alors placé en sortie. Les lignes claviers sont lues à travers le port A du PPI, via le port A du AY-3-8912 (générateur sonore). Même si pour la très grande majorité des utilisateurs, cela n'a eu aucune conséquence, j'ai été témoin de plusieurs CPC dont le AY-3-8912 avait « perdu définitivement » une partie des bits de certains registres du AY. De mémoire, il me semble que ces CPC avait des tonalités particulièrement aigues, si il s'agit de déterminer quels bits étaient « ravagés ».

8 AFFICHAGE, Z80A & GATE ARRAY

Les schémas suivants décrivent, par rapport à quelques instructions de mise à jour de la ram vidéo par le Z80A, quand cette mise à jour est prise en compte par le GATE ARRAY/ASIC.

Les instructions sont localisées par rapport au point de référence C0vs du CRTC.

La lecture en ram par le GATE ARRAY/ASIC pour les données modifiées par le Z80A est identique pour tous les CPC.

8.1 INSTRUCTION LD(HL),reg8 (2 µsec)

HL=0000	reg8=#FF																			
C0vs	00		01		02		03		04		05		06		07		08		09	
Video Ptr	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Z80A Inst	LD (HL),reg8				NOP		NOP		NOP		NOP		NOP		NOP		**LD (HL),#00**			19968-10 µsec
Displayed :	**00**																			

HL=0001	reg8=#FF																			
C0vs	00		01		02		03		04		05		06		07		08		09	
Video Ptr	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Z80A Inst	LD (HL),reg8				NOP		NOP		NOP		NOP		NOP		NOP		**LD (HL),#00**			19968-10 µsec
Displayed :			**00**																	

HL=0002	**reg8=#FF**																			
C0vs	00		01		02		03		04		05		06		07		08		09	
Video Ptr	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Z80A Inst	**LD (HL),reg8**				NOP		NOP		NOP		NOP		NOP		NOP		LD (HL),#00			19968-10 µsec
Displayed :			**FF**																	

HL=0003	**reg8=#FF**																			
C0vs	00		01		02		03		04		05		06		07		08		09	
Video Ptr	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Z80A Inst	**LD (HL),reg8**				NOP		NOP		NOP		NOP		NOP		NOP		LD (HL),#00			19968-10 µsec
Displayed :					**FF**															

8.2 INSTRUCTION LD (aaaa),HL (5 µsec)

aaaa=0004	H=#FF, L=#55																**H'=0, L'=0**															
C0vs	00		01		02		03		04		05		06		07		08		09		10		11		12		13		14		15		16	
Video Ptr	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33
Z80A Inst	LD (aaaa),HL										NOP		NOP		NOP		NOP		NOP		NOP		EXX		**Ld (aaaa),HL**									19968-17 µsec
Displayed :					**00**	**00**																												

aaaa=0005	H=#FF, L=#55																**H'=0, L'=0**															
C0vs	00		01		02		03		04		05		06		07		08		09		10		11		12		13		14		15		16	
Video Ptr	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33
Z80A Inst	LD (aaaa),HL										NOP		NOP		NOP		NOP		NOP		NOP		EXX		**Ld (aaaa),HL**									19968-17 µsec
Displayed :							**00**	**00**																										

aaaa=0006	H=#FF, **L=#55**																**H'=0**, L'=0															
C0vs	00		01		02		03		04		05		06		07		08		09		10		11		12		13		14		15		16	
Video Ptr	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33
Z80A Inst	**LD (aaaa),HL**										NOP		NOP		NOP		NOP		NOP		NOP		EXX		**Ld (aaaa),HL**									19968-17 µsec
Displayed :							**55**	**00**																										
							L																											

aaaa=0007	H=#FF, L=#55																H'=0, L'=0															
C0vs	00		01		02		03		04		05		06		07		08		09		10		11		12		13		14		15		16	
Video Ptr	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33
Z80A Inst	**LD (aaaa),HL**										NOP		NOP		NOP		NOP		NOP		NOP		EXX		Ld (aaaa),HL									19968-17 µsec
Displayed :									**55**	**FF**																								
									L	**H**																								

8.3 INSTRUCTION PUSH reg16 (4 µsec)

SP=0004	D=#FF, E=#55																								**H=0, L=0, aaaa=0003**									
C0vs	00		01		02		03		04		05		06		07		08		09		10		11		12		13		14		15		16		
Video Ptr	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33
Z80A Inst	PUSH DE								NOP		NOP		NOP		NOP		NOP		NOP		NOP		NOP		**Ld (aaaa),HL**										19968-17 µsec
Displayed :			**00**	**00**																														

SP=0005	D=#FF, E=#55																								**H=0, L=0, aaaa=0004**									
C0vs	00		01		02		03		04		05		06		07		08		09		10		11		12		13		14		15		16		
Video Ptr	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33
Z80A Inst	PUSH DE								NOP		NOP		NOP		NOP		NOP		NOP		NOP		NOP		**Ld (aaaa),HL**										19968-17 µsec
Displayed :					**00**	**FF**																												
						D																												

SP=0006	D=#FF, E=#55																								**H=0, L=0, aaaa=0005**									
C0vs	00		01		02		03		04		05		06		07		08		09		10		11		12		13		14		15		16		
Video Ptr	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33
Z80A Inst	PUSH DE								NOP		NOP		NOP		NOP		NOP		NOP		NOP		NOP		**Ld (aaaa),HL**										19968-17 µsec
Displayed :					**00**	**FF**																												
						D																												

SP=0007	D=#FF, E=#55																								**H=0, L=0, aaaa=0006**									
C0vs	00		01		02		03		04		05		06		07		08		09		10		11		12		13		14		15		16		
Video Ptr	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33
Z80A Inst	PUSH DE								NOP		NOP		NOP		NOP		NOP		NOP		NOP		NOP		**Ld (aaaa),HL**										19968-17 µsec
Displayed :							**00**	**FF**																										
								D																										

SP=0008	D=#FF, E=#55																								**H=0, L=0, aaaa=0007**									
C0vs	00		01		02		03		04		05		06		07		08		09		10		11		12		13		14		15		16		
Video Ptr	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33
Z80A Inst	PUSH DE								NOP		NOP		NOP		NOP		NOP		NOP		NOP		NOP		**Ld (aaaa),HL**										19968-17 µsec
Displayed :									**55**	**FF**																								
									E	**D**																								

9 GATE ARRAY

Sur CPC, il existe 5 modèles de GATE ARRAY identifiés, chargés, entre autres choses, d'afficher les pixels selon le mode graphique et la couleur définie pour chaque encre.

- Les modèles 40007 (version 1 : matrice Ferranti ULA20RA023, version 2 : matrice Ferranti ULA16RA023) et 40008 (version 2) sont compatibles au niveau de leurs broches. On peut noter qu'il existe différents marquages du 40007, déclinés aussi sous 40007-4 ou 40007-4X. Ces modèles sont souvent montés avec un dissipateur tartiné de pâte thermique sur les cartes mères.
- Le modèle 40010 existe sous 2 déclinaisons (1ère version 37AA (matrice LSI HSG3170) et 2ème version 36AA (matrice LSI HSG3130, 23% plus petite)).
- L'asic 40226 est utilisé sur les CPC Low Cost (CRTC 4).
- L'asic 40489 (CRTC 3) est utilisé sur les CPC+ et la GX 4000.

AMSTRAD a créé un florilège impressionnant de cartes mères différentes. Cela implique qu'on peut retrouver ces 3 composants sur certains modèles de 464 et de 6128. Je n'ai cependant pas vu de carte mère de 664 capable de supporter un 40007/40008. Enfin, le composant est toujours monté sur support.

Modèle	Nombre emplacements	40007	40008	40010
464	1	☑		
464	2	☑	☑	☑
664	1			☑
6128	1	☑		
6128	1		☑	☑
6128	2	☑	☑	☑

Remarque : La partie 2 du chapitre 3.2.2, page 10 des classeurs WEKA contient une erreur. Les broches du 40007/40008 ne sont pas compatibles avec celles du 40010, contrairement à ce qui est indiqué.

Remarque : Les modèles de 6128 avec un seul emplacement équipé d'un 40007 sont assez rares (cartes mères MC0057A sorties mi 88). Le GATE ARRAY est recouvert d'un dissipateur thermique.

De manière générale, il existe une grande quantité de 464 équipés de divers modèles.
Les 664 sont principalement équipés de 40008 et 40010.
Les 6128 sont très majoritairement équipés de 40010 version 36AA.

Contrairement à ce qui est communément admis, il y a bien des différences identifiables entre ces composants. Le 40007/40008 semble être en avance de 1/16 Mhz sur le 40010 lorsqu'il traite les bits de l'octet récupéré en VRAM.

9.1PIXELISATION

Le GATE ARRAY/ASIC est chargé de convertir la mémoire lue par le CRTC en pixels.

Il effectue cette opération en tenant compte de deux paramètres : le **mode graphique** et la **couleur associée** au pixel à afficher.

Ce document n'est pas consacré au GATE ARRAY en détail, mais il faut retenir que le CPC dispose de 4 modes graphiques, dont 3 « officiels ».

Le mode graphique détermine la taille horizontale d'un pixel (en fonction du nombre de couleurs que ce pixel peut prendre).

Un pixel peut être codé sur 1, 2 ou 4 bits.

Un pixel occupant 1 bit est la résolution la plus fine que les GATE ARRAY peuvent produire.

Mode	VRAM BYTE							
0	A0	B0	A2	B2	A1	B1	A3	B3
1	A0	B0	C0	D0	A1	B1	C1	D1
2	A0	B0	C0	D0	E0	F0	G0	H0
3	A0	B0	x	x	A1	B1	x	x
Bit Nb:	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**

Displayed Pixels							
A3	A2	A1	A0	B3	B2	B1	B0
A1	A0	B1	B0	C1	C0	D1	D0
A0	B0	C0	D0	E0	F0	G0	H0
0	0	A1	A0	0	0	B1	B0
1	2	3	4	5	6	7	8

Definition
2 pixels (16 colors)
4 pixels (4 colors)
8 pixels (2 colors)
2 pixels (4 colors)

A 17th color, stored in the GATE ARRAY, is displayed when BORDER is activated

Cette codification du pixel représente un index dans une table contenant des couleurs codées sur 5 bits

Color Index				INKR			COLOR				
				7	6	5	4	3	2	1	0
0	0	0	0	0	1	0	C1	C1	C1	C1	C1
0	0	0	1	0	1	0	C2	C2	C2	C2	C2
0	0	1	0	0	1	0	C3	C3	C3	C3	C3
0	0	1	1	0	1	0	C4	C4	C4	C4	C4
0	1	0	0	0	1	0	C5	C5	C5	C5	C5
0	1	0	1	0	1	0	C6	C6	C6	C6	C6
0	1	1	0	0	1	0	C7	C7	C7	C7	C7
0	1	1	1	0	1	0	C8	C8	C8	C8	C8
1	0	0	0	0	1	0	C9	C9	C9	C9	C9
1	0	0	1	0	1	0	C10	C10	C10	C10	C10
1	0	1	0	0	1	0	C11	C11	C11	C11	C11
1	0	1	1	0	1	0	C12	C12	C12	C12	C12
1	1	0	0	0	1	0	C13	C13	C13	C13	C13
1	1	0	1	0	1	0	C14	C14	C14	C14	C14
1	1	1	0	0	1	0	C15	C15	C15	C15	C15
1	1	1	1	0	1	0	C16	C16	C16	C16	C16

Lorsque le GATE ARRAY affiche ses pixels, **il est trop "rapide" pour ceux du MODE 2**.

Sur les GATE ARRAY 40007, 40008, 40010 et ASIC 40226 (CRTC 4), les pixels en mode 2 sont affichés un pixel plus tôt que pour les autres modes graphiques.

Ils sont affichés en "avance" de 1/16 µsec (0.0625 µsec).

Autrement dit, le BORDER "cesse" 1 pixel plus tôt sur une ligne en mode 2, et débute 1 pixel plus tôt lorsque C0=R1 si on n'a pas changé de mode graphique en cours de ligne.

Étant donné qu'il est possible de changer de mode graphique entre chaque ligne et/ou durant une ligne, il faut en tenir compte s'il est nécessaire d'aligner verticalement des pixels affichés dans des modes graphiques différents.

L'ASIC 40489 du CPC+ n'est pas concerné par ce décalage.
Quel que soit le mode graphique sur cette machine, les pixels sont alignés.

9.2 INKERISATION

Lorsque la table des couleurs (couleur d'une encre et couleur du border) est mise à jour, la prise en compte du changement de couleur n'est pas exactement identique selon les CRTC.

Cependant, le moment où la mise à jour a lieu est identique quel que soit le mode.

9.2.1 BORDER ET MODE 2

Sur certains CPC, le changement de couleur du BORDER lorsque le CPC est en **mode graphique 2** peut avoir une incidence visuelle. En effet, deux pixels d'une autre couleur peuvent apparaitre au début et à la fin de la micro seconde où la couleur change.

Ce problème est lié à un défaut d'alimentation qui affecte le GATE ARRAY ou les résistances et condensateurs qui l'entourent pour générer des couleurs, mais qu'on retrouve néanmoins sur un nombre significatif de CPC pour que je l'évoque ici. Il ne dépend ni du CRTC, ni du modèle du GA, le phénomène ayant été constaté sur un GA 40010 (28818/36AA) mais pas sur un autre de même référence. Enfin, le tripotage du connecteur d'alimentation pour établir un contact d'alimentation plus franc fait mécaniquement disparaitre les pixels parasites.

Certaines couleurs « redondantes » du GATE ARRAY (au delà de 27) peuvent avoir un effet différent de leur couleur « homologue » dans la palette, dans la mesure où les pixels parasites apparaissent (ou pas). Il est utile de rappeler ici que ces couleurs ne sont pas strictement identiques à leur homologue dans la palette, à cause de la méthode de génération des couleurs du CPC, basé sur des jeux de résistance et de la haute impédance.

9.2.2 VITESSE DE PRISE EN COMPTE

Les deux schémas pages suivantes décrivent la prise en compte de différentes instructions mettant à jour la couleur d'une encre et sa prise en compte par le GATE ARRAY/ASIC lorsqu'il affiche ses pixels.

Cette couleur sur les schémas indique quels pixels selon le mode sont affectés par le changement de couleur.

Les instructions sont calées selon **C0vs** calculé à partir du point de référence VSYNC.

ATTENTION :
Il ne faut pas perdre de vue que c'est l'affichage des pixels en mode 2 qui débute plus tôt que pour les autres modes. La mise à jour de la couleur d'une encre, quant à elle, est prise en compte de manière identique quel que soit le mode graphique.

La couleur change sur le 2ème pixel d'un octet en mode 2.
La couleur change sur le 1er pixel d'un octet pour les autres modes graphiques.

CRTC 0, 1, 2 / GATE ARRAY 40010

Z80a instr.	OUTI on Gate Array Port. Value on (HL) is VC (010ccccc)
C0 from Vsync	06 07 08 09 10 11
C0 Disp by GA	05 06 07 08 09 10
Byte Offset (0.5 µsec)	10 11 12 13 14 15 16 17 18 19 20 21
Pixel Mode 0, 3	20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43
Pixel Mode 1	40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 72 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87
Byte Offset (0.5 µsec)	10 11 12 13 14 15 16 17 18 19 20 21
Pixel Mode 2	80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176

Z80a instr.	OUT (C),r8 on Gate Array Port with r8=010ccccc, color update in the ink of the vram
C0 from Vsync	06 07 08 09 10 11
C0 Disp by GA	05 06 07 08 09 10
Byte Offset (0.5 µsec)	10 11 12 13 14 15 16 17 18 19 20 21
Pixel Mode 0	20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43
Pixel Mode 1	40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 72 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87
Byte Offset (0.5 µsec)	10 11 12 13 14 15 16 17 18 19 20 21
Pixel Mode 2	80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176

Z80a instr.	OUT (n),A on Gate Array Port.
C0 from Vsync	06 07 08 09 10 11
C0 Disp by GA	05 06 07 08 09 10
Byte Offset (0.5 µsec)	10 11 12 13 14 15 16 17 18 19 20 21
Pixel Mode 0	20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43
Pixel Mode 1	40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 72 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87
Byte Offset (0.5 µsec)	10 11 12 13 14 15 16 17 18 19 20 21
Pixel Mode 2	80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176

CRTC 3, 4 / ASIC 40489, 40226

Z80a instr.	OUTI on Gate Array Port. Value on (HL) is VC (010ccccc)
C0 from Vsync	06 07 08 09 10 11
C0 Disp by GA	05 06 07 08 09 10
Byte Offset (0.5 µsec)	10 11 12 13 14 15 16 17 18 19 20 21
Pixel Mode 0	20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43
Pixel Mode 1	40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 72 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87
Pixel Mode 2	80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175

Z80a instr.	OUT (C),r8 on Gate Array Port with r8=010ccccc, color update in the ink of the vram
C0 from Vsync	06 07 08 09 10 11
C0 Disp by GA	05 06 07 08 09 10
Byte Offset (0.5 µsec)	10 11 12 13 14 15 16 17 18 19 20 21
Pixel Mode 0	20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43
Pixel Mode 1	40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 72 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87
Pixel Mode 2	80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175

Z80a instr.	OUT (n),A on Gate Array Port.
C0 from Vsync	06 07 08 09 10 11
C0 Disp by GA	05 06 07 08 09 10
Byte Offset (0.5 µsec)	10 11 12 13 14 15 16 17 18 19 20 21
Pixel Mode 0	20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43
Pixel Mode 1	40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 72 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87
Pixel Mode 2	80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175

9.3 MODE GRAPHIQUE

9.3.1 GÉNÉRALITÉS

On peut considérer qu'une HSYNC indiquée par le CRTC est composée de plusieurs périodes au sein du GATE ARRAY.

Une période, qu'on peut nommer HSYNC-GA, a une durée maximale plafonnée à 6 µsec et ne peut pas dépasser la valeur définie dans R3.

Pour tous les CPC sans exception (CRTC 0 à 4) **il faut programmer une HSYNC de 2 µsec minimum pour que le changement de mode soit pris en compte**.

La mise à jour du registre interne contenant le mode est effective sur la 3ème µseconde de l'instruction Z80A d'I/O OUT [C],r8 à destination du GATE ARRAY.

La période de "latence" minimum pour que le GATE ARRAY génère le signal HSYNC nécessaire au moniteur est de 3 µsec.

Si la HSYNC-GA est limitée à 2 µsec (via la programmation de R3 du CRTC), alors **le signal HSYNC n'est pas généré pour le moniteur**. Dans cette situation, une HSYNC de 2 µsec n'incite pas le moniteur à rendre ses tripes. Ceci permet de changer de mode graphique en cours de ligne sans affecter la synchronisation horizontale.

La HSYNC est prise en compte plus rapidement par le GATE ARRAY sur les CRTC 0, 1 et 2, que sur les ASIC des CRTC 3 et 4.

Les ASIC **diffèrent la HSYNC de 1 µsec, comme ils le font pour l'affichage des caractères**.

Sur le principe du point de référence VSYNC, les schémas suivants décrivent **la prise en compte d'un changement de mode** par une instruction Z80A OUT(C),r8 avant et pendant la HSYNC.

CRTC 0, 1, 2

CRTC-R2=46
CRTC-R3=14 HBL Size=14 chars

				R2																																			
C0 from Vsync	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14			
C0 disp by GA	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	0	1	2	3	4	5	6	7	8	9	10	11	12			
				C3:	0	1	2	3	4	5	6	7	8	9	10	11	12	13																						
				HSYNC-G	Latency			H/C Sync																																	
MODE 2 Rq	OUT GA, MODE 0																																						
MODE 2 Rq		OUT GA, MODE 0																																					
MODE 2 Rq			OUT GA, MODE 0																																				
MODE 2 Rq				OUT GA, MODE 0																																			
MODE 2 Rq					OUT GA, MODE 0																																		
MODE 2 Rq						OUT GA, MODE 0																																	
MODE 2 Rq							OUT GA, MODE 0																																
MODE 2 Rq								OUT GA, MODE 0																															
MODE 2 Rq									OUT GA, MODE 0																														
MODE 2 Rq										OUT GA, MODE 0																													
MODE 2 Rq											OUT GA, MODE 0																											
MODE 2 Rq												OUT GA, MODE 0																										
MODE 2 Rq													OUT GA, MODE 0																									
MODE 2 Rq														OUT GA, MODE 0																								
MODE 2 Rq															OUT GA, MODE 0																							
MODE 2 Rq																OUT GA, MODE 0																						

	Résultat		
>>	**MODE 0**	OUT GA, MODE 2	
>>	**MODE 0**	OUT GA, MODE 2	
>>	**MODE 0**	OUT GA, MODE 2	
>>	**MODE 0**	OUT GA, MODE 2	
>>	**MODE 0**	OUT GA, MODE 2	
>>	**MODE 0**	OUT GA, MODE 2	
>>	**MODE 0**	OUT GA, MODE 2	
>>	**MODE 0**	OUT GA, MODE 2	
>>	MODE 2	OUT GA, MODE 2	
>>	MODE 2	OUT GA, MODE 2	
>>	MODE 2	OUT GA, MODE 2	

Characters displayed
Hsync "displayed"

CRTC 3, 4

CRTC-R2=46
CRTC-R3=14 HBL Size=14 chars

				R2																																			
C0 from Vsync	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14			
C0 disp by GA	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	0	1	2	3	4	5	6	7	8	9	10	11	12			
				C3:	0	1	2	3	4	5	6	7	8	9	10	11	12	13																						
				HSYNC-G	Latency			H/C Sync																																	
MODE 2 Rq	OUT GA, MODE 0																																						
MODE 2 Rq		OUT GA, MODE 0																																					
MODE 2 Rq			OUT GA, MODE 0																																				
MODE 2 Rq				OUT GA, MODE 0																																			
MODE 2 Rq					OUT GA, MODE 0																																		
MODE 2 Rq						OUT GA, MODE 0																																	
MODE 2 Rq							OUT GA, MODE 0																																
MODE 2 Rq								OUT GA, MODE 0																															
MODE 2 Rq									OUT GA, MODE 0																														
MODE 2 Rq										OUT GA, MODE 0																													
MODE 2 Rq											OUT GA, MODE 0																											
MODE 2 Rq												OUT GA, MODE 0																										
MODE 2 Rq													OUT GA, MODE 0																									
MODE 2 Rq														OUT GA, MODE 0																								
MODE 2 Rq															OUT GA, MODE 0																							
MODE 2 Rq																OUT GA, MODE 0																						

	Résultat		
>>	**MODE 0**	OUT GA, MODE 2	
>>	**MODE 0**	OUT GA, MODE 2	
>>	**MODE 0**	OUT GA, MODE 2	
>>	**MODE 0**	OUT GA, MODE 2	
>>	**MODE 0**	OUT GA, MODE 2	
>>	**MODE 0**	OUT GA, MODE 2	
>>	**MODE 0**	OUT GA, MODE 2	
>>	**MODE 0**	OUT GA, MODE 2	
>>	MODE 2	OUT GA, MODE 2	
>>	MODE 2	OUT GA, MODE 2	
>>	MODE 2	OUT GA, MODE 2	

Characters displayed
Hsync "displayed"

9.3.2 MODE SPLITTING

9.3.2.1 GÉNÉRALITÉS

Dans les propos suivants la terminologie **Pixel-M2** désigne un pixel dont la taille horizontale est celle d'un pixel en mode 2.
Un pixel en mode 0 ou 3 est composé de **4 Pixel-M2**.
Un pixel en mode 1 est composé de **2 Pixel-M2**.

Il est possible de changer le mode graphique durant une ligne dès lors que ce changement de mode précède une HSYNC programmée avec une longueur R3>1.

Lorsque R3 est programmé avec la valeur 2, le changement de mode effectué par le GATE ARRAY intervient après que l'affichage ait été rétabli sur les CRTC 0, 1, 2, 4. (à voir pour le 3).

Sur les CRTC 0, 2 et 4, lorsque l'affichage est réactivé par le GATE ARRAY (R3=2), **1 Pixel-M2 est cependant affiché dans l'ancien mode graphique**. Sur CRTC 1, la réactivation de l'affichage correspond exactement au début du changement de mode par le GA, ce qui ne permet pas de profiter d'un **joli pixel mode 2** de l'ancien mode graphique.

Sur les CRTC 0, 1 et 2, le GATE ARRAY bascule le changement de mode sur le 6ème pixel mode 2 (soit le 3ème pixel mode 1, le 2ème pixel mode 0).

Sur le CRTC 4, le GATE ARRAY bascule le changement de mode sur le 4ème pixel mode 2 (soit le 2ème pixel mode 1, le milieu du 1er pixel mode 0).

Étant donné que **le changement de mode intervient pendant le traitement de l'octet lu en ram**, l'algorithme de détermination du PEN par le GA bascule en cours de traitement. En conséquence les PEN calculés sont faussés pour le reste des pixels à afficher dans l'octet. Ils sont pour partie composés des rotations effectuées pour le mode précédent.

Il est par ailleurs possible de modifier, sur les CRTC 0, 1 et 2 la zone de « non affichage HSYNC » via des techniques nommées **R2.JIT** et **R3.JIT**. (JIT= « Just In Time »).

R2.JIT réduit la zone de non affichage de la HSYNC par la gauche et **R3.JIT** stoppe la **HSYNC** 0.25 µsec après sa fin prévue. (voir chapitres 16 et 14 pour plus d'informations).

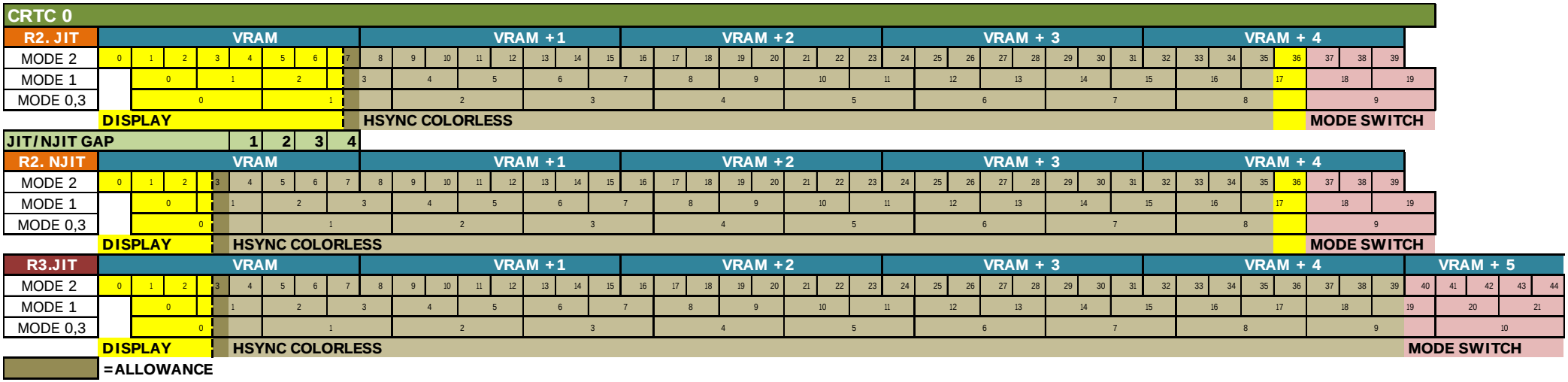
Dans le contexte d'un changement de mode graphique, la technique **R3.JIT** ne permet pas de raccourcir la période nécessaire au changement de mode graphique. (Vous ne pensiez tout de même pas que je n'allais pas essayer…). En effet, si cette technique est utilisée sur la 2ème µsec de la **HSYNC**, le changement de mode graphique n'a pas lieu. Le changement de MODE est pris en compte uniquement sur un **R3.JIT** pendant la 3ème µsec. Ce qui implique une durée de non-affichage de 2.25 µsec pour un changement de mode. Cela présente néanmoins l'intérêt de déplacer le moment ou un pixel mode 2 « malaxé » est visualisé dans un autre mode graphique. Le **R3.JIT** présente également un intérêt énorme dans la mesure où **il permet de modifier la durée des 4 µsec « officielles » de la HSYNC moniteur** avec un gap de 0.25 µsec au lieu des 0.5 µsec utilisés « classiquement » pour réaliser des scrollings horizontaux à l'octet.

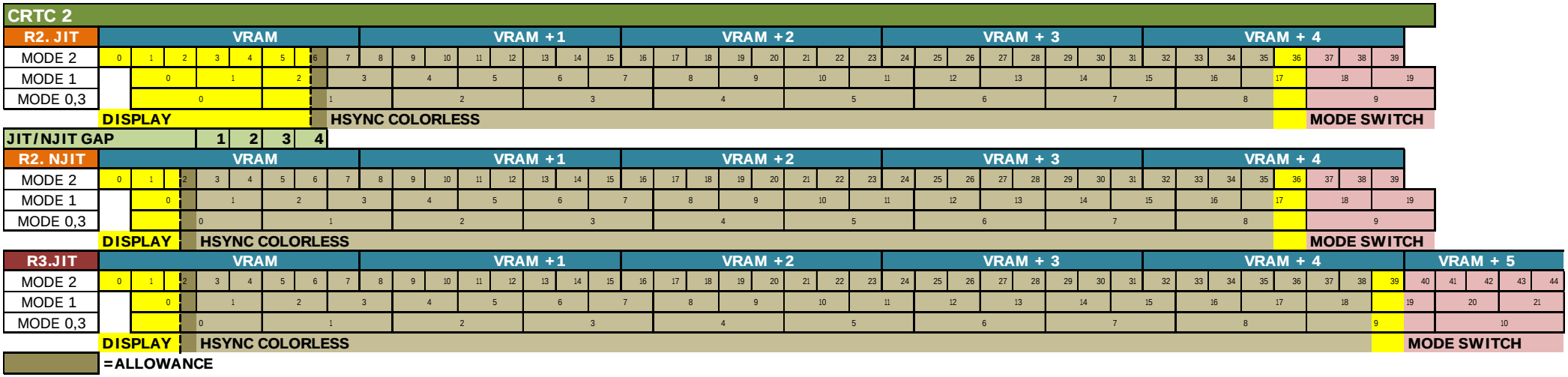
Le retard d'arrêt d'affichage réalisé avec **R2.JIT** ne change rien au moment où l'affichage est réactivé, contrairement au **R3.JIT**. Le GAP entre **R2.JIT** et **R2.NJIT** est de **3 Pixel-M2 (0.1875 µsec) sur CRTC 1** et de **4 Pixel-M2 sur CRTC 0 et 2 (0.25 µsec)**.

9.3.2.2 HSYNC AU MICROSCOPE

Les schémas sur les pages suivantes indiquent, selon les CRTC :

- Le moment où l'affichage cesse lorsqu'une HSYNC a été demandée (avec toutefois une tolérance de ½ Pixel-M2 (**0.03125 µsec**)) en utilisant **R2.JIT** ou **R2.NJIT**.
- Le gap en Pixel-M2 entre une technique **R2.JIT** et **R2.NJIT**.
- Le moment où l'affichage reprend durant 1 **Pixel-M2** (zone indiquée en jaune).
- Le moment où le mode graphique change (indiqué en rose).
- Le moment où l'affichage cesse lorsqu'une **HSYNC** est stoppée en utilisant **R3.JIT.**


CRTC 0
R2. JIT
VRAM
VRAM +1
VRAM +2
VRAM + 3
VRAM + 4
MODE 2
MODE 1
MODE 0,3
DISPLAY
HSYNC COLORLESS
MODE SWITCH
JIT/NJIT GAP
1
2
3
4
R2. NJIT
R3.JIT
VRAM + 5
=ALLOWANCE


CRTC 2
R2. JIT
VRAM
VRAM +1
VRAM +2
VRAM + 3
VRAM + 4
MODE 2
MODE 1
MODE 0,3
DISPLAY
HSYNC COLORLESS
MODE SWITCH
JIT/NJIT GAP
1
2
3
4
R2. NJIT
R3.JIT
VRAM + 5
=ALLOWANCE

CRTC 1

R2. JIT	VRAM	VRAM +1	VRAM +2	VRAM + 3	VRAM + 4
MODE 2	0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23	24 25 26 27 28 29 30 31	32 33 34 35 36 37 38 39
MODE 1	0 1 2 3	4 5 6 7	8 9 10 11	12 13 14 15	16 17 18 19
MODE 0,3	0 1	2 3	4 5	6 7	8 9
	DISPLAY	HSYNC COLORLESS			MODE SWITCH

JIT/NJIT GAP	1	2	3

R2. NJIT	VRAM	VRAM +1	VRAM +2	VRAM + 3	VRAM + 4
MODE 2	0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23	24 25 26 27 28 29 30 31	32 33 34 35 36 37 38 39
MODE 1	0 1 2 3	4 5 6 7	8 9 10 11	12 13 14 15	16 17 18 19
MODE 0,3	0 1	2 3	4 5	6 7	8 9
	DISPLAY	HSYNC COLORLESS			MODE SWITCH

R3.JIT	VRAM	VRAM +1	VRAM +2	VRAM + 3	VRAM + 4	VRAM + 5
MODE 2	0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23	24 25 26 27 28 29 30 31	32 33 34 35 36 37 38 39	40 41 42 43 44
MODE 1	0 1 2 3	4 5 6 7	8 9 10 11	12 13 14 15	16 17 18	19 20 21
MODE 0,3	0 1	2 3	4 5	6 7	8 9	10
	DISPLAY	HSYNC COLORLESS				MODE SWITCH

=ALLOWANCE

CRTC 4

R2. NJIT	VRAM +1	VRAM +2	VRAM + 3	VRAM + 4	VRAM + 5	VRAM + 6
MODE 2	0 1 2 3 4 5 6 7 8	9 10 11 12 13 14 15	16 17 18 19 20 21 22 23	24 25 26 27 28 29 30 31	32 33 34 35 36 37 38 39	40 41 42 43 44 45 46 47
MODE 1	0 1 2 3	4 5 6 7	8 9 10 11	12 13 14 15	16 17 18 19	20 21 22 23
MODE 0,3	0 1	2 3	4 5	6 7	8 9	10 11
	DISPLAY	HSYNC COLORLESS				MODE SWITCH

=ALLOWANCE

9.3.2.3 CRTC 0, 1, 2 : CUISINE DE PIXELS EN R3.NJIT

Lorsque le GATE ARRAY bascule d'un mode graphique vers un autre mode, il est déjà en train de traiter les données d'un octet selon le mode précédent, et le résultat final obtenu traduit sa logique interne.

En **R2.NJIT**, l'affichage cesse :
- Durant 2 µsec sur CRTC 0 et 1 (32 Pixel-M2).
- Durant 2.0625 µsec sur CRTC 2 (33 Pixel-M2).

En **R2.JIT**, l'affichage cesse :
- Durant 1.75 µsec sur CRTC 0 (28 Pixel-M2).
- Durant 1.8125 µSec sur CRTC 1 et 2 (29 Pixel-M2).

Sur les CRTC 0 et 2, l'affichage est rétabli 1 Pixel-M2 avant que le GATE ARRAY change le mode graphique. **Le 30ème Pixel-M2 affiché dispose donc des caractéristiques de l'ancien mode, pour sa fraction correspondante à 1 Pixel-M2.**

Sur CRTC 1, il faut considérer que la zone « **Hsync No Disp** » indiquée sur les schémas est plus longue de 1 Pixel-M2. Le premier Pixel-M2 obtenu lorsque l'affichage est activé n'est affiché que pour les CRTC 0, 2 et 4. Le CRTC 1 n'affiche pas ce pixel.
Ce pixel additionnel est symbolisé en violet sur les schémas qui suivent.

Il existe une différence notable entre les GATE ARRAY 40007/40008 et les 40010. En effet, dans certaines situations, lorsque des bits de la donnée en VRAM ont déjà été utilisés dans le calcul d'un numéro de couleur des pixels du mode de départ, ces derniers sont considérés à 0 (GA 40010) ou 1 (GA 40008) pour le calcul des numéros de couleur du mode d'arrivée.

9.3.2.3.1 MODE 2 VERS MODE 0.1.2.3

Sur une ligne affichée en MODE 2, l'affichage des données reprend :
- Sur CRTC 0 et 2, à partir du **5ème Pixel-M2 du 5ème octet de la VRAM**.
- Sur CRTC 1, à partir du **6ème Pixel-M2 du 5ème octet de la VRAM**.

Sur CRTC 0 et 2, le premier pixel affiché est en MODE 2 (PEN 1 ou 0).

Étant donné que les pixels MODE 2 sont « en avance » sur les pixels des autres modes, le GATE ARRAY va afficher **4 nouveaux Pixel-M2 après le 5ème**, sauf si le mode n'a pas changé.
Cela représente l'affichage de 9 pixels-M2 à partir d'un même octet.

Le GATE ARRAY considère que son compteur de bits n'est pas à 0 mais à 5.

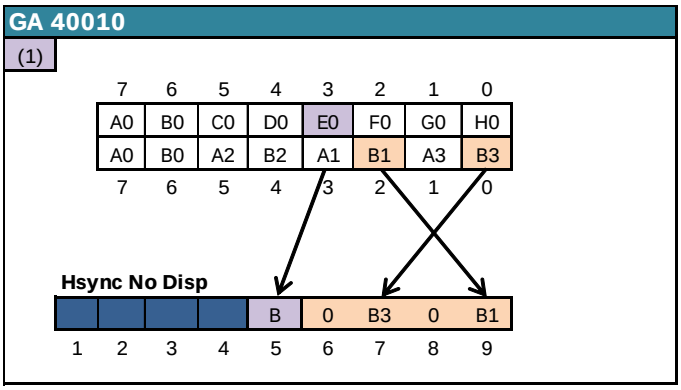
À partir du 6ème bit de l'octet lu en VRAM, il « débute » un traitement de calcul du numéro de couleur comme si c'étaient les 1ers bits d'un octet. Ainsi il considère les bits b2.b1.b0 de l'octet en VRAM comme ceux qu'il aurait rencontré dans les bits b7.b6.b5 de ce même octet.

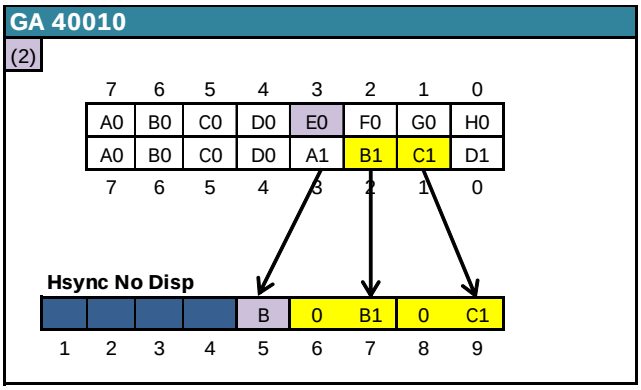
Les bits manqués par le GATE ARRAY 40010 sont considérés comme valant 0, ce qui limite les combinaisons de numéros de couleur :
- **MODE 2 vers MODE 0** : Numéros de couleur **0, 1, 4, 5**
- **MODE 2 vers MODE 1** : Numéros de couleur **0, 1**
- **MODE 2 vers MODE 3** : Numéros de couleur **0, 1**

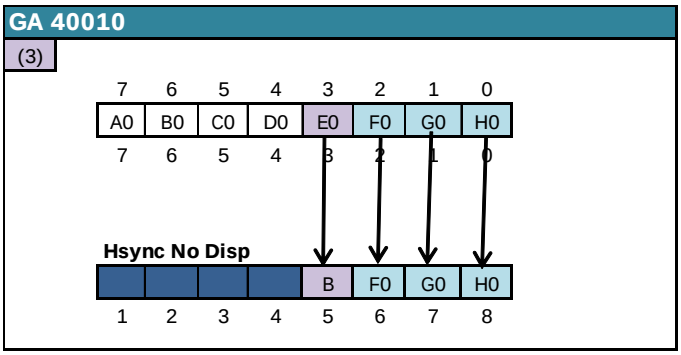
Le tableau ci-après décrit l'interprétation par le GATE ARRAY 40010 de l'octet de VRAM traité lorsque le réaffichage est activé, selon le nouveau mode graphique demandé.

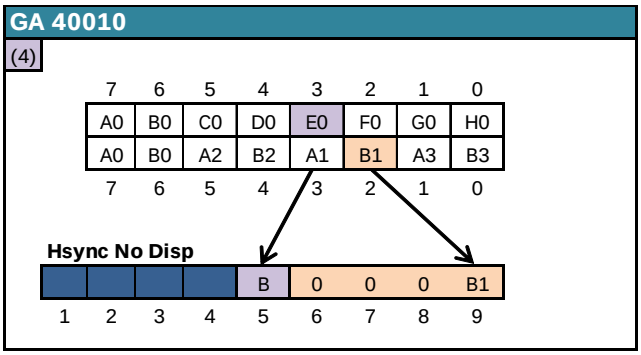
Mode to Mode		VRAM Byte							
2	0	b7	b6	b5	b4	b3	b2	b1	b0
2	1	b7	b6	b5	b4	b3	b2	b1	b0
2	2	b7	b6	b5	b4	b3	b2	b1	b0
2	3	b7	b6	b5	b4	b3	b2	b1	b0

GA 40010 : Displayed Pixels								
Hsync No Disp	(1)	0	b0	0	b2			
Hsync No Disp	(2)	0	b2	0	b1			
Hsync No Disp	(3)	b2	b1	b0				
Hsync No Disp	(4)	0	0	0	b2			
1	2	3	4	5	6	7	8	9









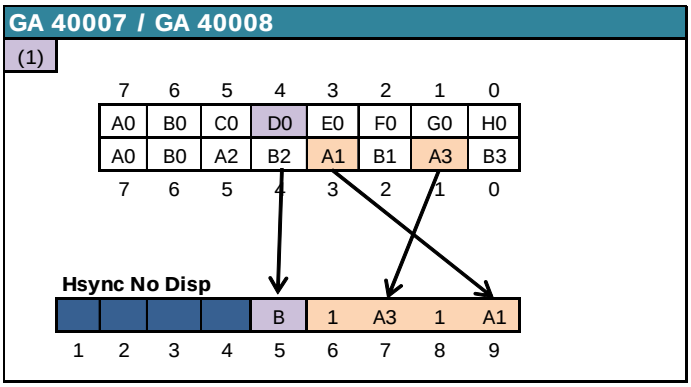
Les GATE ARRAY 40007 et 40008 sont en avance de 0.0625 µsec par rapport au 40010 lorsqu'ils récupèrent les bits permettant de constituer le numéro de couleur. Les bits manqués sont considérés comme valant 1, ce qui limite les combinaisons des numéros de couleur :

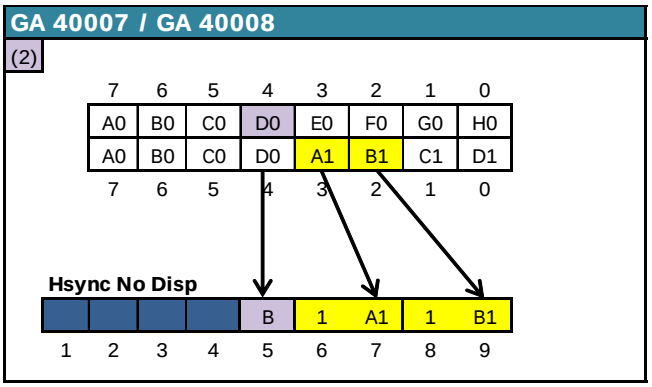
- **MODE 2 vers MODE 0** : Numéros de couleur **10, 11, 14, 15**
- **MODE 2 vers MODE 1** : Numéros de couleur **2, 3**
- **MODE 2 vers MODE 3** : Numéros de couleur **2, 3**

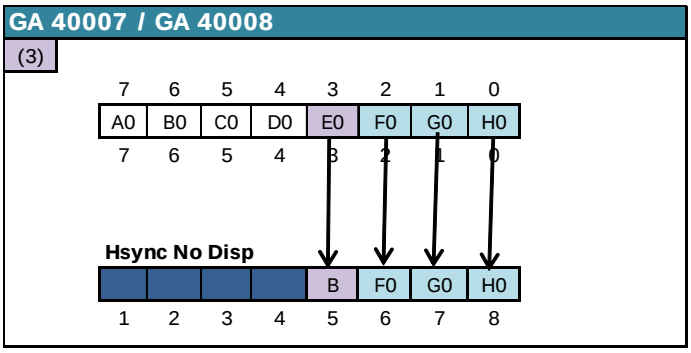
Le tableau ci-après décrit l'interprétation par les **GATE ARRAY 40007 et 40008** de l'octet de VRAM traité lorsque le réaffichage est activé, selon le nouveau mode graphique demandé.

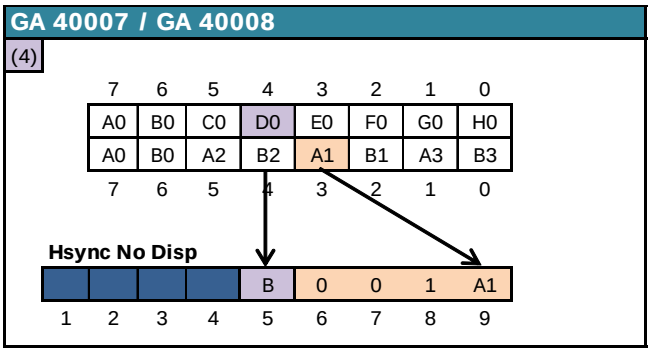
Mode to Mode		VRAM Byte							
2	0	b7	b6	b5	b4	b3	b2	b1	b0
2	1	b7	b6	b5	b4	b3	b2	b1	b0
2	2	b7	b6	b5	b4	b3	b2	b1	b0
2	3	b7	b6	b5	b4	b3	b2	b1	b0

GA 40007/8 : Displayed Pixels								
Hsync No Disp	(1)	1	b1	1	b3			
Hsync No Disp	(2)	1	b3	1	b2			
Hsync No Disp	(3)	b2	b1	b0				
Hsync No Disp	(4)	0	0	1	b3			
1	2	3	4	5	6	7	8	9









9.3.2.3.2 MODE 0 vers MODE 0.1.2.3

Sur une ligne affichée en MODE 0, l'affichage des données reprend :

- Sur CRTC 0 et 2, à partir du **4ème Pixel-M2 du 5ème octet de la VRAM**.
- Sur CRTC 1, à partir du **5ème Pixel-M2 du 5ème octet de la VRAM**.

Sur CRTC 0 et 2, le premier Pixel-M2 affiché est le dernier des **4 pixel-M2 du pixel MODE 0** affiché dans sa couleur d'origine (soit un numéro de couleur compris entre 0 et 15). Autrement dit, seul ¼ du pixel MODE 0 est affiché avant que le GATE ARRAY affiche les pixels suivants.

Si le nouveau mode requis est le MODE 2, alors le GATE ARRAY va afficher 3 nouveaux Pixel-M2 après le 4ème, ce qui représente au total l'affichage de 7 pixels-M2 à partir du même octet.

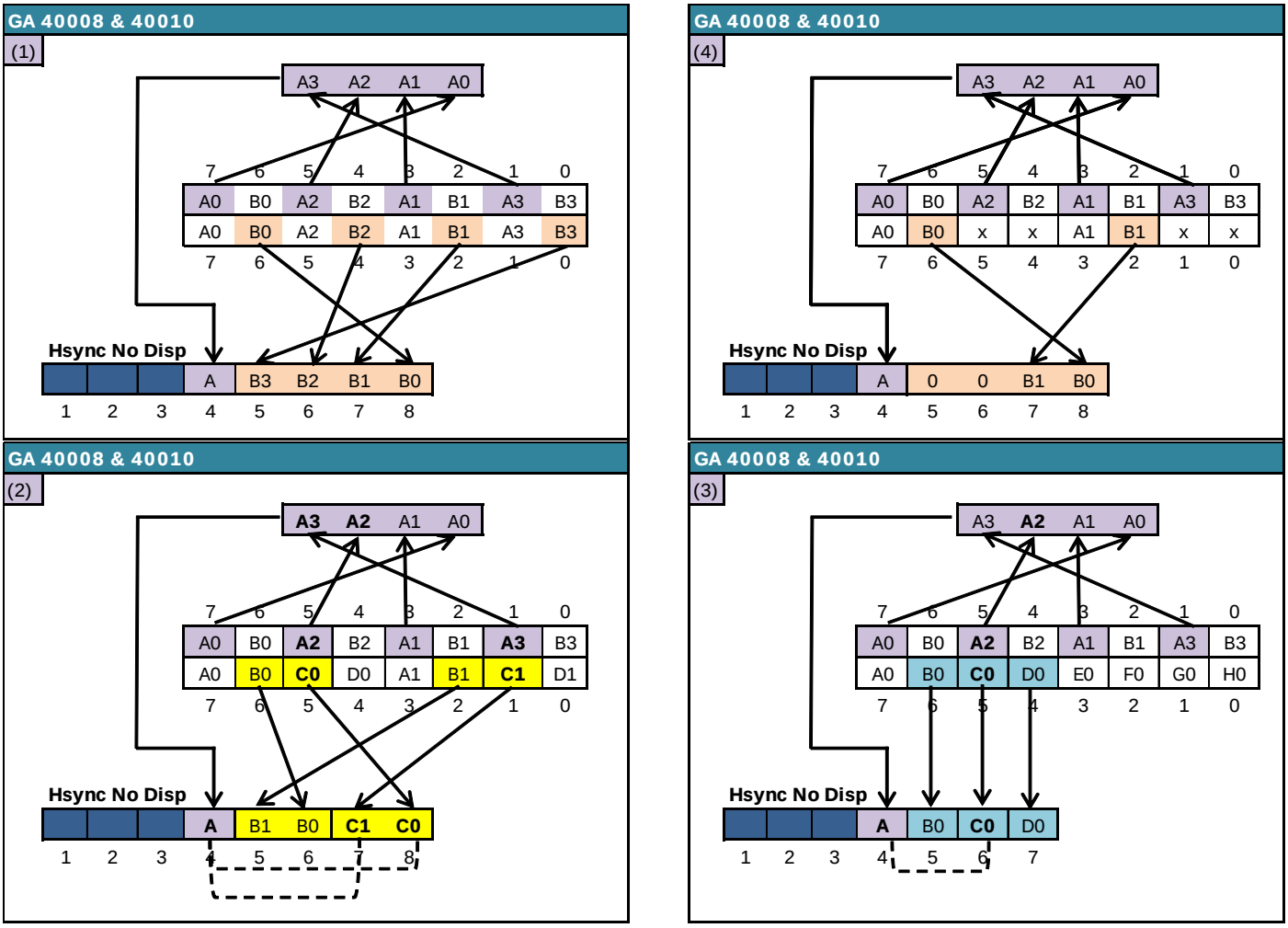
Si le nouveau mode requis est différent du MODE 2, alors le GATE ARRAY va afficher 4 nouveaux Pixel-M2 après le 4ème.

Enfin, si le nouveau mode requis est 1 ou 2, alors le GATE ARRAY, pour calculer les nouveaux numéros de couleur, « réutilise » des bits déjà utilisés pour calculer le numéro de couleur du pixel précédent.

Le tableau ci-après décrit l'interprétation par les GATE ARRAY 40007, 40008 et 40010 de l'octet de VRAM traité lorsque l'affichage est réactivé, selon le nouveau mode graphique demandé.

Mode to Mode		VRAM Byte							
0	0	b7	b6	b5	b4	b3	b2	b1	b0
0	1	b7	b6	b5	b4	b3	b2	b1	b0
0	2	b7	b6	b5	b4	b3	b2	b1	b0
0	3	b7	b6	b5	b4	b3	b2	b1	b0

Displayed Pixels							
Hsync No Disp	(1)	b0	b4	b2	b6		
Hsync No Disp	(2)	b2	b6	b1	b5		
Hsync No Disp	(3)	b6	b5	b4			
Hsync No Disp	(4)	0	0	b2	b6		
1	2	3	4	5	6	7	8



Lors du passage du MODE 0 vers un mode 1 ou 2, l'utilisation commune des bits de données de l'octet lu en VRAM crée une contrainte en liant les numéros de couleur du Pixel-M2 affiché en position 4 avec celui d'un des pixels situé après.

- **MODE 0 vers MODE 1** : Le numéro de couleur du dernier pixel en MODE 1 correspond aux bits 3 et 4 du numéro de couleur du 1er pixel en MODE 0.
  - 1er pixel-M2 N° couleurs 0,1,2,3 = 2ème pixel MODE 1 avec numéro de couleur 0
  - 1er pixel-M2 N° couleurs 4,5,6,7 = 2ème pixel MODE 1 avec numéro de couleur 1
  - 1er pixel-M2 N° couleurs 8,9,10,11 = 2ème pixel MODE 1 avec numéro de couleur 2
  - 1er pixel-M2 N° couleurs 12,13,14,15 = 2ème pixel MODE 1 avec numéro de couleur 2
- **MODE 0 vers MODE 2** : Le numéro de couleur du 2ème pixel en MODE 2 correspond au bit 2 du numéro de couleur du 1er pixel en MODE 0.
  - 1er pixel-M2 N° couleurs 0..3, 8..11 = 2ème pixel MODE 2 de couleur 0

- 1er pixel-M2 N° couleurs 4..7, 12..15 = 2ème pixel MODE 2 de couleur 1

9.3.2.3.3 MODE 1 VERS MODE 0.1.2.3

Sur une ligne affichée en MODE 1, l'affichage des données reprend :

- Sur CRTC 0 et 2, à partir du **4ème Pixel-M2 du 5ème octet de la VRAM**.
- Sur CRTC 1, à partir du **5ème Pixel-M2 du 5ème octet de la VRAM**.

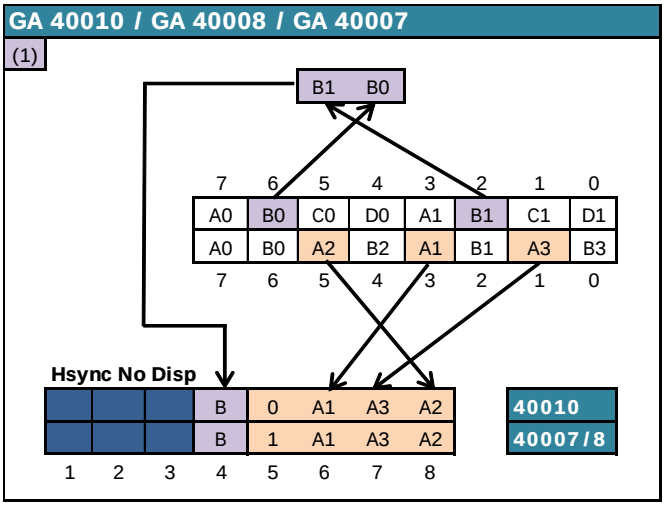
Sur CRTC 0 et 2, le premier Pixel-M2 affiché est le dernier des **2 pixel-M2 du pixel MODE 1** affiché dans sa couleur d'origine (soit un numéro de couleur compris entre 0 et 3). Autrement dit, seule la moitié du pixel MODE 1 est affichée avant que le GATE ARRAY affiche les pixels suivants.

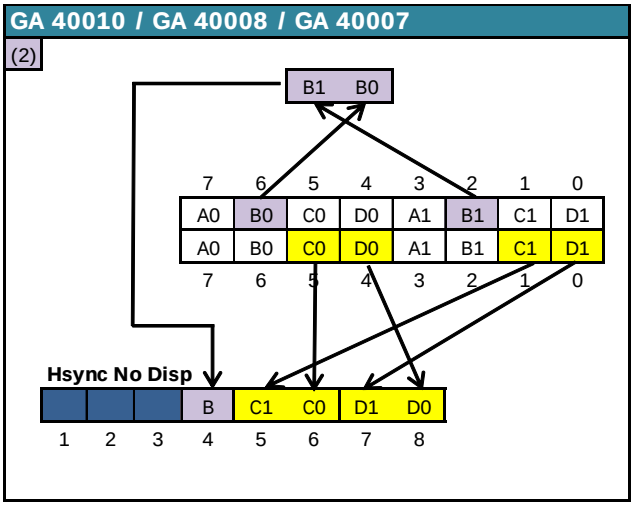
Si le nouveau mode requis est le MODE 2, alors le GATE ARRAY va afficher 3 nouveaux Pixel-M2 après le 4ème, ce qui représente au total l'affichage de 7 pixels-M2 à partir du même octet.
Si le nouveau mode requis est différent du MODE 2, alors le GATE ARRAY va afficher 4 nouveaux Pixel-M2 après le 4ème.

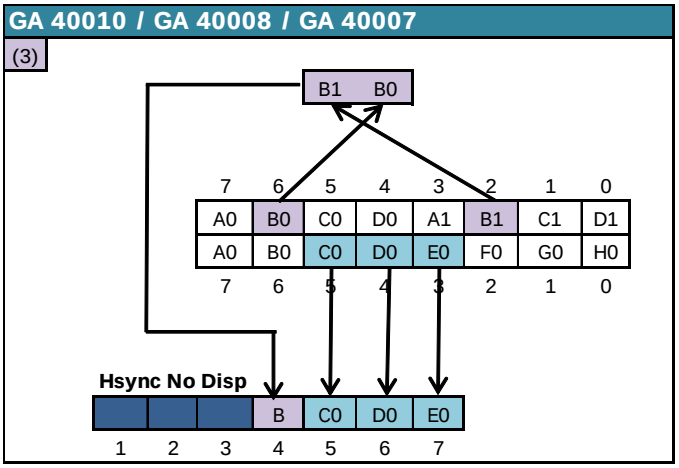
Le tableau suivant décrit l'interprétation par les GATE ARRAY 40007,40008 et 40010 de l'octet de VRAM traité lorsque l'affichage est réactivé, selon le nouveau mode graphique demandé.

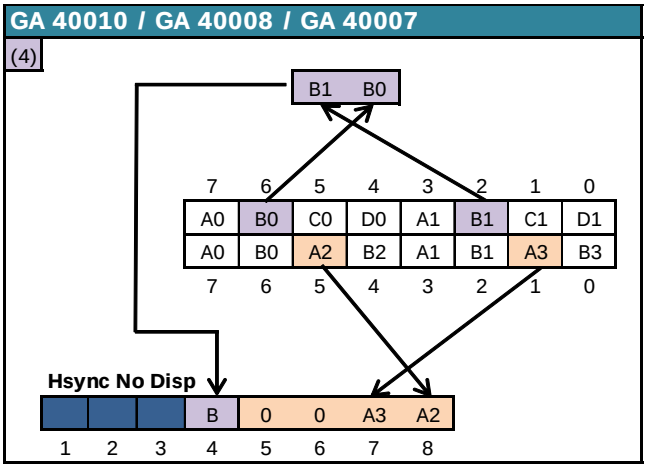
Mode to Mode		VRAM Byte							
1	0	b7	b6	b5	b4	b3	b2	b1	b0
1	1	b7	b6	b5	b4	b3	b2	b1	b0
1	2	b7	b6	b5	b4	b3	b2	b1	b0
1	3	b7	b6	b5	b4	b3	b2	b1	b0

Displayed Pixels							
Hsync No Disp	(1)	0/1	b3	b1	b5		
Hsync No Disp	(2)	b1	b5	b0	b4		
Hsync No Disp	(3)	b5	b4	b3			
Hsync No Disp	(4)	0	0	b1	b5		
1	2	3	4	5	6	7	8









9.3.2.3.4 MODE 3 VERS MODE 0.1.2.3

Sur une ligne affichée en MODE 3, l'affichage des données reprend :

- Sur CRTC 0 et 2, à partir du **4ème Pixel-M2 du 5ème octet de la VRAM**.
- Sur CRTC 1, à partir du **5ème Pixel-M2 du 5ème octet de la VRAM**.

Sur CRTC 0 et 2, le premier Pixel-M2 affiché est le dernier des **4 pixel-M2 du pixel MODE 3** affiché dans sa couleur d'origine (soit un numéro de couleur compris entre 0 et 3). Autrement dit, seul ¼ du pixel MODE 3 est affiché avant que le GATE ARRAY affiche les pixels suivants.

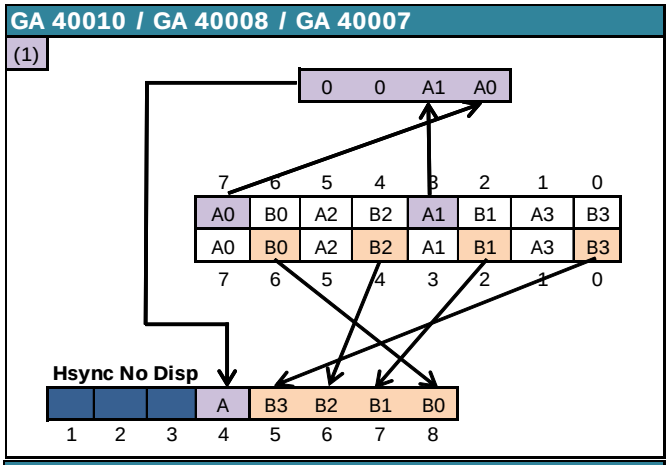
Si le nouveau mode requis est le MODE 2, alors le GATE ARRAY va afficher 3 nouveaux Pixel-M2 après le 4ème, ce qui représente au total l'affichage de 7 pixels-M2 à partir du même octet.

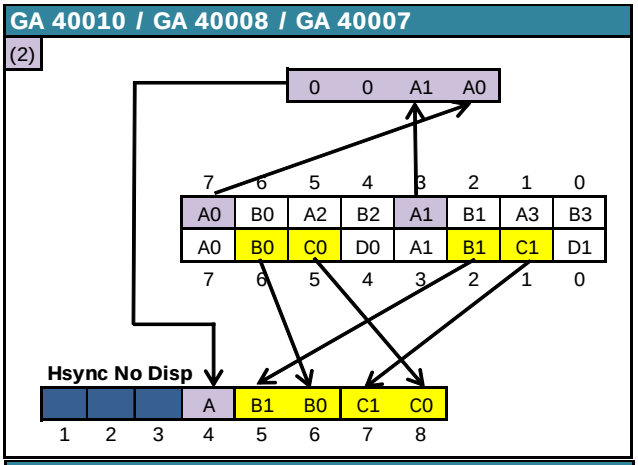
Si le nouveau mode requis est différent du MODE 2, alors le GATE ARRAY va afficher 4 nouveaux Pixel-M2 après le 4ème.

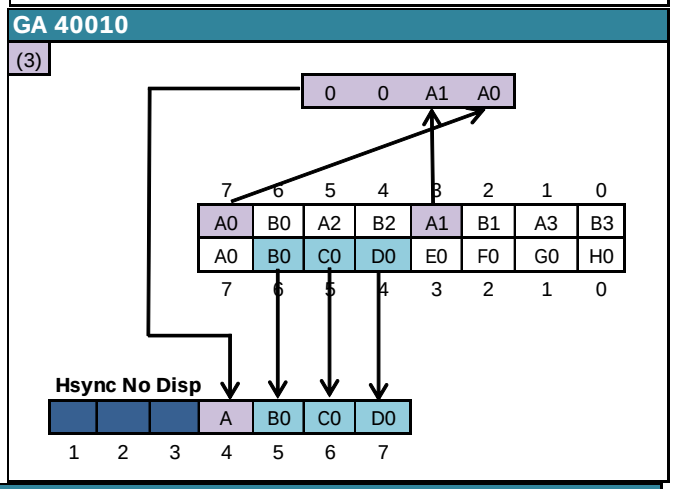
Le tableau suivant décrit l'interprétation par le GATE ARRAY de l'octet de VRAM traité lorsque l'affichage est réactivé, selon le nouveau mode graphique demandé.

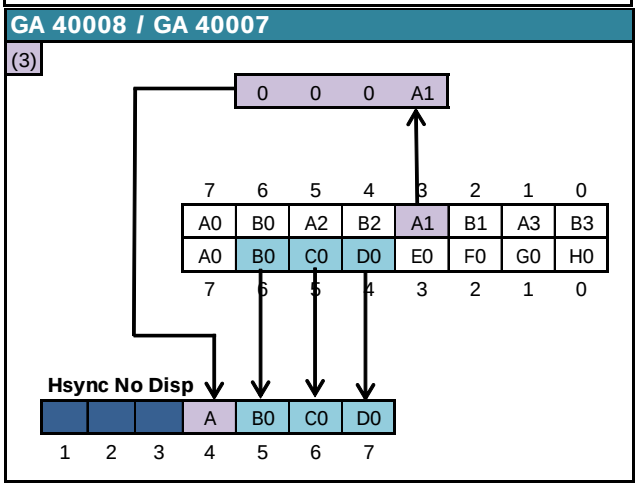
Mode to Mode		VRAM Byte							
3	0	b7	b6	b5	b4	b3	b2	b1	b0
3	1	b7	b6	b5	b4	b3	b2	b1	b0
3	2	b7	b6	b5	b4	b3	b2	b1	b0
3	3	b7	b6	b5	b4	b3	b2	b1	b0

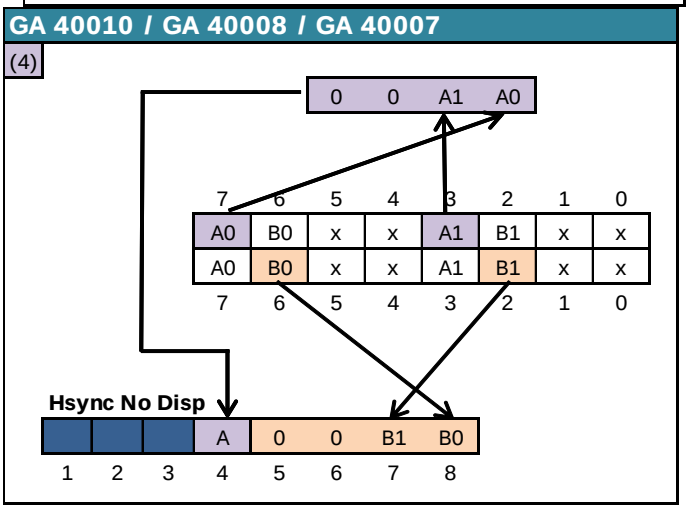
Displayed Pixels							
Hsync No Disp	(1)	b0	b4	b2	b6		
Hsync No Disp	(2)	b2	b6	b1	b5		
Hsync No Disp	(3)	b6	b5	b4			
Hsync No Disp	(4)	0	0	b2	b6		
1	2	3	4	5	6	7	8


GA 40010 / GA 40008 / GA 40007
(1)
0 0 A1 A0
7 6 5 4 3 2 1 0
A0 B0 A2 B2 A1 B1 A3 B3
A0 B0 A2 B2 A1 B1 A3 B3
7 6 5 4 3 2 1 0
Hsync No Disp
A B3 B2 B1 B0
1 2 3 4 5 6 7 8


GA 40010 / GA 40008 / GA 40007
(2)
0 0 A1 A0
7 6 5 4 3 2 1 0
A0 B0 A2 B2 A1 B1 A3 B3
A0 B0 C0 D0 A1 B1 C1 D1
7 6 5 4 3 2 1 0
Hsync No Disp
A B1 B0 C1 C0
1 2 3 4 5 6 7 8


GA 40010
(3)
0 0 A1 A0
7 6 5 4 3 2 1 0
A0 B0 A2 B2 A1 B1 A3 B3
A0 B0 C0 D0 E0 F0 G0 H0
7 6 5 4 3 2 1 0
Hsync No Disp
A B0 C0 D0
1 2 3 4 5 6 7


GA 40008 / GA 40007
(3)
0 0 0 A1
7 6 5 4 3 2 1 0
A0 B0 A2 B2 A1 B1 A3 B3
A0 B0 C0 D0 E0 F0 G0 H0
7 6 5 4 3 2 1 0
Hsync No Disp
A B0 C0 D0
1 2 3 4 5 6 7


GA 40010 / GA 40008 / GA 40007
(4)
0 0 A1 A0
7 6 5 4 3 2 1 0
A0 B0 x x A1 B1 x x
A0 B0 x x A1 B1 x x
7 6 5 4 3 2 1 0
Hsync No Disp
A 0 0 B1 B0
1 2 3 4 5 6 7 8

9.3.2.4 CRTC 0, 1, 2 : CUISINE DE PIXELS EN R3.JIT

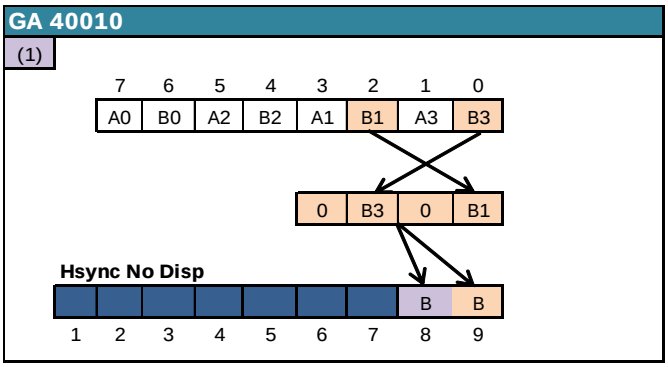
La technique **R3.JIT** permet de retarder la fin d'une HSYNC de 0.25 µsec.

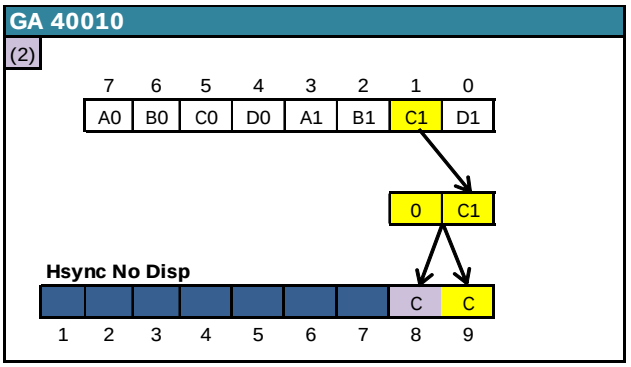
Sur CRTC 0 et 1, la **HSYNC** cesse après le dernier Pixel-M2 noir de la HSYNC.
Sur CRTC 2, la **HSYNC** cesse sur le dernier Pixel-M2 de la HSYNC et permet la visualisation d'un pixel additionnel.
Ce pixel additionnel est symbolisé en violet sur les schémas qui suivent.

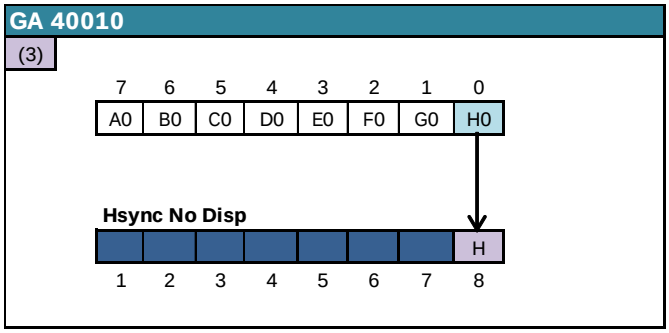
9.3.2.4.1 MODE 2 VERS MODE 0.1.2.3

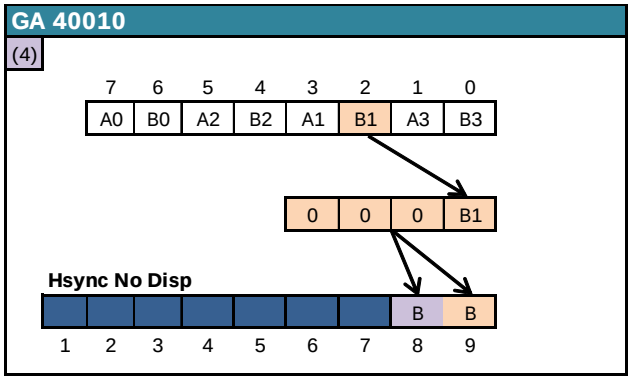
Mode to Mode		VRAM Byte							
2	0	b7	b6	b5	b4	b3	b2	b1	b0
2	1	b7	b6	b5	b4	b3	b2	b1	b0
2	2	b7	b6	b5	b4	b3	b2	b1	b0
2	3	b7	b6	b5	b4	b3	b2	b1	b0

GA 40010 : Displayed Pixels									
Hsync No Disp								(1)	(1)
Hsync No Disp								(2)	b1
Hsync No Disp								(3)	
Hsync No Disp								(4)	(4)
1	2	3	4	5	6	7	8	9	



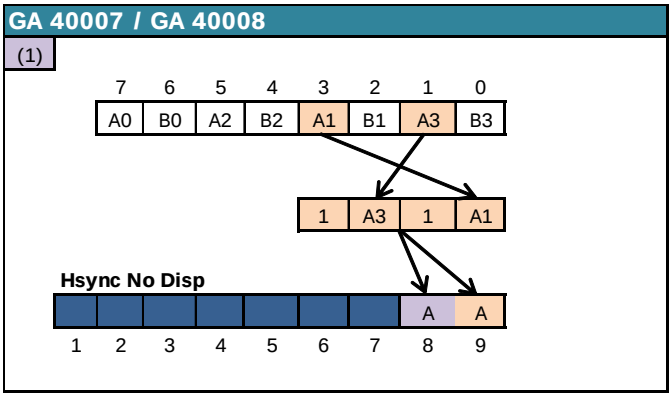


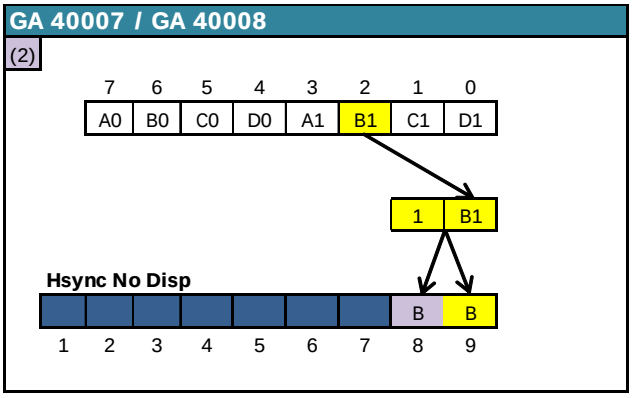


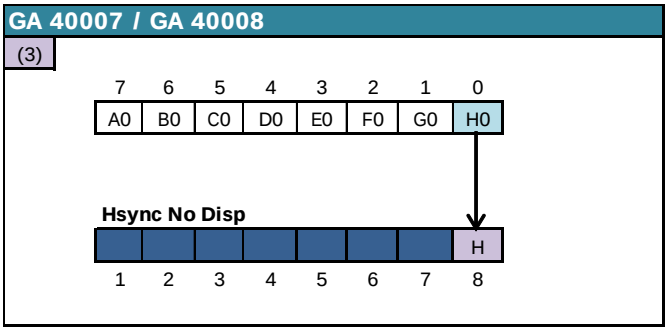


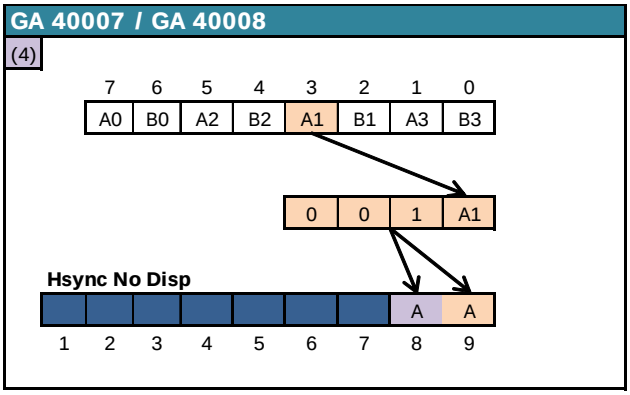
Mode to Mode		VRAM Byte							
2	0	b7	b6	b5	b4	b3	b2	b1	b0
2	1	b7	b6	b5	b4	b3	b2	b1	b0
2	2	b7	b6	b5	b4	b3	b2	b1	b0
2	3	b7	b6	b5	b4	b3	b2	b1	b0

GA 40007/8 : Displayed Pixels									
Hsync No Disp								(1)	(1)
Hsync No Disp								(2)	b2
Hsync No Disp								(3)	
Hsync No Disp								(4)	(4)
1	2	3	4	5	6	7	8	9	





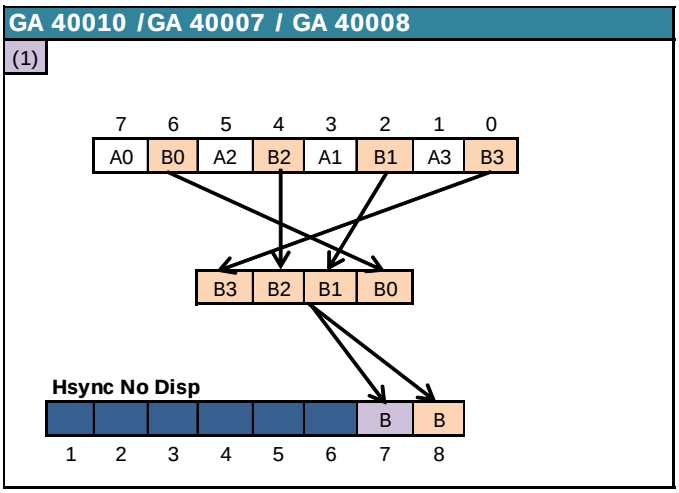


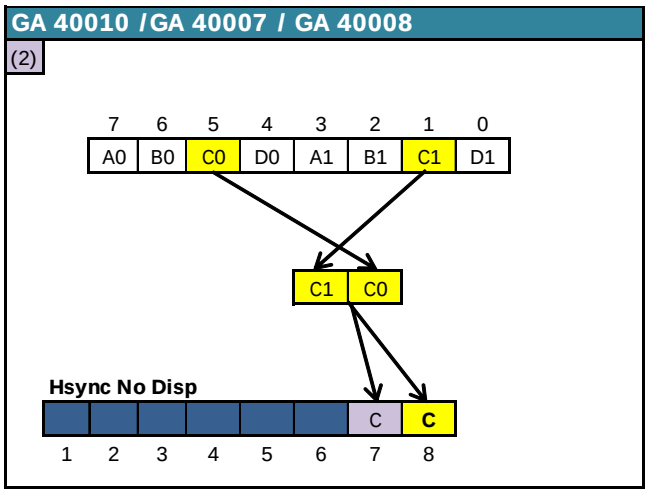


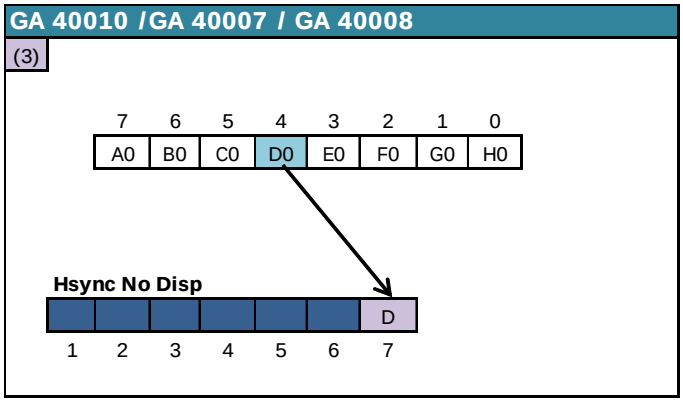
9.3.2.4.2 MODE 0 VERS MODE 0.1.2.3

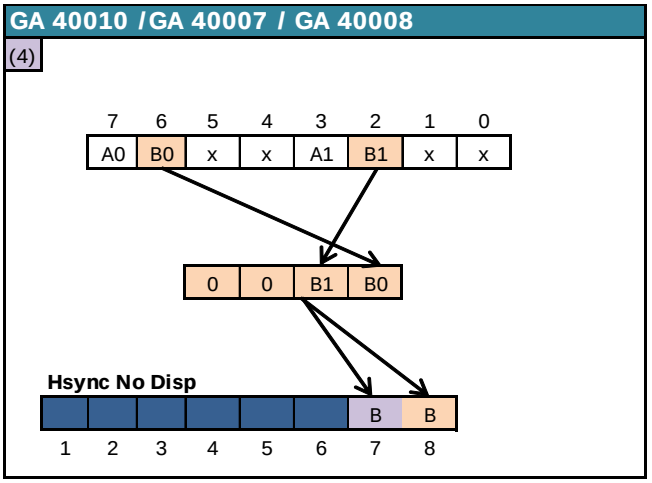
Mode to Mode		VRAM Byte							
0	0	b7	b6	b5	b4	b3	b2	b1	b0
0	1	b7	b6	b5	b4	b3	b2	b1	b0
0	2	b7	b6	b5	b4	b3	b2	b1	b0
0	3	b7	b6	b5	b4	b3	b2	b1	b0

Displayed Pixels		
Hsync No Disp	(1)	(1)
Hsync No Disp	(2)	(2)
Hsync No Disp	(3)	
Hsync No Disp	(4)	(4)
1 2 3 4 5 6 7 8		





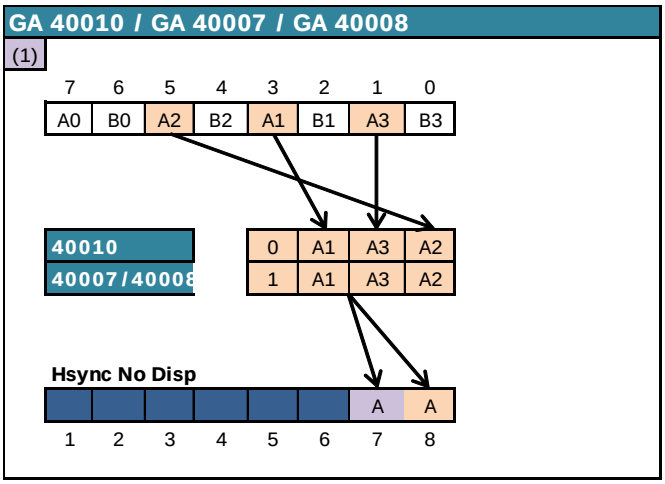


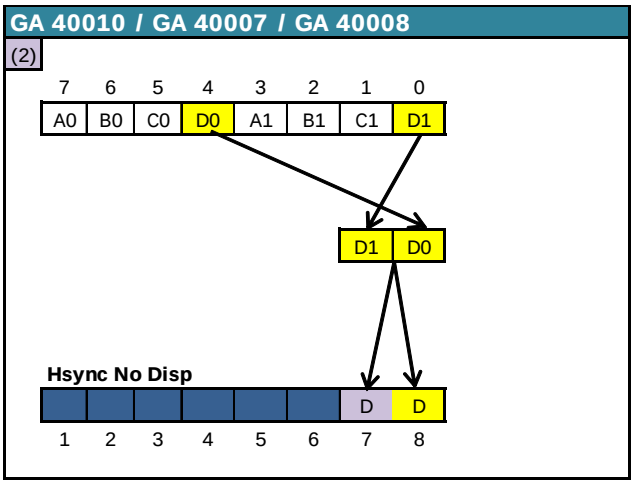


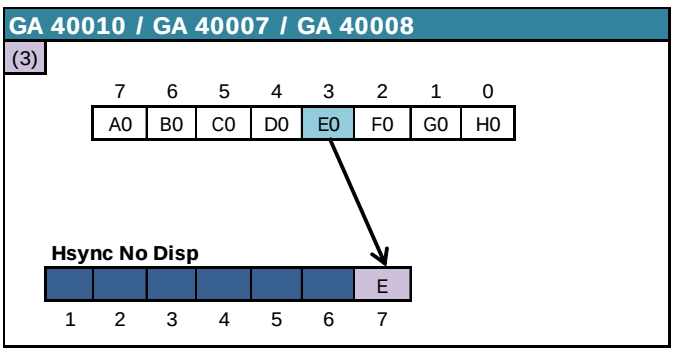
9.3.2.4.3 MODE 1 VERS MODE 0.1.2.3

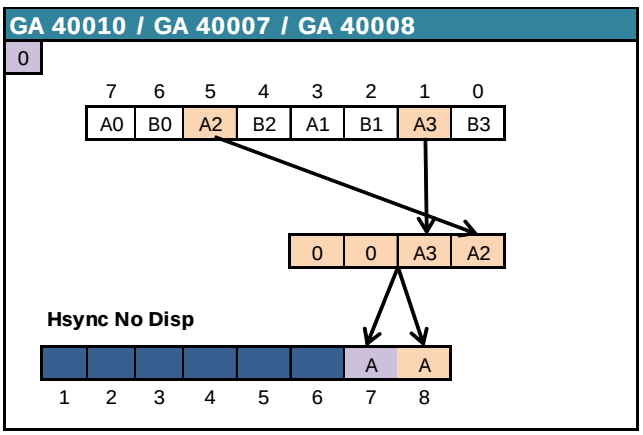
Mode to Mode		VRAM Byte							
1	0	b7	b6	b5	b4	b3	b2	b1	b0
1	1	b7	b6	b5	b4	b3	b2	b1	b0
1	2	b7	b6	b5	b4	b3	b2	b1	b0
1	3	b7	b6	b5	b4	b3	b2	b1	b0

Displayed Pixels		
Hsync No Disp	(1)	(1)
Hsync No Disp	(2)	(2)
Hsync No Disp	(3)	
Hsync No Disp	(4)	(4)
1 2 3 4 5 6	7	8









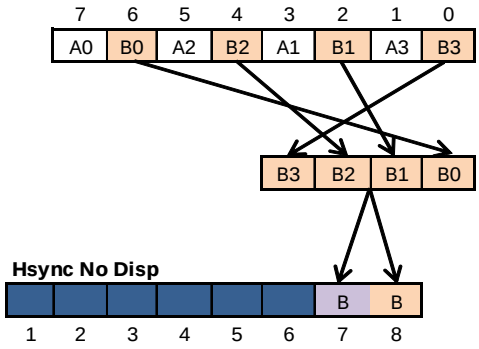
9.3.2.4.4 MODE 3 VERS MODE 0.1.2.3

Mode to Mode		VRAM Byte							
3	0	b7	b6	b5	b4	b3	b2	b1	b0
3	1	b7	b6	b5	b4	b3	b2	b1	b0
3	2	b7	b6	b5	b4	b3	b2	b1	b0
3	3	b7	b6	b5	b4	b3	b2	b1	b0

Displayed Pixels								
Hsync No Disp						(1)	(1)	
Hsync No Disp						(2)	(2)	
Hsync No Disp						(3)		
Hsync No Disp						(4)	(4)	
1	2	3	4	5	6	7	8	

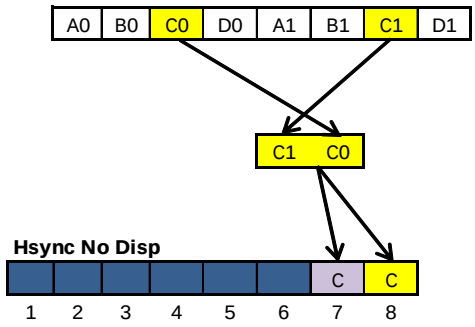
GA 40010 / GA 40007 / GA 40008

(1)



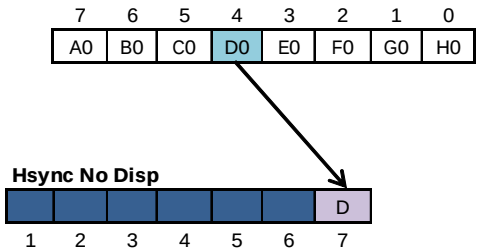
GA 40010 / GA 40007 / GA 40008

(2)



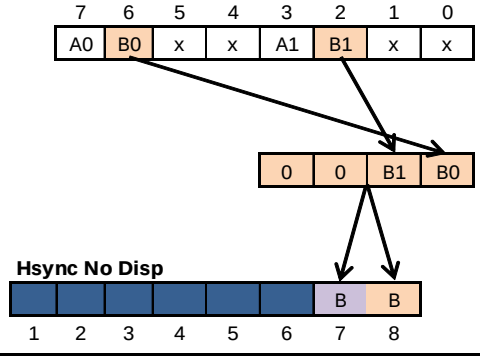
GA 40010 / GA 40007 / GA 40008

(3)



GA 40010 / GA 40007 / GA 40008

(4)



9.3.2.5 CRTC 4 : CUISINE DE PIXELS

Lorsque le PRE-ASIC bascule d'un mode graphique vers un autre mode, il est déjà en train de traiter les données d'un octet selon le mode précédent, et le résultat final obtenu traduit sa logique interne.

L'affichage cesse durant 2 µsec (32 Pixel-M2).
L'affichage est rétabli 1 Pixel-M2 avant que le PRE-ASIC change le mode graphique.

Le 36ème Pixel-M2 affiché dispose donc des caractéristiques de l'ancien mode, pour sa fraction correspondante à 1 Pixel-M2.

9.3.2.5.1 MODE 2 VERS MODE 0.1.2.3

Sur une ligne affichée en MODE 2, l'affichage des données reprend à partir du **3ème Pixel-M2 du 7ème octet de la VRAM**.

Le premier pixel affiché est en MODE 2 (PEN 1 ou 0).

Étant donné que les pixels MODE 2 sont « en avance » sur les pixels des autres modes, le PRE-ASIC va afficher **6 nouveaux Pixel-M2 après le 3ème**, sauf si le mode n'a pas changé.
Cela représente l'affichage de 9 pixels-M2 à partir d'un même octet.

Le tableau ci-après décrit l'interprétation par le PRE-ASIC de l'octet de VRAM traité lorsque l'affichage est réactivé, selon le nouveau mode graphique demandé.

Mode to Mode		VRAM Byte							
2	0	b7	b6	b5	b4	b3	b2	b1	b0
2	1	b7	b6	b5	b4	b3	b2	b1	b0
2	2	b7	b6	b5	b4	b3	b2	b1	b0
2	3	b7	b6	b5	b4	b3	b2	b1	b0

ASIC 40226: Displayed Pixels								
No Disp	b5	0.b2.b0.b4		0	b1	0	b3	
No Disp	b5	b0	b4	0	b3	0	b2	
No Disp	b5	b4	b3	b2	b1	b0		
No Disp	b5	0.0.b0.b4		0	0	0	b3	
1	2	3	4	5	6	7	8	9

9.3.2.5.2 MODE 0 vers MODE 0.1.2.3

Sur une ligne affichée en MODE 0, l'affichage des données reprend à partir **2ème Pixel-M2 du 7ème octet de la VRAM**.

Le premier Pixel-M2 affiché est le dernier des **4 pixel-M2 du pixel MODE 0** affiché dans sa couleur d'origine (soit un numéro de couleur compris entre 0 et 15). Autrement dit, seul ¼ du pixel MODE 0 est affiché avant que le PRE-ASIC affiche les pixels suivants.

Si le nouveau mode graphique requis est le MODE 2, alors le PRE-ASIC affiche 5 nouveaux Pixel-M2 après le 2ème, ce qui représente au total l'affichage de 7 pixels-M2 à partir du même octet.

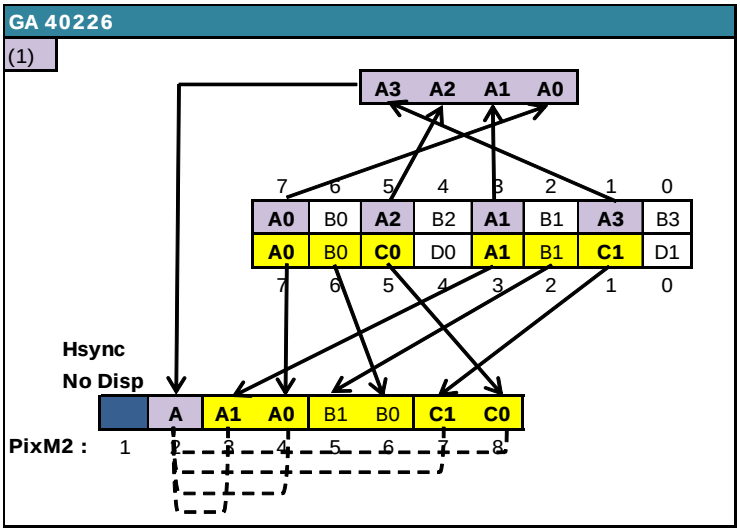
Si le nouveau mode requis est différent du MODE 2, alors le PRE-ASIC affiche 6 nouveaux Pixel-M2 après le 2ème.

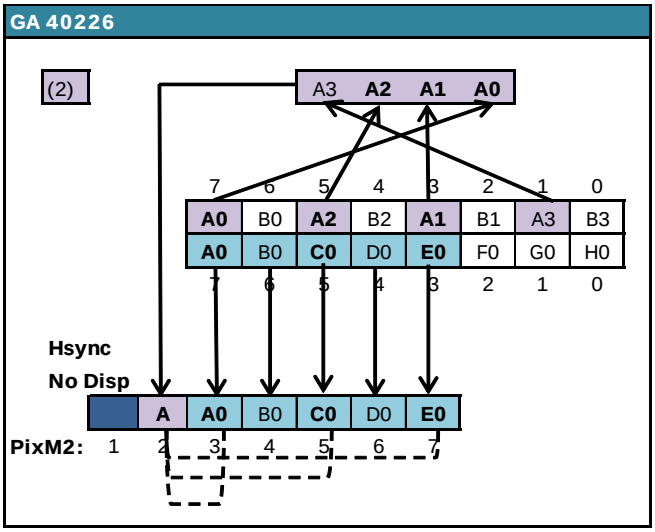
Enfin, si le nouveau mode requis est 1 ou 2, alors le PRE-ASIC, pour calculer les nouveaux numéros de couleur, « réutilise » des bits déjà utilisés pour calculer le numéro de couleur du pixel précédent.

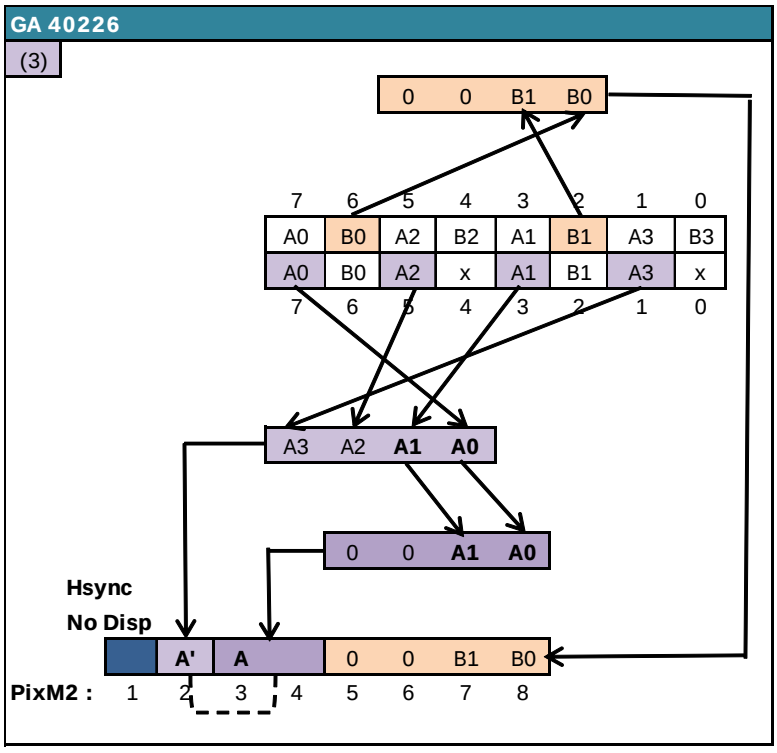
Le tableau ci-après décrit l'interprétation par le PRE-ASIC de l'octet de VRAM traité lorsque l'affichage est réactivé, selon le nouveau mode graphique demandé.

Mode to Mode		VRAM Byte							
0	0	b7	b6	b5	b4	b3	b2	b1	b0
0	1	b7	b6	b5	b4	b3	b2	b1	b0
0	2	b7	b6	b5	b4	b3	b2	b1	b0
0	3	b7	b6	b5	b4	b3	b2	b1	b0

ASIC 40226: Displayed Pixels							
b1.b5.b3.b7				b0	b4	b2	b6
(1)	b3	b7		b2	b6	b1	b5
(2)	b7	b6		b5	b4	b3	
(3)	(3)			0	0	b2	b6
1	2	3	4	5	6	7	8







9.3.2.5.3 MODE 1 vers MODE 0.1.2.3

Sur une ligne affichée en MODE 1, l'affichage des données reprend à partir **2ème Pixel-M2 du 7ème octet de la VRAM**.

Le premier Pixel-M2 affiché est le dernier des **2 pixel-M2 du pixel MODE 1** affiché dans sa couleur d'origine (soit un numéro de couleur compris entre 0 et 3). Autrement dit, seule la moitié du pixel MODE 1 est affiché avant que le PRE-ASIC affiche les pixels suivants.

Si le nouveau mode graphique requis est le MODE 2, alors le PRE-ASIC affiche 5 nouveaux Pixel-M2 après le 2ème, ce qui représente au total l'affichage de 7 pixels-M2 à partir du même octet.

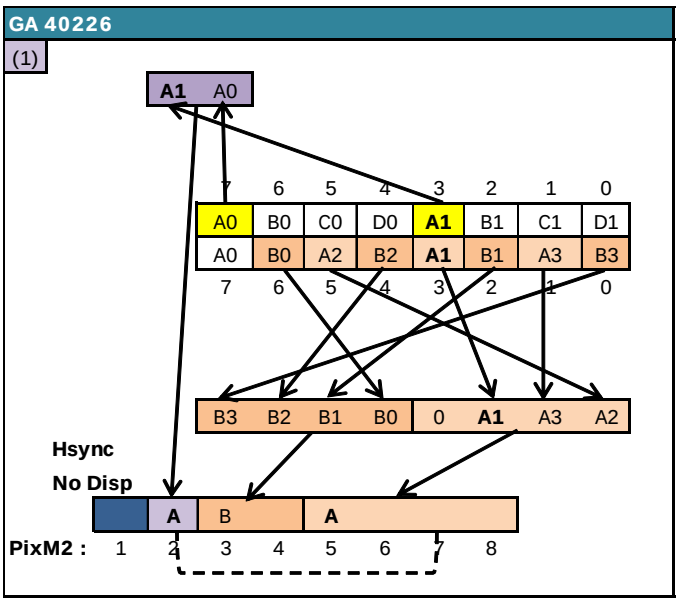
Si le nouveau mode requis est différent du MODE 2, alors le PRE-ASIC affiche 6 nouveaux Pixel-M2 après le 2ème.

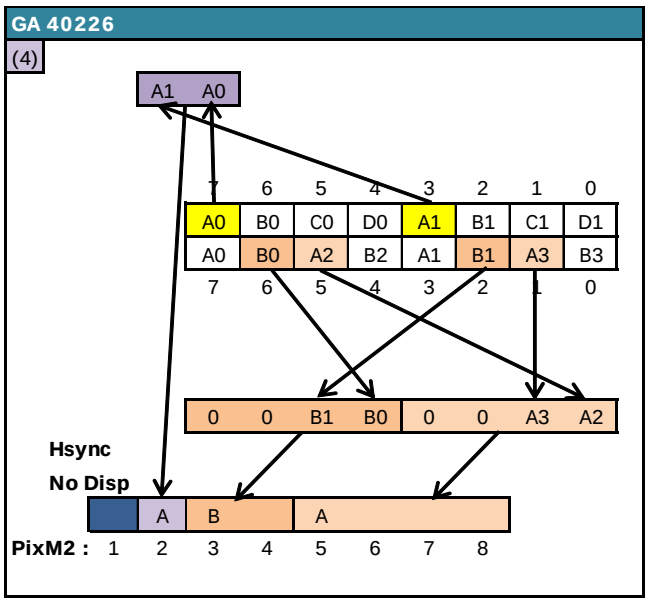
Enfin, si le nouveau mode requis est 1 ou 2, alors le PRE-ASIC, pour calculer les nouveaux numéros de couleur, « réutilise » des bits déjà utilisés pour calculer le numéro de couleur du pixel précédent.

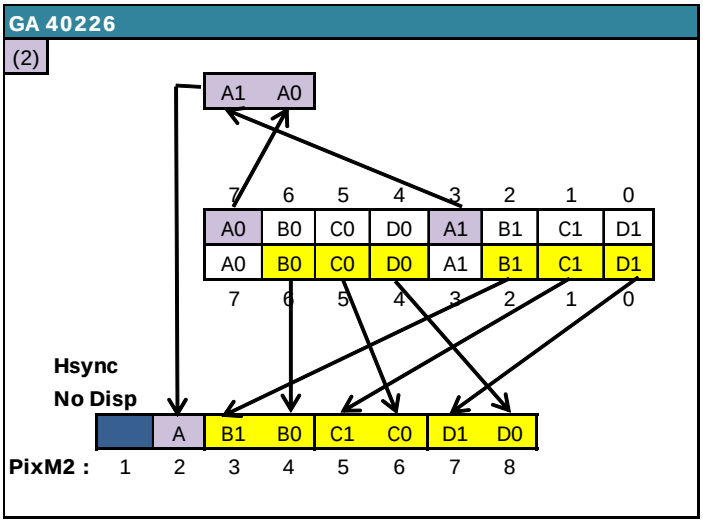
Le tableau ci-après décrit l'interprétation par le PRE-ASIC de l'octet de VRAM traité lorsque l'affichage est réactivé, selon le nouveau mode graphique demandé.

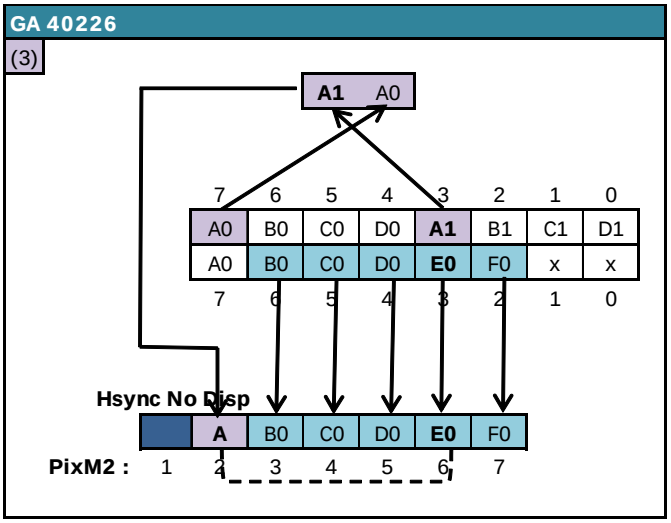
Mode to Mode		VRAM Byte							
1	0	b7	b6	b5	b4	b3	b2	b1	b0
1	1	b7	b6	b5	b4	b3	b2	b1	b0
1	2	b7	b6	b5	b4	b3	b2	b1	b0
1	3	b7	b6	b5	b4	b3	b2	b1	b0

Displayed Pixels							
(1)	b0.b4.b2.b6	0	b3	b1	b5		
(2)	b2	b6	b1	b5	b0	b4	
(3)	b6	b5	b4	b3	b2		
(4)	0.0.b2.b6	0	0	b1	b5		
1	2	3	4	5	6	7	8









9.3.2.5.4 MODE 3 vers MODE 0.1.2.3

Sur une ligne affichée en MODE 3, l'affichage des données reprend à partir **2ème Pixel-M2 du 7ème octet de la VRAM**.

Le premier Pixel-M2 affiché est le dernier des **4 pixel-M2 du pixel MODE 3** affiché dans sa couleur d'origine (soit un numéro de couleur compris entre 0 et 3). Autrement dit, seul le quart du pixel MODE 3 est affiché avant que le PRE-ASIC affiche les pixels suivants.

Si le nouveau mode graphique requis est le MODE 2, alors le PRE-ASIC affiche 5 nouveaux Pixel-M2 après le 2ème, ce qui représente au total l'affichage de 7 pixels-M2 à partir du même octet.

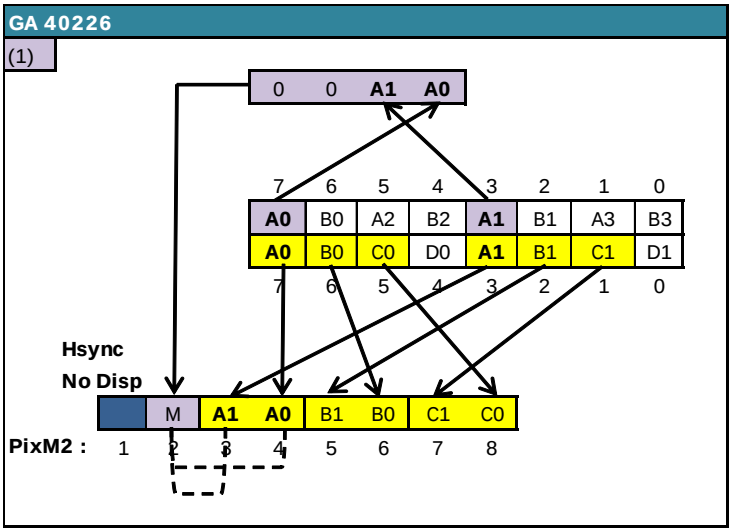
Si le nouveau mode requis est différent du MODE 2, alors le PRE-ASIC affiche 6 nouveaux Pixel-M2 après le 2ème.

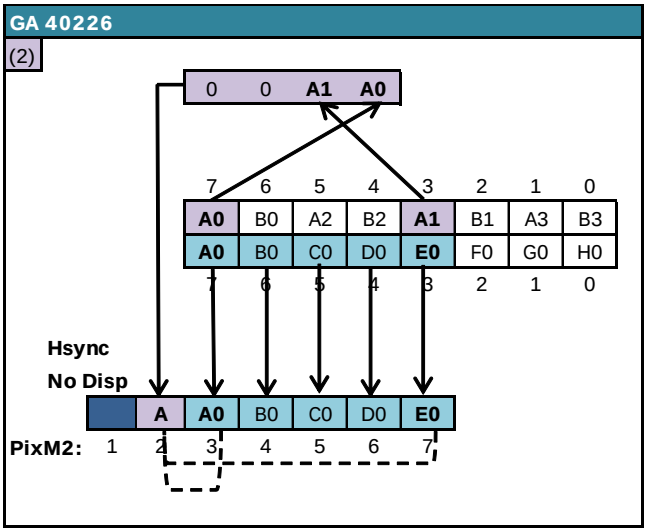
Enfin, si le nouveau mode requis est 1 ou 2, alors le PRE-ASIC, pour calculer les nouveaux numéros de couleur, « réutilise » des bits déjà utilisés pour calculer le numéro de couleur du pixel précédent.

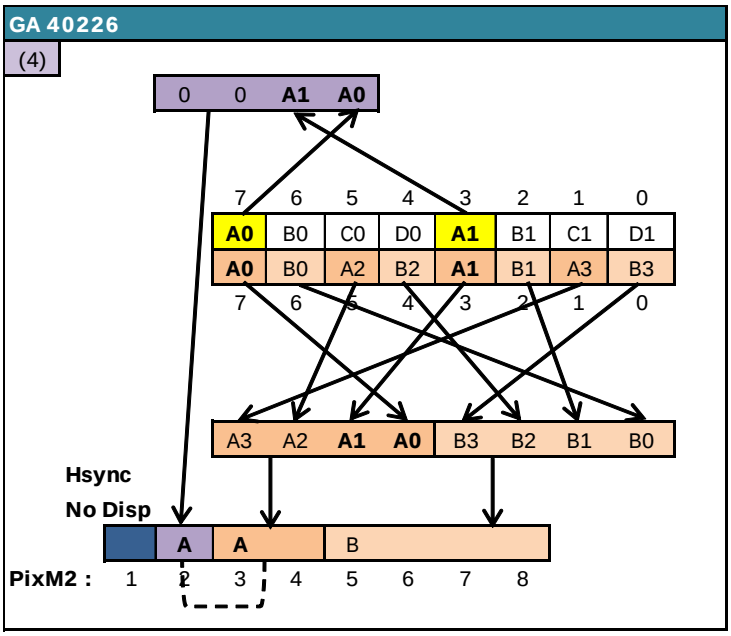
Le tableau ci-après décrit l'interprétation par le PRE-ASIC de l'octet de VRAM traité lorsque l'affichage est réactivé, selon le nouveau mode graphique demandé.

Mode to Mode		VRAM Byte							
3	0	b7	b6	b5	b4	b3	b2	b1	b0
3	1	b7	b6	b5	b4	b3	b2	b1	b0
3	2	b7	b6	b5	b4	b3	b2	b1	b0
3	3	b7	b6	b5	b4	b3	b2	b1	b0

Displayed Pixels							
(3)	b1.b5.b3.b7	b0	b4	b2	b6		
(1)	b3	b7	b2	b6	b1	b5	
(2)	b7	b6	b5	b4	b3		
0.0.b3.b7		0	0	b2	b6		
1	2	3	4	5	6	7	8







10 COMPTAGES : REGISTRE R9

10.1 GÉNÉRALITÉS

Le registre 9 permet de fixer le nombre de lignes verticales de chaque ligne-caractère.

En principe le compteur C9, qui détermine la ligne-raster de la ligne caractère, s'incrémente jusqu'à la valeur de R9. Lorsque le mode « Interlace » IVM est activé, l'algorithme de comptage de C9 est différent selon le type CRTC (voir chapitre 19.3).

Lorsqu'il revient à 0, C4 est incrémenté (et en "principe" doit revenir à 0 lorsque C4=R4).

Le compteur interne de "lignes" C9 est relié directement aux bits 11.12.13 du pointeur de VRAM.

C9 délimite, dans un espace de 16 ko, 8 tranches de 2 ko pour 8 "lignes" différentes possibles.

Etendu à l'espace de 64k adressable, cela représente 8 zones de 8 ko pour les 8 « lignes ».
Ce principe participe à la non linéarité verticale de l'affichage (effet "rideau" lorsque la mémoire est transférée sans découpage) bien connu de tout habitué du CPC.

La définition de R9 est de 5 bits. On dispose de « caractères verticaux » de 32 lignes maximum. Les bits 3 et 4 du compteur ne sont pas pris en compte pour le calcul du pointeur vidéo sur les valeurs de C9 qui dépassent 7.

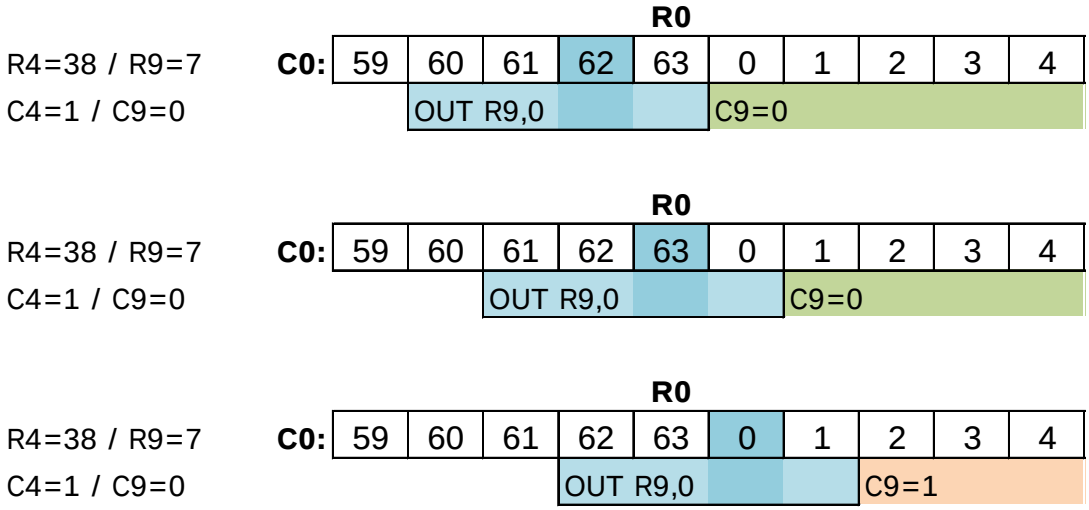
C'est néanmoins une information dont il faut tenir compte si on amène C9 à compter au-delà de 7. En effet, même si C9=8 donne l'affichage d'une ligne 0 (le bit 3 étant "ignoré"), cette valeur ne permet pas toujours de prendre en compte les opérations qui ont lieu lors du changement de ligne-caractère (C4).

10.2 DELAIS DE PRISE EN COMPTE

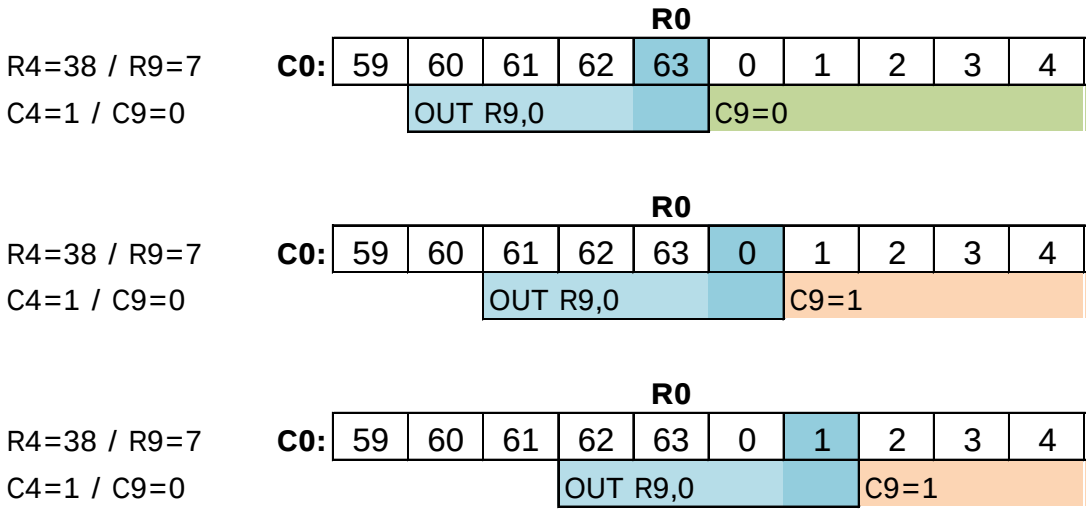
La mise à jour de R9 est prise en compte tant que C0<=R0.

Dans les schémas ci-dessous, R9 est mis à 0 alors qu'il était supérieur à 0 avant.

CRTC 0 , 1, 2



CRTC 3, 4



10.3 REGLES DE COMPTAGE

Si R9 est mis à jour avec 0 alors que C9 est supérieur à 0, par exemple, C9 va compter jusqu'à sa valeur maximale (31) avant de repasser à 0.

Il existe une exception à cette règle pour les CRTC 0 et 2, lorsque R9 est mis à jour sur la dernière ligne du dernier caractère du frame.

10.3.1 CRTC 0

On peut lire assez souvent que la modification de R9 sur les CRTC 0 (et 2) n'est pas prise en compte pour la ligne qui suit celle où la mise à jour a eu lieu car le compteur pour ce registre serait "bufférisé". La seule valeur qui est sauvegardée dans un buffer sur le CRTC est le pointeur vidéo car il est mis de côté à chaque début de ligne (voir chapitres 17 et 20.2).

Mais le CRTC ne "bufférise" pas ses compteurs. **Il teste simplement l'équivalence des compteurs avec ses registres à des instants très précis.**
En l'occurrence, **la valeur de C9 est comparée avec R9 lorsque C0=0** (ainsi que C4 avec R4) pour déterminer si la ligne était la dernière du frame. Cette comparaison est faite après que les compteurs C4 et C9 aient été mis à jour par rapport à la ligne précédente.

Cette comparaison a lieu uniquement pour décider si C4 sera remis à 0 ou si C4 sera incrémenté.
En effet, C4 est un compteur qui doit "**officiellement**" pouvoir dépasser R4, notamment en ajustement vertical.

Note : Les CRTC 3 et 4, pour lesquels C4 ne dépasse pas R4 en ajustement, "solidarisent" les lignes additionnelles avec C4=R4.

Si R9 ou R4 sont modifiés **après que C0 ait dépassé 0**, cela ne change rien. L'affaire est pliée.
Lorsque C4 est incrémenté, C9 passe obligatoirement à 0 (sauf lorsque R0=0).

En dehors de cette gestion particulière de traitement de C4 sur la dernière ligne d'un frame, **la mise à jour de R9 est prise en compte immédiatement**.

10.3.1.1 CAS GENERAL

Si R9 est modifié avec la valeur de C9, il va passer à 0 sur la ligne suivante.
Exemple : C9=0 et on positionne R9=0, qui valait 7 avant.
Exemple : C9=3 et on positionne R9=3, qui valait 7 avant.

Si R9 est modifié avec une valeur inférieure à C9, alors C9=C9+1 (débordement de C9)
Exemple : C9=3 et on positionne R9=1. C9 vaudra 4.
Si le registre C9 déborde, il va compter jusqu'à sa valeur maximale (31) avant de reboucler à 0, et ce jusqu'à atteindre la valeur de R9.

Si R9 est modifié avec une valeur supérieure à C9, alors C9=C9+1. C4 est inchangé.
Exemple : C9=0 et on positionne R9=7. C9 vaudra 1 sur la prochaine ligne.

10.3.1.2 EXCEPTION AU CAS GENERAL : DERNIERE LIGNE ECRAN

La mise à jour de R9 sur cette dernière ligne lorsque C0>1 n'a aucune incidence sur le prochain C9, car C4 sera forcé à 0 et donc C9 passera aussi à 0.

Si C4=R4 et C9=R9 lorsque C0=0, alors un état "**dernière ligne frame**" est levé.
Dans cette situation, C4 sera remis à 0 (hors ligne additionnelle), et C9 passera à 0 quelle que soit la valeur programmée dans R9 ensuite.

Si C4=R4 et C9<>R9 quand C0=0, alors C4 sera incrémenté quoi qu'il arrive.
Dans cette situation, si R9 est modifié sur cette ligne avec la valeur de C9, alors C9 passera à 0 mais C4 sera incrémenté.

L'état « dernière ligne » est testé au tout début de la gestion C0=0.

- Si une mise à jour de R9 a lieu lorsque C0=0, alors l'état dernière ligne ne sera pas levé.
- Si une mise à jour de R9 a lieu lorsque C0=1, alors la gestion additionnelle a lieu et R5 vient se substituer à R9 (C9+1=R5).

10.3.2 CRTC 1

10.3.2.1 CAS GENERAL

Si R9 est modifié avec la valeur courante de C9, alors sur la ligne suivante :

C9 passe à 0.

C4 est incrémenté, et passe à 0 si il valait R4, et dans ce cas l'offset est pris en compte.

Si R9 est modifié avec une valeur inférieure à C9

C9=C9+1 et C4 est inchangé (jusqu'à la fin du débordement de C9).

L'offset sera modifié uniquement si C4=C9=C0=0

Si R9 est modifié avec une valeur supérieure à C9, alors C9=C9+1. C4 est inchangé.

Exemple : C9=0 et on positionne R9=7. C9 vaudra 1 sur la prochaine ligne.

10.3.2.2 AUCUNE EXCEPTION

Tout n'est que pure logique, dans une indicible et insipide simplicité. :-)

10.3.3 CRTC 2

La gestion de C9/R9 est liée à celle de C4/R4.

Le chapitre 12.4.1 page 92 décrit la gestion de comptage et de mise à jour pour les deux registres.

En dehors des cas ou C4 et C9 repassent à 0 à cause d'un état de « **Dernière ligne** », similaire à celui existant sur CRTC 0, C9 compte jusqu'à R9, et repasse à 0 une fois que C9=R9. C4 est alors incrémenté (ou remis à 0) selon les situations.

Si le registre C9 déborde parce que R9 a été mis à jour avec une valeur inférieure à C9 (et en dehors d'une remise à 0 programmée de C9), ce dernier va compter jusqu'à sa valeur maximale (31) avant de reboucler à 0, et ce jusqu'à atteindre de nouveau la valeur de R9.

10.3.4 CRTC 3, 4

10.3.4.1 CAS GENERAL

Lorsque R9 est modifié, sa valeur est prise en compte immédiatement de la manière suivante.
Si R9 est modifié avec une valeur inférieure ou égale à C9, alors C9 passe à 0 sur la ligne suivante, et C4 passe à 0 (si C4=R4) sinon C4=C4+1.

Dis autrement, il est impossible de faire "déborder" C9 sur ces CRTC.

Ce n'est pas un simple test d'égalité qui a lieu mais une comparaison « plus complexe » permise par un ASIC :
Si C9 courant > R9 alors C9 suivant=0
Exemple : Si C9 valait 4, et que R9 est modifié avec 1 (alors qu'il valait 7 auparavant), alors C9 va passer à 0 (et C4=C4+1 ou 0 selon la valeur de C4 et R4)

Après C9=0, C9 sera incrémenté jusqu'à la nouvelle valeur de R9. 1 dans l'exemple.

Cette gestion permet parfois une double compatibilité avec les CRTC 0, 1 et 2.

En effet, si un programme modifie R9 dans le cadre d'une rupture ligne à ligne, une manière classique de procéder est de :

- Positionner R9=0 sur la dernière ligne d'un frame sur CRTC 0 ou 2, afin que la prochaine ligne C9=0 soit aussi considérée comme le dernier frame (si R4=0).
- Positionner R9=0 sur la première ligne d'un frame sur CRTC 1, afin que la prochaine ligne soit considérée comme le dernier frame (si R4=0).

Dans ces deux situations, le CRTC 3 et 4 remettront C9=0 sur la prochaine ligne.

Previous R9=7 C4=R4 (>0)			Event	CRTC 0 (HD6845SP) Result on next line			CRTC 1 (UM6845R) Result on next line			CRTC 2 (MC6845P) Result on next line			CRTC 3, 4 Result on next line		
Case	**R9**	**C9 cur**	**Upd R9**	**C9**	**C4**	**Offs Upd**	**C9**	**C4**	**Offs Upd**	**C9**	**C4**	**Offs Upd**	**C9**	**C4**	**Offs Upd**
1	7	0	0	0	**C4=C4+1 (*)**	No	0	0	Yes	0	**0 (**)**	**If C0<=R1**	0	0	Yes
2	7	1	0	C9=C9+1 (2)	unmodified	No	C9=C9+1 (2)	unmodified	Yes if C4=0	C9=C9+1 (2)	unmodified	No	0	0	Yes
3	7	2	0	C9=C9+1 (3)	unmodified	No	C9=C9+1 (3)	unmodified	Yes if C4=0	C9=C9+1 (3)	unmodified	No	0	0	Yes
4	7	3	0	C9=C9+1 (4)	unmodified	No	C9=C9+1 (4)	unmodified	Yes if C4=0	C9=C9+1 (4)	unmodified	No	0	0	Yes
5	7	4	0	C9=C9+1 (5)	unmodified	No	C9=C9+1 (5)	unmodified	Yes if C4=0	C9=C9+1 (5)	unmodified	No	0	0	Yes
6	7	5	0	C9=C9+1 (6)	unmodified	No	C9=C9+1 (6)	unmodified	Yes if C4=0	C9=C9+1 (6)	unmodified	No	0	0	Yes
7	7	6	0	C9=C9+1 (7)	unmodified	No	C9=C9+1 (7)	unmodified	Yes if C4=0	C9=C9+1 (7)	unmodified	No	0	0	Yes
8	7	7	0	0	0	Yes	C9=C9+1 (8)	unmodified	Yes if C4=0	0	**0**	**If C0<=R1**	0	0	Yes

(*) overflow **(**) if R9 is updated outside of hsync period**

Previous R9=7 C4<>R4			Event	CRTC 0 (HD) Result on next line			CRTC 1 Result on next line			CRTC 2 Result on next line			CRTC 3, 4 Result on next line		
Case	**R9**	**C9 cur**	**Upd R9**	**C9**	**C4**	**Offs Upd**	**C9**	**C4**	**Offs Upd**	**C9**	**C4**	**Offs Upd**	**C9**	**C4**	**Offs Upd**
1	7	0	0	0	**C4=C4+1**	No	0	C4=C4+1	No	0	C4=C4+1	No	0	C4=C4+1	No
2	7	1	0	C9=C9+1 (2)	unmodified	No	C9=C9+1 (2)	unmodified	Yes if C4=0	C9=C9+1 (2)	unmodified	No	0	C4=C4+1	No
3	7	2	0	C9=C9+1 (3)	unmodified	No	C9=C9+1 (3)	unmodified	Yes if C4=0	C9=C9+1 (3)	unmodified	No	0	C4=C4+1	No
4	7	3	0	C9=C9+1 (4)	unmodified	No	C9=C9+1 (4)	unmodified	Yes if C4=0	C9=C9+1 (4)	unmodified	No	0	C4=C4+1	No
5	7	4	0	C9=C9+1 (5)	unmodified	No	C9=C9+1 (5)	unmodified	Yes if C4=0	C9=C9+1 (5)	unmodified	No	0	C4=C4+1	No
6	7	5	0	C9=C9+1 (6)	unmodified	No	C9=C9+1 (6)	unmodified	Yes if C4=0	C9=C9+1 (6)	unmodified	No	0	C4=C4+1	No
7	7	6	0	C9=C9+1 (7)	unmodified	No	C9=C9+1 (7)	unmodified	Yes if C4=0	C9=C9+1 (7)	unmodified	No	0	C4=C4+1	No
8	7	7	0	C9=C9+1 (8)	unmodified	No	C9=C9+1 (8)	unmodified	Yes if C4=0	C9=C9+1 (8)	unmodified	No	0	C4=C4+1	No

Previous R9=7 C4=R4 (>0)			Event	CRTC 0 (HD) Result on next line			CRTC 1 Result on next line			CRTC 2 Result on next line			CRTC 3, 4 Result on next line		
Case	**R9**	**C9 cur**	**Upd R9**	**C9**	**C4**	**Offs Upd**	**C9**	**C4**	**Offs Upd**	**C9**	**C4**	**Offs Upd**	**C9**	**C4**	**Offs Upd**
1	7	0	0	0	**C4=C4+1**	No	0	0	Yes	0	**0 (**)**	**If C0<=R1**	0	0	Yes
2	7	1	1	0	C4=C4+1	No	0	0	Yes	0	**0 (**)**	**If C0<=R1**	0	0	Yes
3	7	2	2	0	C4=C4+1	No	0	0	Yes	0	**0 (**)**	**If C0<=R1**	0	0	Yes
4	7	3	3	0	C4=C4+1	No	0	0	Yes	0	**0 (**)**	**If C0<=R1**	0	0	Yes
5	7	4	4	0	C4=C4+1	No	0	0	Yes	0	**0 (**)**	**If C0<=R1**	0	0	Yes
6	7	5	5	0	C4=C4+1	No	0	0	Yes	0	**0 (**)**	**If C0<=R1**	0	0	Yes
7	7	6	6	0	C4=C4+1	No	0	0	Yes	0	**0 (**)**	**If C0<=R1**	0	0	Yes
8	7	7	7	0	0	Yes	0	0	Yes	0	**0 (**)**	**If C0<=R1**	0	0	Yes

(**) if R9 is updated outside of hsync period

Previous R9=7 C4<>R4			Event	CRTC 0 (HD) Result on next line			CRTC 1 Result on next line			CRTC 2 Result on next line			CRTC 3, 4 Result on next line		
Case	**R9**	**C9 cur**	**Upd R9**	**C9**	**C4**	**Offs Upd**	**C9**	**C4**	**Offs Upd**	**C9**	**C4**	**Offs Upd**	**C9**	**C4**	**Offs Upd**
1	7	0	0	0	**C4=C4+1**	No	0	C4=C4+1	No	0	C4=C4+1	No	0	C4=C4+1	No
2	7	1	1	0	C4=C4+1	No	0	C4=C4+1	No	0	C4=C4+1	No	0	C4=C4+1	No
3	7	2	2	0	C4=C4+1	No	0	C4=C4+1	No	0	C4=C4+1	No	0	C4=C4+1	No
4	7	3	3	0	C4=C4+1	No	0	C4=C4+1	No	0	C4=C4+1	No	0	C4=C4+1	No
5	7	4	4	0	C4=C4+1	No	0	C4=C4+1	No	0	C4=C4+1	No	0	C4=C4+1	No
6	7	5	5	0	C4=C4+1	No	0	C4=C4+1	No	0	C4=C4+1	No	0	C4=C4+1	No
7	7	6	6	0	C4=C4+1	No	0	C4=C4+1	No	0	C4=C4+1	No	0	C4=C4+1	No
8	7	7	7	0	C4=C4+1	No	0	C4=C4+1	No	0	C4=C4+1	No	0	C4=C4+1	No

Previous R9=7 C4=R4			Event	CRTC 0 (HD) Result on next line			CRTC 1 Result on next line			CRTC 2 Result on next line			CRTC 3, 4 Result on next line		
Case	**R9**	**C9 cur**	**Upd R9**	**C9**	**C4**	**Offs Upd**	**C9**	**C4**	**Offs Upd**	**C9**	**C4**	**Offs Upd**	**C9**	**C4**	**Offs Upd**
1	7	2	1	C9=C9+1 (3)	unmodified	No	C9=C9+1 (3)	unmodified	Yes if C4=0	C9=C9+1 (3)	unmodified	No	0	0	Yes

Previous R9=0 C4=R4=0			Event	CRTC 0 (HD) Result on next line			CRTC 1 Result on next line			CRTC 2 Result on next line			CRTC 3, 4 Result on next line		
Case	**R9**	**C9 cur**	**Upd R9**	**C9**	**C4**	**Offs Upd**	**C9**	**C4**	**Offs Upd**	**C9**	**C4**	**Offs Upd**	**C9**	**C4**	**Offs Upd**
1	0	0	0	0	0	Yes	0	0	Yes	0 (*)	C4=0 / C4+1	if C0<=R1/No	0	0	Yes
2	0	0	1	0	0	Yes	C9=C9+1 (1)	0	Yes (C4=0)	0 / 1 (*)	C4=0 / C4+1	if C0<=R1/No	C9=C9+1 (1)	0	No
3	0	0	2	0	0	Yes	C9=C9+1 (1)	0	Yes (C4=0)	0 / 1 (*)	C4=0 / C4+1	if C0<=R1/No	C9=C9+1 (1)	0	No
4	0	0	3	0	0	Yes	C9=C9+1 (1)	0	Yes (C4=0)	0 / 1 (*)	C4=0 / C4+1	if C0<=R1/No	C9=C9+1 (1)	0	No
5	0	0	4	0	0	Yes	C9=C9+1 (1)	0	Yes (C4=0)	0 / 1 (*)	C4=0 / C4+1	if C0<=R1/No	C9=C9+1 (1)	0	No
6	0	0	5	0	0	Yes	C9=C9+1 (1)	0	Yes (C4=0)	0 / 1 (*)	C4=0 / C4+1	if C0<=R1/No	C9=C9+1 (1)	0	No
7	0	0	6	0	0	Yes	C9=C9+1 (1)	0	Yes (C4=0)	0 / 1 (*)	C4=0 / C4+1	if C0<=R1/No	C9=C9+1 (1)	0	No
8	0	0	7	0	0	Yes	C9=C9+1 (1)	0	Yes (C4=0)	0 / 1 (*)	C4=0 / C4+1	if C0<=R1/No	C9=C9+1 (1)	0	No

(*) According authorization of last line in Hsync

11 COMPTAGES : REGISTRE R5

11.1 GÉNÉRALITÉS

Le registre R5 permet d'ajouter des lignes dites d'ajustement vertical en fin de frame.
L'objectif de ce registre est de permettre de compléter le nombre total de lignes verticales affichées lorsque (R4+1)x(R9+1) appliqué durant un frame ne correspond pas à 312 lignes (pour le standard Européen).

Ce registre contient un nombre de lignes sur 5 bits (0 à 31) qui correspond à l'ajustement maximum possible par rapport au nombre maximum de lignes possibles d'un caractère, fixé par R9. En effet, les CRTC 1, 2, 3, 4 ne respectent pas ce principe fonctionnel car l'ajustement n'est pas sensé contenir plusieurs caractères. Dis autrement, si R5/(R9+1)>1, R4 devrait être ajusté par le développeur.

Si R5 = 0 alors il n'y pas d'ajustement vertical spécifique.
Si R5 > 0 alors un ajustement a lieu avec création de R5 lignes complémentaires.

Si un des deux modes « Interlace » est programmé (R8=1 ou R8=3) alors une ligne d'ajustement complémentaire est ajoutée sur les frames pairs après les lignes éventuellement générées via R5.

L'implémentation de la fonction d'ajustement vertical est source de plusieurs différences entre les CRTC. Sur les CRTC 0, 3 et 4, il n'existe pas de compteur C5 spécifique et c'est C9 qui est utilisé pour la comparaison avec R5. Sur les CRTC 1 et 2, il existe un compteur C5 spécifique utilisé conjointement avec C9 afin de permettre une gestion de « caractères » au sein des lignes d'ajustement.

L'incrémentation de C4 suit différentes logiques :

- Sur le CRTC 0, C4 s'incrémente 1 seule fois et il vaut C4=R4+1 pour toutes les lignes additionnelles. C9 est comparé avec R5 et R9.
- Sur les CRTC 1 et 2, C4 s'incrémente quelle que soit la valeur de R4 à chaque fois que C9=R9, tant que C5+1 n'a pas atteint R5. Pour la première ligne additionnelle, C4=R4+1 si C9=R9.
- Sur les CRTC 3 et 4, C4 ne s'incrémente pas et vaut R4. C9 est uniquement comparé avec R5.

La gestion de R1 pour la mise à jour du pointeur vidéo continue à être assurée durant cette gestion de lignes additionnelles sur les CRTC 0, 1, 2 lorsque C9=R9.

Sur les CRTC 3 et 4, cette gestion de R1 pour la mise à jour du pointeur vidéo n'a lieu que la première fois où C9=R9 sur le caractère C4=R4 concerné par les lignes additionnelles.

R7 peut être modifié avec une des valeurs atteinte par C4 en ajustement vertical pour déclencher une VSYNC. Sur CRTC 2, si R7 est positionné avec C4 durant une HSYNC sur CRTC 2, une GHOST VSYNC débute. Sur CRTC 0 la VSYNC est bloquée si R7 est mis à jour avec C4 sur une position C0<2. Sur CRTC 3 et 4, la VSYNC ne débute que lorsque C4=R7 sur la position C0=C9=0.

Sur le CRTC 0, 1 et 2, si R9 est modifié avec une valeur différente de C9 sur la dernière ligne du frame avant l'ajustement vertical, alors la ligne additionnelle correspondra à C9+1 et C4 ne sera pas incrémenté.

Par exemple, si C4=R4=38, C9=R9=7 et R5=1 sur la dernière ligne, et que R9 est modifié avec 6 ou 8 sur cette ligne, alors la ligne additionnelle sera C4=38 et C9=8.

11.2 COMPTAGE EN AJUSTEMENT VERTICAL

11.2.1 GÉNÉRALITÉS

Les schémas suivants décrivent la diversité des méthodes de comptage (C4, C9, C5) et de mise à jour du pointeur vidéo durant cette gestion, en considérant que R5 et R9 ne sont pas modifiés durant l'ajustement. Les registres sont initialisés ainsi : R4=10, R5=16, **R9=3**, R1=40, R0=63.

CRTC 0			
C4	C9	PTR-VRAM	LINE
11	0	0	&0
11	1	0	&800
11	2	0	&1000
11	3	0	&1800
11	4	40	&2000
11	5	40	&2800
11	6	40	&3000
11	7	40	&3800
11	8	40	&0
11	9	40	&800
11	10	40	&1000
11	11	40	&1800
11	12	40	&2000
11	13	40	&2800
11	14	40	&3000
11	15	40	&3800

CRTC 1, 2				
C4	C9	C5	PTR-VRAM	LINE
11	0	0	0	&0
11	1	1	0	&800
11	2	2	0	&1000
11	3	3	0	&1800
12	0	4	40	&0
12	1	5	40	&800
12	2	6	40	&1000
12	3	7	40	&1800
13	0	8	80	&0
13	1	9	80	&800
13	2	10	80	&1000
13	3	11	80	&1800
14	0	12	120	&0
14	1	13	120	&800
14	2	14	120	&1000
14	3	15	120	&1800

CRTC 3, 4			
C4	C9	PTR-VRAM	LINE
10	0	0	&0
10	1	0	&800
10	2	0	&1000
10	3	0	&1800
10	4	0	&2000
10	5	0	&2800
10	6	0	&3000
10	7	0	&3800
10	8	0	&0
10	9	0	&800
10	10	0	&1000
10	11	0	&1800
10	12	0	&2000
10	13	0	&2800
10	14	0	&3000
10	15	0	&3800

11.2.2 CRTC 0

Sur CRTC 0, les ingénieurs HITACHI ont fait l'économie d'un compteur C5 afin d'utiliser C9 en lieu et place. En gestion additionnelle, C9 est comparé avec R9 et R5.

La nouvelle limite de C9 n'est plus R9 mais R5-1 (R5 doit être supérieur à 0 pour qu'il y ait au moins une ligne générée sauf dans le cadre des modes « Interlace »).
La gestion additionnelle inhibe alors simplement la remise à 0 de C9.

C9 continue à être comparé avec R9 **pour prendre en compte le pointeur vidéo (VMA'=VMA) lorsque C0=R1 et C9=R9**.

Ce n'est pas « pratique » pour gérer plusieurs caractères automatiquement durant l'ajustement, mais cela permet de passer sur une autre adresse sans que C9 soit égal à 0.

Ainsi, le schéma page suivante, qui reprend les données du premier schéma, montre l'impact d'une mise à jour R9 en cours d'ajustement pour faire évoluer le pointeur VMA'.

CRTC 0				
C4	C9	PTR-VRAM	LINE	
11	0	0	&0	
11	1	0	&800	
11	2	0	&1000	
11	**3**	0	&1800	**R9**
11	4	40	&2000	**OUT R9,10**
11	5	40	&2800	
11	6	40	&3000	
11	7	40	&3800	
11	8	40	&0	
11	9	40	&800	
11	**10**	40	&1000	**R9**
11	11	80	&1800	
11	12	80	&2000	
11	13	80	&2800	
11	14	80	&3000	
11	15	80	&3800	

Dans l'exemple :

- Lorsque C9 atteint R9 (=3), alors le pointeur vidéo est mis à jour avec celui qui a été mémorisé lorsque C0=R1 et C9=R9. (R1=40)
- R9 est modifié avec 10 alors que C9=4
- Lorsque C9 atteint R9 (=10), alors le pointeur est une nouvelle fois mis à jour, et passe donc à 80.
- Enfin, lorsque C9 atteint R5, l'ajustement vertical cesse.

11.2.3 CRTC 1, 2

Sur ces circuits, les compteurs C5 et C9 sont dissociés.
La gestion C9=R9 prend en compte le pointeur vidéo (VMA'=VMA) lorsque C0=R1.
C9 est remis à 0 lorsque C9=R9 et C4 est incrémenté.
La mise à jour de R9 est prise en compte pour le comptage courant de C9.

CRTC 1, 2					
C4	C9	C5	PTR-VRAM	LINE	
11	0	0	0	&0	
11	1	1	0	&800	
11	2	2	0	&1000	
11	3	3	0	&1800	**R9**
12	0	4	40	&0	**OUT R9,10**
12	1	5	40	&800	
12	2	6	40	&1000	
12	3	7	40	&1800	
12	4	8	40	&2000	
12	5	9	40	&2800	
12	6	10	40	&3000	
12	7	11	40	&3800	
12	8	12	40	&0	
12	9	13	40	&800	
12	10	14	40	&1000	**R9**
13	0	15	80	&0	

Si une ligne additionnelle est ajoutée via l'activation de la fonction « Interlace », cette logique n'est pas modifiée. Le comptage s'effectue comme si cette ligne avait été ajoutée à R5.

Exemple : Si on programme R4=37, R9=7 et R5=7 , alors C4 vaudra 38 durant les 8 premières lignes additionnelles. La ligne additionnelle « Interlace » fera alors partie du caractère C4=38.
Si R5 est programmé avec 8 dans cet exemple, alors C4 vaudra 38 sur les R5 lignes additionnelles, et C4 vaudra 39 sur la ligne additionnelle « Interlace ». (une fois sur 2 si la parité n'est pas figée, puisque la ligne additionnelle est ajoutée uniquement à la fin d'un frame pair)
Voir le chapitres 19.3 et 19.5 concernant les notions d'interlace et de parité.

11.2.4 CRTC 3, 4

Sur les CRTC 3 et 4, le compteur C4 n'est pas incrémenté lorsque la gestion additionnelle débute. C4 est égal à R4. Cependant, le pointeur vidéo est mis à jour avant le début des lignes additionnelles (VMA'=VMA) lorsque C0=R1.

La gestion additionnelle positionne ensuite C9=0 et compare C9 avec R5 pour désactiver cette gestion, sans mettre à jour le pointeur vidéo.

Ceci a pour effet de "solidariser" le dernier caractère avec le caractère généré en ajustement vertical, et la ligne « Interlace » si elle a été programmée.

Dans l'exemple précédent, le caractère C4=10 contient 16 lignes de plus.

11.3 MISE À JOUR DE R5 DURANT UN AJUSTEMENT

Quel que soit le CRTC, si R5 est modifié avec C5+1 (ou C9+1 sur CRTC 0, 3, 4) sur la ligne C5 (ou C9), alors l'ajustement vertical est "stoppé".

Dans cette situation, C4=C9=0 pour la prochaine ligne (quelle que soit la valeur courante de C9), sauf si les conditions sont présentes pour qu'une ligne « Interlace » soit générée.

L'état « dernière ligne » est alors traité correctement et conduit à la remise à 0 de C4.

11.3.1 CRTC 0, 2

Lorsque le numéro de la prochaine ligne additionnelle (C5+1 sur CRTC 2, C9+1 sur CRTC 0) atteint R5 alors la gestion additionnelle R5 prend fin.

La ligne additionnelle « Interlace » (R8) est gérée si les conditions pour sa génération sont respectées.

Si R5 est modifié avec une valeur inférieure à C5+1/C9+1, alors le compteur déborde et continue à compter jusqu'à repasser à 0 pour atteindre la nouvelle valeur de R5.

Attention cependant car sur CRTC 0, la mise à jour de R5 est prise en compte uniquement lorsque C0<3. Autrement dit si R5 est modifié lorsque C0>2 sur la dernière ligne (C9+1=R5), alors la valeur de R5 n'est pas prise en compte (C4 et C9 repasseront alors à 0).

Si aucune ligne « Interlace » n'était programmée, alors C4 et C9 repassent à 0.

11.3.2 CRTC 1

Lorsque le numéro de la prochaine ligne additionnelle (C5+1) atteint R5 alors la gestion additionnelle R5 prend fin.
La ligne additionnelle « Interlace » (R8) est gérée si les conditions pour sa génération sont respectées.
Si R5 est modifié avec une valeur inférieure à C5+1, alors le compteur C5 déborde et continue à compter jusqu'à repasser à 0 pour atteindre la nouvelle valeur de R5.

Cependant, **si la nouvelle valeur de R5 est fixée à 0**, cela provoque un bug qui désactive (sournoisement) la remise à 0 de C4 pour le nouveau frame. À l'issue des lignes additionnelles, C4 s'incrémente et déborde.

11.3.3 CRTC 3,4

Lorsque le numéro de la prochaine ligne additionnelle (C9+1) atteint R5, alors la gestion additionnelle R5 prend fin.
La ligne additionnelle « interlace » (R8) est gérée si les conditions pour sa génération sont respectées.

Si R5 est modifié avec une valeur inférieure à C9+1, alors la ligne est considérée comme la dernière et la gestion additionnelle cesse.

Que ce soit avec R5 ou R9, il est impossible de faire déborder C9.

11.4 MISE À JOUR DE R5 AVANT UN AJUSTEMENT

11.4.1 CRTC 1,2,3,4

La gestion de R5 est prise en compte sur chaque position de C0.
Sur CRTC 1, la mise à jour de R5 sur C0=R0 déclenche un bug, décrit dans le chapitre suivant.

11.4.2 CRTC 0

La mise à jour de R5>0 après la position C0>2 sur la dernière ligne du frame n'est pas prise en compte car son évaluation est terminée. Voir chapitre 13.2

Sauf si une ligne « Interlace » a été définie, la prochaine ligne affichée sera C4=C9=0 (respect de la condition « **Dernière Ligne** »)

11.5 RUPTURE FOR DUMMIES (R.F.D.) SUR CRTC 1

Sur le CRTC 1, il existe un bug très intéressant lorsque R5 est mis à jour avec une valeur différente de 0 sur la position C0=R0 de certains C9 lorsque R5 valait 0.
Je vais nommer **RFD** la technique de rupture qui découle de ce bug.

L'activation d'une **RFD** positionne deux états.

- Le premier correspond à l'état qui définit la source de mise à jour de VMA (VMA' ou R12/R13) lorsque C0=0.
- Le second état active la gestion de la parité dans le test de C9/R9 réalisé en IVM lorsque C0 atteint R1 (et qui permet le positionnement de l'état de mise à jour de VMA lorsque la condition C9=R9 sur C0=R1 est vérifiée).

La valeur de R5 a peu d'importance, **sauf sur certains CRTC 1,** pour la valeur **#10**.
Ce CRTC sera identifié comme le **CRTC 1-B** en attendant de savoir si la différence vient réellement du CRTC. L'effet obtenu par cette valeur sera identifié comme **RFD#10**.
Le **CRTC 1-A** est défini comme le CRTC 1 qui ne gère pas la **RFD#10**.

La gestion additionnelle continue d'être gérée normalement si R5 n'est pas remis à 0 après le déclenchement de la RFD. Si une gestion additionnelle n'est pas souhaitée pour le frame, la remise à 0 de R5 peut se faire sur n'importe quelle position de C0 après l'activation de la RFD.
Il est donc possible de déclencher une RFD via un « OUT R5,1 », suivi immédiatement d'un « OUT R5,0 » si c'est uniquement l'effet RFD qui est recherché. Si une gestion additionnelle et une RFD sont souhaités, il est possible de le faire en mettant R5 à jour une seule fois sur C0=R0 avec le nombre de lignes additionnelles souhaité.

Le traitement courant de C4 par rapport à C9 n'est pas affecté.

La RFD démontre qu'il existe un état qui permet au CRTC 1 d'accepter la prise en compte de R12/R13 en début de ligne sans considération de la valeur de C4.

Pour rappel, sur le premier caractère d'un frame, VMA est affecté avec R12/R13 et est chargé avec VMA' sur les autres caractères. VMA' est lui-même chargé avec VMA lorsque C9 atteint R9 et C0 atteint R1 (afin de prendre en compte l'avancée de l'offset sur la prochaine ligne par rapport au BORDER). La mise à jour de VMA avec R12/R13 ou VMA' a lieu à partir de C0=0.

L'état de mise à jour de VMA via R12/R13 est habituellement vrai lorsque C4=0, et devient faux ensuite (VMA est alors rechargé avec VMA'). Cependant, si pour une raison donnée, la condition permettant de tester que le BORDER a été atteint sur la dernière ligne du caractère **n'est pas remplie**, alors l'état de mise à jour de VMA via R12/R13 reste vrai quelle que soit la valeur de C4.

Il est toutefois important de noter que si c'est la condition C0=R1 qui n'est pas atteinte (car R1>R0), alors la condition C9=R9 suffit à désactiver la mise à jour de VMA avec R12/R13. Il ne suffit donc pas que R1>R0 pour pouvoir modifier l'offset sur chaque ligne (on s'en serait aperçu).

La RFD positionne un autre état de mode « Interlace » IVM qui fait intervenir la parité du frame dans l'équivalence C9=R9. Ainsi, la mise à jour de VMA via R12/R13 est activée par la RFD et cet état perdure tant que la seule équivalence vraie est C0=R1. La RFD ne modifie cependant pas le mode de comptage associé au mode IVM.

Autrement dit cet état reste actif tant que la parité du frame combinée au mode de calcul de C9 entre en jeu sur le traitement comparatif de C9. Il est ainsi possible de se retrouver avec une situation équivalente à R1>R0 (voir chapitre 17.4.2) alors que R1<R0.

La condition C9=R9 « hors parité » continue en effet d'être traitée normalement pour le calcul de C4 et la remise à 0 de C9. La **RFD** n'affecte pas le mode de comptage de C9 et de C4, mais détermine uniquement les valeurs comparatives de C9 lorsque C0=R1 pour activer l'affectation de VMA avec VMA' (au lieu de VMA=R12/R13).

11.5.1 RFD ET PARITÉ

L'activation d'une **RFD** implique un test de R9 réalisé avec la parité courante du frame.

Sur le **premier frame (cas 1)**, VMA' n'est plus mis à jour lorsque C0 atteint R1 car le test C9=R9 est alors défaillant à cause de la parité impaire qui entre en jeu, et les caractères se répètent. C4 continue d'être géré chaque fois que C9 atteint R9. L'état de mise à jour de VMA est bloqué sur R12/R13 et il est possible de modifier l'adresse sur chaque ligne de chaque caractère, « comme si » C4 valait toujours 0.

Sur le **second frame (cas 2)**, VMA' est mis à jour lorsque C0 atteint R1 et C9=R9. La parité ne corrompt plus le test C9/R9 et les caractères ne se répètent donc pas car VMA est mis à jour avec VMA'. Sur ce frame un changement d'adresse avec R12/R13 n'est plus pris en compte dès que VMA a été mis à jour avec VMA' (donc lorsque C9=R9). Une **RFD** déclenchée sur la dernière ligne C9=R9 désactive l'état permettant la mise à jour de VMA avec R12/R13. Cependant, la **RFD#10 des CRTC 1-B** permet toujours, lorsque C9=R9, de désactiver la prise en compte de la parité dans le test C9=R9 (voir chapitre suivant).

C4 s'incrémente dans les deux situations et le frame reste parfaitement synchronisé via C4=R7.

La parité alterne une fois à chaque fois que le frame débute (C4=C9=C0=0 et R9 impair). Si elle n'a jamais été **fixée**, cela provoque un effet stroboscopique entre deux frames (sauf à positionner, au bon moment, R1>R0 pour que toutes les lignes se répètent également sur les deux frames). Mais nous allons voir qu'un tel subterfuge est inutile car il existe quelque chose de bien plus simple.

11.5.2 IVM ON/OFF

Comme on vient de le voir, la répétition du premier caractère sur le frame défini comme le cas 1 est la conséquence d'un défaut du test d'équivalence C9=R9 permettant l'affectation de VMA' (et qui verrouille l'état de mise à jour de VMA). Ce test de C9 fait intervenir la parité du frame. Selon cette parité, le test permettant l'affectation VMA' est affecté entre chaque frame.

Il est cependant possible de **fixer la parité d'un frame**, sachant que **cette parité bascule de pair à impair (et vice-versa) à chaque nouveau frame (et dans quelques autres cas détaillés dans le chapitre 19.5.3)**. Il suffit par exemple **d'activer et désactiver le mode IVM avec un R9 impair** pour fixer une **parité paire** (OUT R8,3 suivi de OUT R8,0). Je vais nommer cette action « **IVM ON/OFF** ». Attention toutefois à effectuer ces mises à jour sur un C9 pair car le bit 0 de C9 peut passer à 0 sur la ligne et donc corrompre le comptage si C9 était impair.

Si un « **IVM ON/OFF** » a lieu **avant une RFD**, alors tous les frames contiendront des lignes pour lesquelles VMA' n'est plus mis à jour avec VMA (cas 1 évoqué dans le chapitre précédent). Autrement dit, **il est alors possible de modifier l'offset sur chaque ligne de l'intégralité du frame sans autre formalité complémentaire**.
Si un « **IVM ON/OFF** » a lieu **après la RFD**, alors tous les frames contiendront des lignes pour lesquelles VMA' est mis à jour à chaque fois que C9=R9 (cas 2 évoqué dans le chapitre précédent). Autrement dit, il est possible de modifier l'offset tant que C9<>R9 à partir de la ligne où la **RFD** est générée. Le « **IVM ON/OFF** » ayant eu lieu après la mise à jour de R5, **c'est sur le nouveau frame que la parité paire va basculer pour devenir impaire et provoquer cette gestion à partir du second frame et les suivants**.

Sur les **CRTC 1-B**, la valeur #10 dans R5 **désactive la gestion de la parité dans le test C9=R9**, alors que les autres valeurs **activent cette gestion de parité (y compris la valeur #10 sur les CRTC 1-A)**.
La marque et le modèle des **CRTC 1-B** ne diffèrent pas des autres CRTC 1 (UM6845R).
Le numéro de lot semble ne pas être en cause : La RFD#10 fonctionne par exemple sur un 6128 UM6845R-8804T, mais pas sur un 6128 avec UM6845R-8802T.
Sur 7 machines testées, 3 disposaient de cette capacité complémentaire.
Si vous souhaitez identifier cette capacité sur un CRTC 1, vous pouvez utiliser SHAKER 2.1 en exécutant le module SHAKE21B.BIN, test « O ».

On peut définir que la RFD :

- Active un état autorisant la mise jour de VMA avec R12/R13 lorsque C0=0 (cet état est désactivé lorsque le test C9=R9 sur C0=R1 est vrai).
- Active (ou désactive sur le **CRTC 1-B** avec la RFD#10) un état autorisant la prise en compte de la parité dans le test C9=R9 réalisé lorsque C0=R1.

Lorsque l'état IVM qui autorise la prise en compte de la parité dans le test C9=R9 devient actif avec une RFD, il n'est plus possible de le désactiver tant que le frame n'est pas terminé.
Ainsi :

- Si une RFD active cet état, une RFD#10 ne peut plus le désactiver.
- Si une RFD#10 (**CRTC 1-B**) désactive cet état, une RFD « hors #10 » permet de le réactiver.

Remarque 1: Le mode « Interlace Sync » (IS ON/OFF) ne permet pas (à ma connaissance) de fixer la parité du frame. L'alternance de traitement entre chaque frame n'est pas impactée si R8 bascule entre la valeur 1 et 0 (ou 2).

11.5.3 R.F.D. EN BREF

Recette pour changer l'adresse sur un caractère sans avoir à se préoccuper de R4 ou R7.

- Sur C0=R0, faire passer R5 de 0 à 1 sur toute ligne différente de C9=R9 pour un C4 où l'offset doit pouvoir être modifié.
  - OUT R5,1+OUT R5,0
- Modifier R12/R13 sur la ligne qui précède la mise à jour de R5.
- Faire passer R8 de 3 à 0 une fois sur une ligne C9 paire.
  - OUT R8,3+OUT R8,0

Recette pour changer l'adresse sur chaque ligne.

- Attendre d'être sur un nouveau frame (C4=C9=0)
- Faire passer R8 de 3 à 0 une fois sur une ligne C9 paire
  - OUT R8,3+OUT R8,0
- Sur C0=R0, faire passer R5 de 0 à 1 sur n'importe quelle ligne 1 fois pour le frame.
  - OUT R5,1+OUT R5,0
- Modifier R12/R13 sur la ligne qui précède celle où l'adresse doit être modifiée.

Cette méthode ne permet pas de faire une RLAL « conventionnelle », dans la mesure où C9 participe à la construction du pointeur vidéo. C9 est à 0 en RLAL. Il est toujours possible de dupliquer la ram vidéo des différents C9, mais cela implique une consommation sérieuse de la ram en fonction de la valeur de R9 programmée pour le frame.

11.5.4 R.F.D. ET AUTRES CRTCs

Le guide technique HITACHI (CRTC 0) contient, page 120, un tableau faisant état des anomalies capables de survenir lorsque certains registres sont modifiés durant l'affichage. Concernant R5, il est notamment indiqué que si une mise à jour de R5 a lieu sur C0=R0, alors il y a « certains cas » où R5 n'est pas vraiment pris en compte…

Les contre-indications de ce tableau sont un vrai terrain de jeu !

11.6 R6 ET AJUSTEMENT VERTICAL

Lorsque C4 atteint R6, l'affichage des données est stoppé.
R6 doit être positionné en fonction des valeurs atteintes par C4 pour afficher les caractères correspondants générés pendant l'ajustement.

Sur le CRTC 1, il existe une gestion particulière de R6 lorsqu'il vaut 0, qui permet d'activer le BORDER de manière non définitive.

Ce traitement particulier est également géré par le CRTC 1 en ajustement vertical.
La technique qui utilise cette gestion particulière est appelée « Split-Border ».

Cette technique est possible sur CRTC 0, 3 et 4 en utilisant une fonction de R8 (voir chapitre 19.2.1, page 170).

11.7 AJUSTEMENT DURANT L'INTERLACE

En mode « Interlace », R5 lignes sont ajoutées au frame.

Sur CRTC 2, c'est assez logique, car ce CRTC ne requière pas, comme pour les autres CRTC, une mise à jour de R4 pour doubler le nombre de caractères affichés (puisque chaque caractère contient 2 fois moins de lignes en mode « Interlace ») et une adaptation des registres associés à C4, comme R6 et R7.

Sur les CRTC 0, 1, 3, et 4, R5 lignes sont ajoutées sur chaque frame.
Si par exemple, on a 38 caractères de 8 lignes et 8 lignes additionnelles, il faut programmer R4=37 et R5=8 (38 x 8 + 8=312). Si on passe en « Interlace » IVM (R8=3), pour conserver le même nombre de lignes, il faut doubler le nombre de caractères affichés. R4 doit donc être égal à 75 (38x2 – 1).

Cependant R5 doit rester à 8 : 76 x 4 + 8 = 312.

Dans tous les cas, les compteurs C9/C4 continuent d'évoluer selon la logique de traitement « Interlace » associées aux lignes selon le CRTC (voir chapitre 11.2).

Par exemple, sur CRTC 1 dans l'exemple précédent, C4 va s'incrémenter 2 fois durant la période d'ajustement R5 car pour chaque C4, il y aura 4 C9.

11.8 LIGNE D'AJUSTEMENT INTERLACE

En mode « Interlace », une gestion spécifique d'ajustement vertical est réalisée, avec l'ajout d'une ligne un frame sur deux. Cet ajustement intervient dans des conditions bien précises, qui dépendent de l'état de R8 mais également de la parité courante du frame.
Voir chapitre 19.3, page 175.

Cet ajustement est indépendant de celui effectué via R5. Lorsque la condition d'ajustement est remplie, la ligne « Interlace » est ajoutée **après** les lignes éventuellement programmées dans R5.

La condition d'ajustement (mode « Interlace » (IVM/Non IVM) activé et frame pair) est évaluée sur la dernière ligne d'un frame, lorsque C0=R0, et uniquement si R8 contient la bonne valeur sur la dernière ligne. Cette dernière ligne peut être une des lignes d'ajustement affichée via R5.

Il est donc possible de mettre à jour R8 sur une des lignes affichées via R5 pour activer ou désactiver le traitement de la ligne additionnelle « Interlace ».

Sur CRTC 2, si le mode « Interlace » est désactivé (R8=0) pendant que la ligne « Interlace » est affichée, alors la condition de « **Dernière Ligne** » est annulée. La ligne courante n'est plus considérée comme une ligne « Interlace ». C9 continue alors de compter jusqu'à R9, et C4 est incrémenté si il est différent de R4. Il est parfaitement possible de reprogrammer R4 avec C4 (égal à l'ancien R4+1) afin de réactiver l'état « **Dernière Ligne** » lorsque C9 atteindra R9.

12 COMPTAGES : REGISTRE R4

12.1 GÉNÉRALITÉS

Le registre R4 du CRTC permet de définir le nombre de « ligne-caractères » à afficher.
Une ligne-caractère est composée de plusieurs lignes-raster, dont le nombre est fixé par le registre R9. Voir chapitre 9.3.2, page 51.

La ligne et le caractère sont numérotés à partir de 0.
Le nombre à programmer représente une valeur à atteindre.

Lorsque toutes les lignes d'un caractère sont affichées (C9=R9), alors C4 est incrémenté (ce dernier repasse à 0 si C4 valait R4 avant incrémentation).

La valeur de C4 est comparée avec R7 pour déclencher une VSYNC, et comparée avec R6 pour déclencher du BORDER. À noter toutefois que le mode « Interlace » du CRTC diffère le déclenchement de la VSYNC.

Lorsque C4=R4 et que toutes les lignes du dernier caractère sont affichées, une gestion de ligne(s) additionnelle(s) peut éventuellement débuter si R5 > 0 ou que le mode « Interlace » est activé.

À noter qu'à l'exception des CRTC 3 et 4, les lignes additionnelles via R5 continuent à incrémenter C4 (qui dépasse donc R4 dans cette situation).

À noter également que cette gestion de ligne(s) additionnelle(s) est obligatoirement déclenchée lorsque R0<2 sur CRTC 0.

Lorsque C4 passe à 0 (et tant qu'il est à 0 sur CRTC 1), VMA est mis à jour avec le contenu de R12/R13.

Sur CRTC 2 cependant, VMA est mis à jour avec VMA', qui est lui-même mis à jour avec R12/R13 lorsque C0=R1 de la dernière ligne (lorsque C9=R9).

12.2 CRTC 0

Lorsque C0=0, le CRTC évalue si C9=R9 et C4=R4 pour **déterminer si il est sur la dernière ligne du frame**. Il ne refait plus ce test sur les autres valeurs de C0.

Il n'est pas forcément nécessaire d'anticiper la programmation de R4 sur la ligne précédente pour que C4=R4 au début de la ligne suivante. En positionnant un OUT R4 sur C0vs=#3E (si R0=#3F), R4 est mis à jour lorsque C0=0 pour satisfaire le test de "**dernière ligne**". Ceci permet une compatibilité avec les autres CRTC. À noter que cela est aussi vrai pour R9 si la condition C9=R9 est la seule qui manque pour le test « **dernière ligne** » sur la nouvelle ligne.

Lorsque le CRTC a déterminé qu'il est sur la dernière ligne du frame, C4 passera à 0 quoi qu'il arrive (ainsi que C9). Si R4 et/ou R9 sont modifiés en cours de ligne lorsque C0>1 alors que c'est **la dernière ligne, cela ne change rien pour C4 qui va donc passer à 0**.

Lorsque Le CRTC a déterminé qu'il n'était pas sur la dernière ligne (et même si R9 a été modifié avec C9), C4 va être incrémenté quoi qu'il arrive.
Ainsi, si R4 et/ou R9 sont modifiés en cours de ligne alors que **ce n'est pas la dernière ligne, cela ne change rien pour C4 qui sera incrémenté** (lorsque C9=R9).
Ce sont les valeurs incrémentées (ou remises à 0) des compteurs C4 et C9 qui sont testées sur C0=0.

Lorsqu'on est sur la dernière ligne du frame :

- Modifier R4 avec C4 n'empêche pas C4 de s'incrémenter si C9=R9 (si l'objectif était de le faire passer à 0).
- Modifier R4 avec 0 alors que C4 valait R4 n'empêche pas C4 de revenir à 0 (si l'objectif était de le faire déborder).

Le test de "dernière ligne" est fait en début de ligne, aussi il est nécessaire d'anticiper la mise à jour de R4 et R9 en fonction de ce que l'on souhaite obtenir dans C4.

À noter que C9 peut uniquement déborder dans un contexte particulier (lorsque R0=0).

Le compteur C4 peut s'incrémenter au-delà de la valeur fixée dans R4 dans plusieurs cas de figure :

- Si R4 est mis à jour avec une valeur inférieure à C4 ou qu'il a été incrémenté dans les conditions décrites ci-dessus, alors C4 va "déborder". Autrement dit, il va s'incrémenter jusqu'à sa valeur maximale (127) avant de reboucler (si une nouvelle mise à jour de R4 n'intervient pas avant).
- C4 peut également dépasser la valeur de R4 lorsqu'une gestion de lignes verticales additionnelles est demandée. Cet évènement peut survenir lorsque R5>0 (ou R5=0 avec R0<2), ou si le mode « Interlace » est activé sur des frames pairs (voir chapitre 19.3, page 175).
  À noter que si C4 dépasse R4 sur au moins un de ces évènements, **il va revenir à 0 une fois la gestion additionnelle terminée**.

12.2.1 CAS PRATIQUE : RUPTURE LIGNE À LIGNE (R.L.A.L.)

L'objectif de cette technique est d'obtenir des lignes consécutives pour lesquelles C9=C4=0, afin de pouvoir en modifier l'adresse via R12 et/ou R13.
Pour les deux exemples suivants, on considère que les registres R4 et R9 sont supérieurs à 0.

À partir de la première ligne d'un frame :
Si R9 et/ou R4 sont positionnés à 0 lorsque C9=C4=0, cette ligne ne sera pas considérée comme la dernière du frame (le test a eu lieu lorsque C0=0).
Sur cette deuxième ligne, C4 va alors être incrémenté (C4=1). Et C9 va passer à 0.
Au début de cette deuxième ligne, C4(=1)<>R4(=0). C9=R9=0.
Le CRTC va donc incrémenter C4 quelle que soit la valeur qu'on programme dans R4, ...

Si, sur la ligne 1, on programme R4=1 et R9=0 (au lieu de R4=0 et R9=0), les choses vont mieux se passer. Cette ligne n'ayant pas été considérée comme la dernière, C4 va s'incrémenter sur la deuxième ligne (C4=1)(et C9 sera égal à 0)

Cependant, le test qui a eu lieu en début de ligne 2 indique que c'était la dernière (C4=R4=1 et C9=R9=0). C4 va donc passer à 0 sur la troisième ligne.
Mais si on veut que cette troisième ligne et toutes les autres soient aussi considérées comme les dernières, il faut....
.... modifier R4 avec 0 sur la deuxième ligne. Et le tour est joué.

À partir de la dernière ligne d'un frame (C9=R9 et C4=R4):
Si R9 et R4 sont mis à jour avec 0 lorsque C9=R9 et C4=R4, cette ligne est déjà considérée comme la dernière. Sur la nouvelle ligne, C9 et C4 sont tous les deux passés à 0, mais pas à cause de la modification de R9 ou de R4 ou d'une quelconque « bufferisation ».

Au début de cette ligne entre C0=0 et 1, la condition C9=R9=0 et C4=R4=0 est vraie.
Cette ligne et les prochaines seront considérées comme les dernières.

Remarque : Lorsque C9 devient égal à R9 (dernière ligne), il faut attendre que C0=2 pour modifier R9, car le CRTC 0 a besoin des traitements particuliers en C0=0 et 1 pour gérer C9 et désactiver notamment la gestion additionnelle de lignes activée par défaut.

C4:	0																		
C9:	**7**																		
C0:	**0**	**1**	2	3	4	5	6	7	8	9	10	11	12	...	58	59	60	61	62	63
R9:	**7**	**7**	**0**	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0
R4:	0	0	0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0
	OUT R9,0														Update R12/R13 before C0=0					

Comment on sort ?
Pour arrêter d'obtenir des lignes C4=0 et C9=0, il est nécessaire de modifier R4 et/ou R9.
Étant donné que chaque ligne est aussi une "**dernière ligne**", C4 et R9 seront remis à 0 sur la ligne suivante, quelle que soit la mise à jour de R4 et/ou R9.
(car C4 sera forcé à 0, et de facto, C9 aussi).

Dans la perspective d'un code commun avec le CRTC 2, il est conseillé de :

- Modifier R12/R13 avant que C0=R1.
- Gérer la mise à jour de R9 en HSYNC comme indiqué au chapitre 12.4.2.
- Ne pas positionner R2=0 (afin que le BORDER ne reste pas activé).

12.3 CRTC 1

Si R4 est mis à jour avec la valeur de C4 :

- Si on était sur la ligne C9 entre 0 et R9-1, alors C9=C9+1 (C4 passera à 0 lorsque C9 repassera à 0 et l'offset (R12/R13) sera pris en compte).
- Si on était sur la dernière ligne (C9=R9), alors C9 passe à 0, C4=0 et R12.R13 est pris en compte.

Si R4 est mis à jour avec une valeur inférieure à C4, alors C4 va s'incrémenter jusqu'à sa valeur maximale (127) avant de reboucler.

Contrairement au CRTC 0, mettre R4=0 sur la dernière ligne d'un frame va provoquer un "débordement" de C4, la valeur 0 étant gérée comme un cas général.
Si on veut mettre R4 à 0 pour que C4 boucle à 0, il faut donc le faire uniquement quand C4=0.

Créer une rupture ligne à ligne sur ce CRTC est trivial, car il suffit de mettre R4 et R9 à 0 lorsque C4 et C9 sont à 0.

12.4 CRTC 2

12.4.1 CONCEPT DE DERNIERE LIGNE

Comme sur le CRTC 0, il existe une notion de « **Dernière Ligne** » qui arme irrémédiablement la remise à 0 de C4 et C9 sur la ligne suivante.

Lorsque cet état « Dernière Ligne » est positionné, il ne peut plus être modifié.

Cela signifie que si cet état « **Dernière Ligne** » est vrai, alors C4 et C9 passeront à 0 sur la ligne suivante, quelles que soient les valeurs programmées ensuite dans R4 et R9 durant le reste de la ligne.

L'état de « **Dernière Ligne » est évalué en début de ligne** (sur la position C0=0) ou **lors d'une mise à jour de R4 et/ou R9 si un état « Gestion Dernière Ligne » est vrai.**

Si C4<>R4 ou C9<>R9 sur la position C0=0, alors l'état « Dernière Ligne » est faux.
L'état « Gestion Dernière Ligne » est vrai sauf **si C4=0 et C9=0 (auquel cas il est faux)**. Autrement dit, si on n'est pas sur la dernière ligne en début de ligne, il sera possible de mettre à jour l'état « **Dernière Ligne** » durant la ligne (en modifiant R9 et/ou R4 hors d'une HSYNC), sauf si on est sur une première ligne (C9=C4=0) (dans ce cas les mises à jour de R9 et/ou R4 ne sont pas prises en compte pour évaluer l'état « **Dernière Ligne** »).

L'état de « **Gestion Dernière Ligne** » permet, si il est vrai, d'évaluer les conditions de l'état « **Dernière Ligne** » sur les positions C0>0 à partir d'une mise à jour de R9 et/ou R4. Il y a une **exception à cette règle**, si la mise à jour de R9 et/ou R4 qui satisfait la condition « **Dernière Ligne** » survient durant une HSYNC (l'état « **Dernière Ligne** » reste faux).

Si C4=R4 et C9=R9 sur la position C0=0, alors l'état « Dernière Ligne » est vrai (l'état « **Gestion Dernière Ligne** » est faux). Cependant, il existe 2 exceptions pour lesquelles l''état « **Dernière Ligne** » est faux :

- Si la ligne précédente était une dernière ligne. Un état (oui, encore un) « **Dernière Ligne Précédente** » est géré sur la dernière position C0 de la **HSYNC**.
- Si une **HSYNC** a lieu sur la position C0=0

Durant une **HSYNC**, un test est réalisé sur la position C0=R2+R3-1, afin de déterminer si la ligne N est une dernière ligne pour la ligne N+1 (en C0=0). Ainsi, si C4=R4 et C9=R9, alors « **Dernière Ligne Précédente** » est vrai, sinon il est faux.

Par ailleurs, si C4<>R4 ou C9<>R9 sur cette dernière position de la HSYNC, cela met à jour l'état de « **Gestion Dernière Ligne** » en le positionnant à vrai. Ainsi, si « **Dernière Ligne** » était faux et sa gestion également à cause d'une ligne précédente « déjà dernière » (évaluée durant la HSYNC), cela permet de forcer la réévaluation de dernière ligne afin de contourner l'exception sur les positions C0>=(R2+R3)

Lorsque l'état « **Dernière Ligne** » est vrai plus rien ne peut empêcher le passage de C4=0 (et C9=0) sur la ligne suivante, sauf si une ligne additionnelle est programmée via R5. Si R5 est programmé avec une valeur supérieure à 0 sur n'importe quelle position de C0, alors les lignes

additionnelles programmées dans R5 précéderont la remise à 0 de C4 et de C9. Lors de l'affichage de ces lignes additionnelles, C4 sera incrémenté (en dépassant R4)(voir chapitre 11.2.3).

Si un objectif est de maintenir la remise à 0 de C4 et de C9 sur chaque ligne, il est possible de forcer l'état de dernière ligne sur une première ligne, en créant la condition en fin de HSYNC qui permet sa prise en compte.

Autrement dit il suffit d'autoriser la gestion de la « **dernière ligne** » sur le dernier caractère de la HSYNC pour permettre à ce CRTC de gérer la remise à 0 de C4 et de C9. **Modifier R9 judicieusement avant et après la HSYNC permet d'atteindre cet objectif**.

12.4.2 CAS PRATIQUE : RUPTURE LIGNE À LIGNE (R.L.A.L.)

L'objectif de cette technique est que toutes les lignes affichées débutent avec C4=0 et C9=0, afin que les registres R12 et R13, modifiés avant que C0=R1, soient pris en compte.

Supposons que nous soyons sur la **dernière ligne** d'un frame avec C4=R4 et C9=R9 (R4 et/ou R9 sont supérieurs à 0). On sait que sur la prochaine ligne, C9=0 et C4=0, mais on souhaite que pour les prochaines lignes, C9 et C4 soient également à 0 (et pouvoir modifier l'adresse via R12 / R13).

Il sera nécessaire de reprogrammer R9 et R4 avec 0 afin que la condition de **dernière ligne** soit active sur chaque ligne. Chaque nouvelle ligne sera **une première et une dernière ligne** du frame.

Pour cet exemple, la HSYNC sera positionnée sur la position C0=1 (via R2=1).
Il faut tenir compte des contraintes liées aux autres défauts de gestion durant la HSYNC, comme la non-activation de la broche VSYNC et la désactivation du BORDER sur C0=0.

On va considérer que cette HSYNC aura le droit au temps minimum nécessaire à une synchro horizontale, à savoir 6 µsec (via R3=6) car il sera nécessaire de modifier 2 fois R9. Autant le faire au plus vite lorsque l'index sur le registre du CRTC est déjà sélectionné.

Si C9=C4=0 et que la condition de « **dernière ligne »** est vraie sur le dernier caractère de la HSYNC, alors l'état « **Dernière Ligne** » ne pourra plus être géré. En modifiant R9 durant la HSYNC afin que la condition de dernière ligne soit fausse, il est possible d'autoriser cette gestion d'évaluation de l'état « **Dernière Ligne** ». En modifiant R9 avec une autre valeur que celle de C9, il est possible d'activer la « **Gestion Dernière Ligne**» afin qu'elle puisse être prise en compte.

Une fois que la HSYNC est terminée, remettre la valeur de R9 identique à celle de C9 (donc 0), permet d'indiquer quelle est la limite réelle de C9, afin que la condition « **dernière ligne** » soit vraie pour permettre la remise à 0 de C4.

Exemple en image avec 3 lignes:

1ère ligne : On suppose ici que C4=R4=0 et C9=R9=7, et la HSYNC est représentée en orange :

	R2													…						
C4:	0																			
C9:	**7**																			
C0:	0	1	2	3	4	5	6	7	8	9	10	11	12	…	58	59	60	61	62	63
C3:		1	2	3	4	5	6													**R1**
R9:	7	7	7	7	7	7	7	0	0	0	0	0	0	…	0	0	0	0	0	0
R4:	0	0	0	0	0	0	0	0	0	0	0	0	0	…	0	0	0	0	0	0
						OUT R9,0					Update R12/R13 before C0=R1									

Cette ligne est considérée comme la dernière du frame car C9=R9 et C4=R4 lorsque C0=0. Cette condition de « dernière ligne » est vraie car il n'y pas de HSYNC sur C0=0 et la ligne précédente (C9=6) n'était pas une dernière ligne (évaluation faite en fin de HSYNC de C9=6).
R12 et/ou R13 sont modifiés avant que C0=R1 afin que VMA' soit mis à jour avec R12/R13 car on est... sur la dernière ligne.

Remarque : Cette affectation de R12/R13 dépend de l'état « dernière ligne » lorsque C0=R1 (et non de l'égalité C4=R4 et C9=R9 qui positionne cet état). Ceci implique que la modification de R9 avant l'égalité C0=R1 n'empêche pas cette affectation.

Remarque : La modification de R9=0 sur cette ligne n'est plus pris en compte pour l'état de dernière ligne car cet état est déjà passé à « vrai ». Il va cependant permettre de créer la condition « dernière ligne » sur C0=0 de la ligne suivante.

2ème ligne :

	R2													…						
C4:	0																			
C9:	**0**																			
C0:	0	1	2	3	4	5	6	7	8	9	10	11	12	…	58	59	60	61	62	63
C3:		1	2	3	4	5	6													**R1**
R9:	**0**	0	0	**1**	1	1	1	**0**	0	0	0	0	0	…	0	0	0	0	0	0
R4:	0	0	0	0	0	0	0	0	0	0	0	0	0	…	0	0	0	0	0	0
		OUT R9,1				OUT R9,0					Update R12/R13 before C0=R1									

Sur cette première ligne 0 (C4=R4=C9=R9=0), l'état « **Dernière Ligne** » est vrai.
Cependant, la condition de « **dernière ligne** » n'ayant pas changé durant la HSYNC précédente, la ligne précédente est considérée comme une dernière ligne. À ce titre, la ligne courante n'est plus considérée comme une dernière ligne sauf si la « **Gestion Dernière Ligne** » est réactivée durant la **HSYNC** (via la condition C4<>R4 ou C9<>R9 sur la dernière position de la HSYNC).

Ainsi, sur C0=6 (dernier caractère de la HSYNC) le test C9(0)<>R9(1).
Ceci débloque l'évaluation de « **dernière ligne** ».
Le OUT R9,0 a pour effet de modifier R9 lorsque C0=7 et que la HSYNC est terminée.

Sur C0=7, le CRTC arme la remise à 0 de C4 et de C9 (condition de « **dernière ligne** » satisfaite). Cette remise à 0 de C9 (et de C4) ne peut alors plus être « annulée ».

Si, dans l'exemple ci-dessus, un OUT R9,10 était ajouté derrière le OUT R9,0 (R9 serait mis à jour avec 10 sur C0=11), cela ne changerait rien pour C9 et C4, qui repasseraient à 0 sur la ligne suivante.

Ainsi sur la 3ème ligne, on aura C9=R9=0 et C4=R4=0.

Remarque : Si la HSYNC avait été positionnée après la mise à jour de R9 avec 0 sur la 1ère ligne, alors elle n'aurait pas été considérée comme une dernière ligne lors de l'évaluation C9=R9/C4=R4 en fin de HSYNC sur la ligne C9=7. Il n'aurait alors pas été nécessaire de modifier R9 durant la HSYNC de la ligne 2 pour que l'état « **Dernière Ligne** » reste armé pour la 3ème ligne.

3ème ligne :

	R2																			
C4:	0																			
C9:	**0**																			
C0:	0	1	2	3	4	5	6	7	8	9	10	11	12	…	58	59	60	61	62	63
C3:		1	2	3	4	5	6													**R1**
R9:	**0**	0	0	**1**	1	1	1	**0**	0	0	0	0	0	…	0	0	0	0	0	0
R4:	0	0	0	0	0	0	0	0	0	0	0	0	0	…	0	0	0	0	0	0
		OUT R9,1				OUT R9,0					Update R12/R13 before C0=R1									

Cette ligne ne pose pas davantage de problème, car la condition de « **Dernière Ligne** » est de nouveau fausse sur C0=6 (HSYNC) pour autoriser sa prise en compte, qui a lieu sur C0=7 lorsque R9 passe à 0 (pour satisfaire C9=R9/C4=R4 (Hors HSYNC)).
Et ainsi de suite…

Remarque : Pour des raisons pratiques il est parfaitement possible de déplacer l'instruction OUT R9,1 de 4 µsec (sur C0=4) (R9 est différent de C9 sur C0=R2+R3-1).

Enfin, tout comme sur le CRTC 0 pour « sortir » d'un traitement de « **dernière ligne** », il faut garder à l'esprit que lorsqu'on remet R9 à une valeur supérieure à 0, cette valeur n'est plus prise en compte si le CRTC considère qu'il est sur une dernière ligne.

Il va remettre C9 et C4 à 0 sur la ligne suivante, tel que le ferait le CRTC 0.

Si on a laissé la condition de « **Dernière Ligne** » active durant la dernière HSYNC (il y a eu deux dernières lignes avec un même C9/R9), C9 sera traité en relation avec la valeur de R9. C9 sera incrémenté si C9<>R9 comme le ferait un CRTC 1, mais C4 sera cependant incrémenté inconditionnellement si C9=R9 (et C9 repassera à 0).

Dans la perspective d'un code commun avec le CRTC 0, il est conseillé de :

- Traiter la mise à jour de R9 sur la dernière ligne comme sur les lignes précédentes pour que la prochaine ligne soit considérée comme une dernière ligne, afin d'être compatible avec le fonctionnement « dernière ligne » du CRTC 0.
- Ne pas modifier R9 avant C0=2 pour les mêmes raisons.

12.5 CRTC 3, 4

Si R4 est mis à jour avec la valeur de C4 :

- Si on était sur la ligne C9 entre 0 et R9-1, alors C9=C9+1, C4 passera à 0 lorsque C9 repassera à 0 et l'offset (R12/R13) est pris en compte.
- Si on était sur la dernière ligne (C9=R9), alors C9 passe à 0, C4=0 et R12/13 sont pris en compte.

La modification du registre 4 est prise en compte immédiatement à la fin de la ligne.
Si on veut mettre R4 à 0 pour que C4 boucle à 0, il faut donc le faire quand C4=0.

Si R4 est mis à jour avec une valeur inférieure à C4, alors il y a débordement du compteur C4. (contrairement à ce qui se produit avec C9/R9).

À noter également qu'une gestion additionnelle de lignes verticale n'incrémente pas C4 au-delà de R4, comme le font les CRTC 0, 1 et 2.

Créer une rupture ligne à ligne sur ce CRTC est encore plus trivial que sur CRTC 1, car il suffit de mettre R4 et R9 à 0 lorsque C4=0.

En effet, C9 passe à 0 si C9>R9, ce qui améliore le niveau de compatibilité entre les CRTC 0 et 1.

13 COMPTAGES : REGISTRE R0

13.1 GÉNÉRALITÉS

Le registre R0 du CRTC sert à définir le nombre de caractères que le circuit va générer sur une ligne. Un compteur C0 (aussi appelé HCC par quelques excentriques) compte de 0 jusqu'à la valeur de R0 incluse. Ce registre contient le nombre de caractères CRTC souhaité par « ligne » moins 1.

Lorsque C0 passe à 0, différents compteurs sont mis à jour (C4, « C5 », C9, ...).

Le compteur C0 est comparé à plusieurs autres registres, comme R1 (gestion du border/pointeur vidéo) et R2 (gestion HSYNC).

On pourrait s'attendre à ce que la condition Cx=Rx définisse l'incrémentation ou la remise à 0 d'autres compteurs.

Cependant, si C0 n'atteint pas certaines valeurs fixes définies en interne, cela pose quelques problèmes, notamment sur le CRTC 0, car des traitements ont lieu sur ces valeurs précises de C0.

Ces traitements spécifiques ont pour objet d'autoriser la remise à zéro ou l'incrémentation d'autres compteurs.

C0 est comparé à R2 pour définir le début de la HSYNC. Il est utile ici de rappeler qu'il y a un décalage entre les registres internes du CRTC et l'affichage effectif des caractères correspondants par le GATE ARRAY et/ou ASIC. Les compteurs du CRTC sont en avance sur les GATE ARRAY.

Le GATE ARRAY ne respecte pas ce "différé" pour générer les HSYNC. Ainsi, la HSYNC survient sur le C0 du « CRTC ».

Les ASIC cependant, respectent ce différé pour générer la HSYNC au moment où le caractère correspondant à C0=R2 est affiché, ce qui correspond à C0+1 du CRTC.

13.2 CRTC 0

Ce CRTC réalise des opérations sur des positions précises de C0 (armements de remises à zéro et d'incrémentations de compteurs, inhibition de traitements).

Le détail de ce fonctionnement permet de mieux appréhender le traitement spécifique de remise à 0 de C4, qui intervient seulement lorsque le CRTC évalue qu'il est sur la **dernière ligne du frame.**

Les concepteurs du circuit ont sans doute "préféré" gérer sous conditions une remise à 0 de C4, car ce compteur a été prévu pour **pouvoir dépasser R4** sur tous les CRTC « non ASICé ».

C'est notamment le cas lorsque R5 est supérieur à 0 (il devient alors la nouvelle limite de C9+1) ou que le mode « Interlace » est actif durant un frame « pair ».

Ces différents « tests » ont été répartis sur plusieurs valeurs de C0 :

- **Lorsque C0=0**, alors les compteurs C4 et C9 sont mis à jour en fonction d'états décidés sur la ligne précédente. Puis les compteurs sont évalués pour programmer leur mise à jour sur la ligne suivante. Si C9 a atteint R9 alors C4 sera programmé pour être incrémenté sur la prochaine ligne. Si une gestion additionnelle est en cours, alors si C9 a atteint R5, C9 repasse à 0. À noter que cette évaluation de C9 a lieu une seule fois pour la ligne (cette gestion de C9 est désactivée une fois réalisée). L'incrémentation de C4 sera éventuellement annulée sur C0=2 si cette ligne n'était pas la dernière du frame (C4<>R4) ou qu'il y a des lignes additionnelles à ajouter (R5>0 et/ou ligne « Interlace » active).
- **Lorsque C0=1**, alors la gestion de C9 est de nouveau autorisée pour le prochain C0=R0. Si cette « autorisation » n'est pas donnée (contexte R0=0) alors le compteur C9 n'est plus géré sur le C0 suivant. Si R0 est mis à 0 sur la position C0=0, C9 est préalablement mis à jour par rapport à la dernière valeur de R9 si R0 de la ligne précédente était >1.
- **Lorsque C0=2**, le CRTC détermine si une gestion de ligne additionnelle doit avoir lieu. Si cette gestion additionnelle est activée, alors C4 sera incrémenté quelle que soit la valeur de R4 pour le prochain C0=0. Dans le cas contraire, C4 et C9 repasseront à 0.

Ainsi :

- Si **R0=0**, alors C0 n'atteint jamais 1 (et donc reste à 0) et **C9 ne peut plus compter**.
  Il reste alors **figé sur la valeur qu'il avait avant que R0=0.**
  - Si C9<>R9 sur le premier C0=0 pour lequel R0=0, alors **tous les compteurs du CRTC sont figés** tant que R0=0. Par exemple figer R0=0 pendant 64x8 µsec revient à « oublier » 8 lignes (C4-1 si R9=7).
  - Si C9=R9 sur le premier C0=0 pour lequel R0=0, **alors C4 est incrémenté une fois sur le second C0=0**. C4 n'est incrémenté qu'une seule fois car la gestion additionnelle n'est « désactivée » que sur C0=2 (une gestion additionnelle n'incrémente donc C4 qu'une seule fois, car elle désactive la gestion C9=R9 pour basculer sur la gestion C9+1=R5). Après ce second C0=0, **tous les compteurs du CRTC sont figés** tant que R0=0.

  Remarque : La gestion C9 ayant lieu 1 seule fois si C0 ne dépasse pas 1, la mise à jour des registres R4, R5 et R9 n'est plus prise en compte tant que R0=0. Par contre, R8 continue d'être pris en compte à chaque fois que C0=0.

- Si **R0=1**, alors C0 n'atteint jamais 2, et la gestion additionnelle de ligne(s) décidée en C0=0 reste enclenchée si C4=R4 et C9=R9 sur le caractère précédent (C0=0). C9 peut s'incrémenter en fonction de R9. Si R5=0, la gestion additionnelle dure alors 1 ligne avant de cesser (C4+1, C9=0). Sur la prochaine ligne, la fin de gestion additionnelle repositionne C4=0 et C9=0.

Les conditions de remise à 0 de C4 sont donc multiples :

- Lorsque C0=0, C4=R4 et C9=R9 conditionne l'état "**dernière ligne**", qui active alors un état « **gestion additionnelle** » pour lequel C4 incrémenté sans condition une seule fois. À noter que durant cette gestion additionnelle, C9 continue à être testé avec R9 pour la mise à jour de VMA/VMA'.
- Lorsque C0=2, si il n'y avait aucune ligne additionnelle en attente, l'état de gestion additionnelle est désactivé (R5=0 ou pas de ligne « Interlace », permet de le déterminer) ce qui évite que C4 soit incrémenté pour l'addition des lignes additionnelles.

À défaut de remplir toutes ces conditions après C0>2, C4 sera incrémenté lorsque C0 repassera à 0. Autrement dit, **modifier R4 ou R9 après que les tests soient réalisés ne changera rien au résultat de C4**.

Si le CRTC n'était pas sur la dernière ligne (ou en ayant modifié R4 ou R9 après cette évaluation sur C0=0), le sort de C4 est scellé : **C4=C4+1**.

Si R5 est programmé sur la dernière ligne du frame avec une valeur supérieure à 0 avant que C0=3 (C0=0, 1 ou 2), alors R5 lignes additionnelles seront prises en compte (avec C4=R4+1 sur la ligne suivante). **Par contre, si R5 devient supérieur à 0 lorsque C0>2, alors aucune ligne additionnelle ne sera ajoutée au frame**. La ligne suivante correspondra à un nouveau frame avec C4=C9=0.

GEL DE VSYNC…

D'autres opérations ont lieu sur le CRTC 0 entre les valeurs 0 et 2 de C0.
Notamment pour déterminer si une **VSYNC** sera autorisée pour la prochaine occurrence de C0=0.

À chaque fois que C0=2, un état permet de valider la prise en compte de C4=R7 sur le prochain C0=0. Cet état est annulé dès que C0=0.

Ainsi, si R0 est modifié avec une valeur inférieure à 2 sur la ligne précédent l'équivalence C4=R7 (Sur la ligne où C9=R9 et C4=R7-1), la VSYNC ne sera pas autorisée sur C0=0. Cela provoquera par ailleurs un blocage de la VSYNC pour la valeur C4=R7.

Une modification de R7 lorsque C0<2 ayant pour objet de « déclencher » une VSYNC après C0=0 (lorsque C4<>R7) va également provoquer le blocage de la VSYNC pour cette valeur C4=R7.

Le déblocage de la VSYNC n'est alors plus possible pour cette valeur de C4, sauf deux conditions particulières. Voir chapitre 15 pour la description de ces conditions.

GEL EN LIGNE ADDITIONNELLE

Si une ligne additionnelle est programmée (R5=1 par exemple), on aura sur cette ligne C4=R4+1 et C9=0. Si R0 passe à 0 sur C0=0 de cette ligne, alors C9 reste figé à 0 et C4 ne pourra passer à 0 que lorsque C9 sera de nouveau géré (dès que C0=1). C4 et C9 restent figés comme pour une

ligne « non additionnelle » pour laquelle C9<>R9. Modifier R5 pendant une période durant laquelle R0=0 n'a aucune incidence sur la valeur des compteurs tant que C0 reste à 0, car C9 n'est plus géré par rapport à R5 tant que C0 n'est pas repassé à 1.
À noter que lorsque la gestion C9 est inhibée, la mise à jour de R9 et R4 est également ignorée tant que R0=0.

GEL DE C9…

Le sort de C9 est particulier lorsque R0=0 puisque ce dernier n'est plus incrémenté.
Au premier passage de C0=0, lorsque R0 devient égal à 0, la valeur de C9 est calculée une première fois par rapport à R9 (par exemple si C9 valait 4 et que R9=7, C9 passe à 5).

Sur le premier C0=0 :

- Le CRTC gère différents états mis à jour sur les C0=0.1.2 précédents pour mettre à jour C4 et C9 sur C0=0. Chaque état est annulé une fois le compteur mis à jour. En principe, 4 états sont possibles :
  - Incrémenter C4
  - Annuler C4
  - Incrémenter C9
  - Annuler C9

  À noter que le traitement des états de C9 et sa gestion de comptage est subordonnée à un autre état de 'gestion C9'.
- Des tests sont réalisés pour déterminer les prochains états C4/C9, notamment par rapport à la gestion de comptage C9/R9 ou par rapport à celle de C9/R5 si une gestion additionnelle est décidée (dans ce cas, c'est la valeur de R5 par rapport à C0 qui sert à fixer la fin de la gestion additionnelle).
- Une gestion additionnelle est décidée si C4=R4 et C9=R9. Elle serait en principe confirmée sur C0=2 si C0 réussissait à atteindre cette valeur.
- La gestion de traitement C9 est désactivée. Elle serait en principe activée sur C0=1 si C0 réussissait à atteindre cette valeur.

Sur le second C0=0

- La gestion de C9 n'a pas été réactivée (car C0 n'a pas atteint 1). La valeur de C9 reste donc la même que celle qu'il avait sur le premier « C0=0 ».
- Deux situations se présentent alors :
  - Si C9 avait atteint R9 sur le premier C0=0, alors la mise à 0 de C9 avait été armée ainsi que l'incrémentation de C4. C9 étant figé, seul C4 va s'incrémenter.
  - Si C9 était différent de R9 sur le premier C0=0, alors l'incrémentation de C9 avait été armée (mais pas l'incrémentation de C4). C9 étant figé, aucun compteur n'a bougé.

Sur le troisième C0=0 (et les suivants)

- La gestion de C9 n'a pas été réactivée (car C0 n'a pas atteint 1). La valeur de C9 reste donc la même que celle qu'il avait sur le premier « C0=0 ».
- Dans la situation où C4 avait été incrémenté sur le second « C0=0 », cette incrémentation est désactivée car elle a eu lieu. Tous les compteurs sont figés.

Remarque : R5 est une quantité de ligne(s). C'est donc la valeur C9+1 qui est comparée avec R5, alors que R9 est une valeur de C9 à atteindre.
R5=0 indique qu'il n'y a « en principe » aucune ligne.

Si R0 est modifié pour redevenir supérieur à 1, **la gestion d'incrémentation de C0 reprend normalement** sans que les autres compteurs ne soient affectés.

Cependant, **si les conditions d'une dernière ligne sont satisfaites durant C0=0, alors C9 ne sera plus géré avec R9 mais avec R5**. Le passage par C0=2 ne l'annule pas.
Elle s'arrêtera seulement lorsque C9+1=R5 sur C0=0.

La gestion d'ajustement vertical a 2 particularités sur le CRTC 0:

- **C4 est incrémenté une seule fois, quelle que soit la valeur de R5.**
- **C4 revient à 0 une fois l'ajustement terminé, quelle que soit la valeur de R4.**
  Cela signifie que C4 n'est plus géré lorsque C9 atteint R9 durant une gestion additionnelle. À noter cependant que le CRTC continue de gérer les transferts VMA/VMA' spécifiques à C0=R1 lorsque C9=R9. (Voir chapitre 11, page 78)

Remarque : Sur le CRTC 1, 2, 3 et 4, l'évaluation de « **dernière ligne** » a lieu durant toutes les valeurs de C0, **alors que le CRTC 0 ne le gère que sur C0=0 (et complète l'état sur C0=2)**. Le CRTC 2 arme cependant une remise à 0 de dernière ligne qui ne peut plus être désarmée ensuite, quelles que soient les valeurs de R9 et R4. En respectant la plus forte contrainte, si R4 est modifié afin que sa valeur soit mise à jour sur C0=0 (OUT déclenché sur C0vs=#3E sur CRTC 0, 1, 2 et sur #3D sur CRTC 3, 4 pour R0=#3F) alors une seule instruction est nécessaire sans adaptation du code. Si toutefois il est aussi nécessaire de modifier R9, qui fait partie des conditions de « dernière ligne », il est nécessaire d'anticiper sa mise à jour.
Voir chapitre 10, page 71, et chapitre 12, page 88.

13.2.1 CAS PRATIQUE : R0=1

Lorsque R0 vaut 1 (et que C0<2) on obtient des "lignes" de 2 caractères (2 µsec au total).

Si C4=R4 et C9=R9 lorsque C0=0, alors on est potentiellement sur la dernière ligne du frame. Mais C0 n'ayant pas atteint 2, la gestion de ligne additionnelle n'est pas désarmée et va avoir lieu.

Si C4=R4 et C9=R9, alors "le frame" prend fin lorsque C0 atteint R0.
C'est notamment le cas lorsque R4 et R9 valent 0, pour créer des "frames" de 2 µsec.
(La première ligne est alors également la dernière)

Ainsi :

- Lorsque C0=0, le CRTC évalue si il est sur la dernière ligne, et si c'est le cas, arme un flag interne par défaut pour déclencher la gestion d'ajustement vertical.
- Lorsque C0=2, le CRTC évalue les conditions pour désarmer ce mécanisme d'ajustement vertical (notamment en testant la valeur de R5).

En conséquence :

- Si C0 ne dépasse jamais 1, et que R4=R9=0, le CRTC génère un ajustement vertical de "1 ligne" à chaque "frame" lorsque C4=R4 et C9=R9.
- Le CRTC ne peut plus désarmer la gestion additionnelle (test de R5 et/ou Ligne additionnelle « Interlace ») et un "frame" de 2 µsec est généré. Cela se traduit par une incrémentation unique de C4 au-delà de la valeur programmée dans R4. **L'ajustement cesse cependant après 1 ligne**. Pour une raison que je n'ai pas encore déterminée, C9 (et non C9+1) est comparé à R5 pour stopper l'ajustement.
  Remarque: C4 et C9 repassent à 0 une fois l'ajustement terminé.

Lorsque R4=R9=0, chaque "ligne" de 2 µsec est donc immédiatement suivie par une "ligne" de 2 µsec pour laquelle C9=0 et C4=1.

Une fois cette ligne "additionnelle" de 2 µsec terminée, C4 et C9 étant passés à 0, l'offset est pris en compte. Dans cette situation, les registres R12/R13 peuvent donc être pris en compte seulement après 4 µsec, alors que c'est possible toutes les 2 µsec sur un CRTC 1, 2, 3 ou 4 (et même 1 µsec lorsque R0=0 sur ces CRTC). Si R9>0, cette "ligne" de 2µsec (pour laquelle C4=1) survient après la dernière valeur de C9 (lorsque C9=R9).

13.2.2 CAS PRATIQUE : R0=0

Lorsque R0 vaut 0 et que C0=0, alors C0 reste à 0 (frame ou ligne de 1 µsec).

Les choses se compliquent légèrement :

- **Blocage du comptage C9**
  Étant donné que C0 n'atteint jamais 1, C9 n'est plus géré.
  Si par exemple, il valait 3 au moment ou R0 est passé à 0, il va rester à 3 quelle que soit la valeur de R9, tant que R0=0.
- **Dernier hoquet de C4**
  Si C9=R9 sur C0=0 (lorsque R0 passe à 0) alors C9 n'est plus géré, mais C4 sera cependant incrémenté quelle que soit la valeur de R4. C4 a été incrémenté sans que C9 soit revenu à 0. **Magique !**
  Après ça, plus aucun compteur ne sera géré à part C0.
- **Gestion additionnelle**
  Si C9=R9 et C4=R4 (lorsque R0 passe à 0), alors on est dans la même situation que dans le paragraphe précédent, sauf qu'une gestion additionnelle est activée, et qui le restera lorsque C0 pourra de nouveau dépasser 1. C'est alors R5 qui pilote la fin de la gestion additionnelle. Pour stopper cette gestion il faut programmer R5 avec C9+1.

Pour résumer :

C9 reste "figé" sur la valeur qu'il avait avant que R0=0 et ne s'incrémente donc plus, ni ne revient à 0.
Si C9=R9 et C4<>R4 alors C4=R4+1. Lorsque R0>0, C4 continue à être géré via C9/R9.
Si C9=R9 et C4=R4 alors C4=R4+1. Lorsque R0>0, C4 est géré par C9/R5.
Dans les autres cas, C4 reste figé.

Exemple **Contexte : C0=R0=C4=R4=C9=R9=R5=0**			
Compteurs	C0	C9	C4
1er caractère C0=0	0	0	0
Le pointeur vidéo courant est mis à jour. CRTC-VMA'=CRTC-VMA=R12/R13 Lorsque C0 boucle une première fois sur R0, C0 passe de 0 à ... 0. C4=0. Le CRTC est sur la fin de frame(il vient de finir les lignes R4/R9 d'un frame de 1 µsec (C4=R4/C9=R9))			
2ème caractère C0=0	0	0	1
C9=R9 et C4=R4. C4 est incrémenté (soit 1). On est sur une gestion additionnelle. À noter que C9 n'est pas « réellement » remis à 0 car il n'est plus géré.			
3ème caractère C0=0 (et suivants…)	0	0	1
Le CRTC est en gestion additionnelle, mais C9 est figé.			
Lorsque R0 redevient > 2	0	0	1
La gestion additionnelle étant en cours, elle ne peut plus être annulée sur C0=2 C9+1 sera comparé à R5 sur le prochain C0=0			
Prochain C0=0 à l'issue de de C0=R0>2	0	1	1
C9+1 étant différent de R5, alors C9 est incrémenté.			
Exemple 2 (suite du film) **On va remettre R0=0 sur le premier C0 après que C0=R0>2**			
Compteurs	C0	C9	C4
1er caractère C0=0	0	1	1
Reprise des valeurs de la fin du premier exemple			
2ème caractère C0=0	0	1	1
C9 est figé. La gestion additionnelle est toujours en cours.			

13.2.3 CAS PRATIQUE : RUPTURE VERTICALE LAST LINE (R.V.L.L.)

UN PEU D'HISTOIRE TECHNIQUE... LA R.V.I.

Mise au point initialement sur le CRTC 0 à l'aide d'un canif et d'une ficelle, une ancienne technique ancestrale appelée "RVI" (Rupture Verticale Invisible © Overflow) a pour objet de permettre de choisir le numéro C9 de la ligne visible en créant de petites lignes dans la partie non visible de l'écran (durant la HSYNC), tout en modifiant l'adresse de la ligne visible. Pour y parvenir, il existe ***à minima*** *3 manières différentes, qui peuvent être combinées : Soit en modifiant R9, soit en modifiant la taille des lignes (R0), soit en limitant leur nombre (action synchrone Z80A/CRTC). Cela permet de pouvoir contrôler individuellement l'adresse de beaucoup plus de lignes que les 2k (parmi 16k d'une page) autorisés par la valeur C9=0.*

Une des contraintes majeure de cette technique est ***l'interdépendance entre les lignes.*** *Aussi vous apprécierez vraiment le prochain chapitre.*

En effet la valeur de C9 calculée pour la nouvelle ligne affichée dépend de la valeur de C9 de la ligne précédente. Créer un algorithme qui détermine automatiquement les évolutions souhaitées de C9 en fonction du C9 courant altère encore davantage la CPU disponible sur une ligne de 64 µsec.

C'est en principe la valeur de R9 qui fixe la valeur de C9 maximale. Il faut garder à l'esprit que seuls les 3 bits de C9 sont "conservés" pour la composition de l'adresse.

Un des corolaires de cette technique est en général la mise à jour de CRTC-VMA via une mise à jour de R12 et/ou R13. Pour cela il faut que C9 et C4 repassent à 0.
Remarque : Pour le CRTC 2, il faut de surcroit que R12/R13 soient modifiés avant que C0=R1 et que C0 atteigne R1 lorsque C9=R9, mais vous n'êtes pas sur le bon chapitre.

Par exemple, si C9=8 correspond bien à C9=0 pour la constitution d'adresse, C4 ne repasse pas à 0 et donc R12/R13 ne peut pas être pris en compte. Pour qu'un changement d'adresse soit pris en compte, il ***faut que C9 "boucle" et repasse par 0, ainsi que C4****. Par ailleurs si le C9 affiché est une « dernière ligne », il va provoquer la remise à 0 de C9.*
Remarque : Pour le CRTC 1, il n'y a aucun besoin que C4 repasse à 0 pour que l'adresse soit prise en compte. C'est vrai à chaque fois que C0=0 tant que C4=0.

Lorsque des frames de 2 µsec sont créés dans le bord (ou pas si on veut exposer les dessous de l'affaire...) alors que R9>0, alors C9 va s'incrémenter jusqu'à atteindre R9.

Mais dans cette situation, lorsque C9=R9 (donc ici C0=1, puisque R0=1), une « ligne » additionnelle va être générée avec C4=1 et C9=0. Cette gestion "additionnelle" a pour conséquence la création d'une "ligne" complémentaire lorsque C9 atteint R9.

Le changement d'adresse n'a pas lieu car C4=1. C'est sur le frame suivant que C4 repassera à 0 (avec C9). Une des conséquences est que le nombre d'incrémentations de C9 pour ces "lignes de 2µs" est réduit de 1 par rapport aux CRTC 1, 2, 3, 4.

Cela ne permet pas d'accéder, avec R0=1, à C9=7 ou 5 en changeant l'offset, car il faut d'une part que ***R9 soit égal à 7 ou 5, mais qu'il ait pu reboucler à 0 avant****.*

Les valeurs finales de C9 impaires n'arrangent pas les choses, sauf si vous êtes un pro du mode « Interlace » "controlé" (bla, bla, avocat, bla, bla), mais je m'égare...

Cette technique ayant été développée sur le CRTC 0, l'usage de R0=0 a été abandonné rapidement à cause de la difficulté à appréhender le comportement des compteurs à l'époque.

Sur les CRTC 1, 2, 3 et 4, cette valeur (R0=0) ne pose aucun problème. Ni blocage de C9, ni ligne additionnelle C4=1, mais c'est plus subtil pour le CRTC 2. Il est donc bien plus simple de créer ce type de ruptures dans le bord permettant d'accéder à toutes les valeurs de C9 sur la ligne principale. La remise à 0 « armée » de « dernière ligne » est ici particulièrement intéressante par rapport au CRTC 1.

Au premier abord, la notion de « **dernière ligne** » sur le CRTC 0 (et le 2) parait contraignante, car elle ne permet pas une prise en compte « immédiate » des mises à jour des registres R4 et R9.

Par ailleurs, le CRTC 0 est limité à faire des ruptures de 2 µsec minimum pour un fonctionnement « correct » du compteur C9. Enfin, si une « dernière ligne » se produit lorsque cette ligne fait 2 µsec, alors une nouvelle ligne de 2 µsec est générée (C4=1).

Cependant, le principe de « **dernière ligne** » représente un formidable outil pour **programmer à l'avance une remise à 0 de C4 et C9**, notamment dans le cadre d'une rupture verticale, que je vais nommer Rupture Verticale Last Line (en bon franglais qui se respecte).

En effet, si on considère que chaque ligne visible est une « dernière ligne », alors **chaque première rupture de 2 µsec sera la première ligne d'un frame avec C4=C9=0**.
Ainsi il est possible d'anticiper la modification de R9 en sachant que le prochain C9 sera égal à 0.

Dans cette situation, la sélection du prochain C9 est grandement simplifiée puisqu'il suffit de mettre le numéro souhaité dans R9 et de créer le nombre de ruptures approprié (durant la HSYNC si l'objectif est de les cacher) afin que C9 sur la ligne suivante soit égal au R9 programmé.

La mise à jour de l'offset (R12/R13) a alors lieu lors de la première rupture de 2 µsec.
Étant donné que la dernière ligne (C9=R9) est toujours la ligne visible dont la taille est supérieure à 2 µsec, le CRTC ne génère pas de rupture de 2 µsec complémentaire.

Cela signifie qu'il faut 7 ruptures de 2 µsec pour atteindre C9=7, soit 14 µsec au total.

Cela signifie aussi, sauf à tricher de manière barbare avec la valeur des offset, que les ruptures de 2 µsec commencent plus ou moins tôt dans les 14 µsec afin d'amener C9 à la valeur souhaitée.

À cette fin, la valeur de R0 doit donc correspondre au moment où les ruptures de 2 µsec commencent. Dis autrement R0 doit être égal à 63 – (R9 x 2) (pour R9>1).

Sur le schéma ci-dessous les zones en rose correspondent aux instructions OUT sur R0. R0-1 correspond à la mise à jour de R0 au début de la ligne (et donc débutée sur la fin de la ligne précédente), et R0-2 correspond à la valeur de R0 pour les ruptures « cachées ».

R0-1	R0-2	R9
63	x	0
50	12	1
59	1	2
57	1	3
55	1	4
53	1	5
51	1	6
49	1	7

			R2		OUT R0-2												OUT R0-1	New C9
C0:	... 47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	0
C0:	... 47	48	49	50	**0**	1	2	3	4	5	6	7	8	9	10	11	12	1
C0:	... 47	48	49	50	51	52	53	54	55	56	57	58	59	**0**	1	0	1	2
C0:	... 47	48	49	50	51	52	53	54	55	56	57	**0**	1	0	1	0	1	3
C0:	... 47	48	49	50	51	52	53	54	55	**0**	1	0	1	0	1	0	1	4
C0:	... 47	48	49	50	51	52	53	**0**	1	0	1	0	1	0	1	0	1	5
C0:	... 47	48	49	50	51	**0**	1	0	1	0	1	0	1	0	1	0	1	6
C0:	... 47	48	49	**0**	1	0	1	0	1	0	1	0	1	0	1	0	1	7
C3:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15			
																	R3	

Remarque: La 2ème instruction OUT qui débute sur le 63ème caractère de la ligne a pour objet de redimensionner la ligne pour le C9 suivant. Cette instruction ne peut pas débuter sur le 64ème caractère sans provoquer un débordement de C4 sur la ligne redimensionnée (voir chapitre 13.6.2).

Sur un moniteur CTM, la partie visible d'une ligne est de 48 µsec, et 16 µsec ne sont pas visibles. Cependant, pour que le frame soit centré, il faut que la HSYNC débute sur le 51ème caractère, lorsque C0 atteint 50 (si R0=63). Et donc que R2 soit programmé avec cette valeur.

Pour disposer des 14 µsec nécessaires pour atteindre C9=7, il faut que R0 soit programmé avec 49 et il faudrait en principe que R2 soit à 0 sur cette ligne précise (en inhibant toutefois les HSYNC en protégeant la zone avec une valeur de R3 couvrant toutes les autres occurrences de C0=0).

Une HSYNC n'étant pas forcément nécessaire sur chaque ligne, il est toutefois possible de positionner R2 avec 50, en sachant que les lignes C9=7 profitent de la synchronisation des autres lignes, mais cela implique qu'il n'y ait pas plusieurs lignes C9=7 consécutives.
C'est ce qui est fait dans la démonstration R.V. fournie avec SHAKER, où les lignes 7 ne sont pas synchronisées. Cela ne pose aucun problème sur des moniteurs natifs.

Pour pouvoir utiliser toute la ram sans cette contrainte, il suffit de positionner R2=49.
Si C9=7 n'est pas utilisé, il est possible de positionner R2=50.

13.3 CRTC 1

R0 accepte toutes les valeurs sans que cela pose de problème aux autres compteurs.

Si R0 vaut 0, alors C9 et R4 continuent d'être gérés normalement.

L'offset est modifiable selon les timings indiqués dans les schémas des pages suivantes.

Remarque : Il est possible de créer 14 ruptures "cachées", permettant d'accéder simplement à toutes les valeurs de C9, et ce d'autant que l'offset est pris en compte tant que C4=0 (et non pas lorsqu'il repasse à 0).

Remarque : La mise à jour d'un des registres d'offset prenant à minima 4 µsec, la seule instruction Z80A de mise à jour couvre 4 "frames (lignes C0)" lorsque R0=0.

Remarque : L'utilisation de l'instruction OUTI pour modifier R0 lorsque C4=C9=0 a des conséquences inattendues par rapport à cette même modification réalisée avec un OUT(C),R8. La modification du registre R0 est effective sur la 5ème µseconde de l'instruction OUTI, et sur le 3ème µseconde de l'instruction OUT(C),R8. En conséquence, l'instruction OUT(C),R8 doit débuter 2 µseconde plus tard que l'instruction OUTI. Mais dans cette circonstance, la HSYNC ou le BORDER sont affectés. Ce bug ayant été reproduit sur 2 machines différentes avec un CRTC 1 natif.

Je n'ai pas encore défini complètement ce qui se produit dans cette situation (dans la prochaine version du document), mais cela n'implique ni la position de la HSYNC (R2), ni la taille de la ligne (R1), et cela n'a lieu que lorsqu'un nouvel « frame » débute sur C0=0 lorsque C4=C9=0 (Sur C0>0 il n'y a pas de problème notable). La valeur de R0 a cependant une importance sur le bug et selon la machine. J'ai constaté une détérioration de la situation dans le temps sur un de mes CPC.

13.3.1 CAS PRATIQUE : RUPTURE VERTICALE INVISIBLE (R.V.I.)

La R.V.I., sur un CRTC 0, est très limitative car placer R0 < 2 pose quelques contraintes et cela limite le nombre de ruptures qu'il est possible de faire dans la partie « invisible » de la ligne.
En effet, lorsque R0 vaut 1, une ligne additionnelle est générée, et lorsque R0 vaut 0, C9 est affecté. Cette technique implique que C9 repasse à 0 pour que R12 et/ou R13 soi(en)t pris en compte, et cela nécessite impérativement un nombre élevé de ruptures.
Elle est impossible sur le CRTC 2, notamment à cause des contraintes de prise en compte de R12/R13.

Lorsqu'il est question d'écrire un programme fonctionnant sur les autres CRTC que le 0, la valeur R0=0 ne pose aucun problème particulier. Si l'objectif reste d'accéder à tous les C9, on préfèrera néanmoins la R.V.L.L. sur les CRTC 0 et 2 qui permet de s'affranchir d'un algorithme de gestion transitionnelle entre les C9. Les CRTC 1, 3 et 4 ne disposent pas de cette possibilité.

Le CRTC 1, par rapport aux CRTC 3 et 4 (et même si le CRTC 3 a bien d'autres moyens pour modifier l'offset de chaque ligne) dispose d'un avantage complémentaire pour gérer une R.V.I. (si je ne l'ai pas déjà écrit). En effet, l'offset est pris en compte tant que C4=0, quelle que soit la valeur de C9, contrairement à tous les autres CRTC. (voir chapitre 20.3).

Cela permet donc de modifier l'offset sans que C9 ait « rebouclé » à 0, et permet de limiter le nombre de ruptures nécessaires pour atteindre le C9 souhaité avec un offset « neuf », ce qui permet de placer R2 à 50 pour un centrage parfait.

Les tableaux suivants décrivent les 64 états transitionnels entre 2 C9. La méthode employée implique une mise à jour de R0 1 ou 2 fois par ligne, ainsi qu'une mise à jour de R9, R12 et/ou R13. Une joie pour tous les adeptes des SudokuZ80A !

Les valeurs des nouveaux C9 (indiqués en rouge) correspondent à une possibilité « C4=0 » du CRTC 1. Pour les CRTC 3 et 4, il suffit de remplacer les valeurs par celles indiquées sous chaque tableau. On notera que le passage C9=0 à 7 implique que R0 de la ligne principale soit égal à 49, limitant de facto la possibilité de mettre R2 à 50. Ce n'est pas du tout gênant pour un CRTC 4, puisque c'est la valeur qui permet le centrage sur ce circuit. (voir chapitre 16.1)

C9=0

Old C9	R0.1	R0.2	R9
0	63		0
0	63		1
0	59	0	2
0	57	0	3
0	55	0	4
0	59	0	5
0	58	0	6
0	57	0	7

	R2	OUT R0.2	OUT R0.1	New C9
C0:	... 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61	62 63		0
C0:	... 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61	62 63		1
C0:	... 47 48 49 50 51 52 53 54 55 56 57 58 59 **0** 0	**0** 0		2
C0:	... 47 48 49 50 51 52 53 54 55 56 57 **0** 0 0 **0**	0 0		3
C0:	... 47 48 49 50 51 52 53 54 55 **0** 0 0 0 **0** 0	0 0		4
C0:	... 47 48 49 50 51 52 53 54 55 56 57 58 59 **0** 0	0 0		5
C0:	... 47 48 49 50 51 52 53 54 55 56 57 58 **0** 0 0	0 0		6
C0:	... 47 48 49 50 51 52 53 54 55 56 57 **0** 0 0 0	0 0		7
C3:	1 2 3 4 5 6			
	R3			

Compatibility CRTC 3 & 4

Old C9	R0.1	R0.2	R9
0	59	0	1
0	53	0	5
0	51	0	6
0	49	0	7

	R2	New C9
C0:	... 47 48 49 50 51 52 53 54 55 56 57 58 59 **0** 0	1
C0:	... 47 48 49 50 51 52 53 **0** 0 0 0 0 **0** 0 0	5
C0:	... 47 48 49 50 51 **0** 0 0 0 0 0 **0** 0 0 0	6
C0:	... 47 48 49 **0** 0 0 0 0 0 0 **0** 0 0 0 0	7

C9=1

Old C9	R0.1	R0.2	R9
1	63		1
1	59	0	4
1	63		2
1	58	0	3
1	56	0	4
1	54	0	5
1	59	0	6
1	58	0	7

	R2	OUT R0.2	OUT R0.1	New C9
C0:	... 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61	62 63		0
C0:	... 47 48 49 50 51 52 53 54 55 56 57 58 59 **0** 0	0 **0**		1
C0:	... 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61	62 63		2
C0:	... 47 48 49 50 51 52 53 54 55 56 57 58 **0** 0 **0**	0 0		3
C0:	... 47 48 49 50 51 52 53 54 55 56 **0** 0 0 **0** 0	0 0		4
C0:	... 47 48 49 50 51 52 53 **0** 0 0 0 **0** 0 0	0 0		5
C0:	... 47 48 49 50 51 52 53 54 55 56 57 58 59 **0** 0	0 0		6
C0:	... 47 48 49 50 51 52 53 54 55 56 57 58 **0** 0 **0**	0 0		7
C3:	1 2 3 4 5 6			
	R3			

Compatibility CRTC 3 & 4

Old C9	R0.1	R0.2	R9
1	59	0	3
1	52	0	6
1	49	0	7

	R2	New C9
C0:	... 47 48 49 50 51 52 53 54 55 56 57 58 59 **0** 0	2
C0:	... 47 48 49 50 51 52 **0** 0 0 0 0 **0** 0 0 0	6
C0:	... 47 48 49 50 **0** 0 0 0 0 0 **0** 0 0 0 0	7

C9=2

Old C9	R0.1	R0.2	R9
2	63		2
2	59	0	2
2	58	0	2
2	63		3
2	57	0	4
2	55	0	5
2	53	0	6
2	59	0	7

	R2	OUT R0.2	OUT R0.1	New C9
C0:	... 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61	62 63		0
C0:	... 47 48 49 50 51 52 53 54 55 56 57 58 59 **0** 0	0 **0**		1
C0:	... 47 48 49 50 51 52 53 54 55 56 57 58 **0** 0 **0**	0 0		2
C0:	... 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61	62 63		3
C0:	... 47 48 49 50 51 52 53 54 55 56 57 **0** 0 **0** 0	0 0		4
C0:	... 47 48 49 50 51 52 53 54 55 **0** 0 0 **0** 0 0	0 0		5
C0:	... 47 48 49 50 51 52 53 **0** 0 0 0 **0** 0 0 0	0 0		6
C0:	... 47 48 49 50 51 52 53 54 55 56 57 58 59 **0** 0	0 0		7
C3:	1 2 3 4 5 6			
	R3			

Compatibility CRTC 3 & 4

Old C9	R0.1	R0.2	R9
2	59	0	3
2	51	0	7

	R2	New C9
C0:	... 47 48 49 50 51 52 53 54 55 56 57 58 59 **0** 0	3
C0:	... 47 48 49 50 51 **0** 0 0 0 0 **0** 0 0 0 0	7

C9=3

Old C9	R0.1	R0.2	R9
3	63		3
3	58	0	3
3	57	0	3
3	56	0	3
3	63		4
3	56	0	5
3	54	0	6
3	52	0	7

	...	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	New C9
					R2							OUT R0.2					OUT R0.1		
C0:	...	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	0
C0:	...	47	48	49	50	51	52	53	54	55	56	57	58	0	0	0	0	0	1
C0:	...	47	48	49	50	51	52	53	54	55	56	57	0	0	0	0	0	0	2
C0:	...	47	48	49	50	51	52	53	54	55	56	0	0	0	0	0	0	0	3
C0:	...	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	4
C0:	...	47	48	49	50	51	52	53	54	55	56	0	0	0	0	0	0	0	5
C0:	...	47	48	49	50	51	52	53	54	0	0	0	0	0	0	0	0	0	6
C0:	...	47	48	49	50	51	52	0	0	0	0	0	0	0	0	0	0	0	7
C3:						1	2	3	4	5	6								
R3:																			

Compatibility CRTC 3 & 4

Old C9	R0.1	R0.2	R9
3	58	0	4

	...	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	New C9
					R2														
C0:	...	47	48	49	50	51	52	53	54	55	56	57	58	0	0	0	0	0	4

C9=4

Old C9	R0.1	R0.2	R9
4	63		4
4	57	0	4
4	56	0	4
4	55	0	4
4	59	0	4
4	63		5
4	55	0	6
4	53	0	7

	...	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	New C9
					R2							OUT R0.2					OUT R0.1		
C0:	...	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	0
C0:	...	47	48	49	50	51	52	53	54	55	56	57	0	0	0	0	0	0	1
C0:	...	47	48	49	50	51	52	53	54	55	56	0	0	0	0	0	0	0	2
C0:	...	47	48	49	50	51	52	53	54	55	0	0	0	0	0	0	0	0	3
C0:	...	47	48	49	50	51	52	53	54	55	56	57	58	59	0	0	0	0	4
C0:	...	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	5
C0:	...	47	48	49	50	51	52	53	54	55	0	0	0	0	0	0	0	0	6
C0:	...	47	48	49	50	51	52	53	0	0	0	0	0	0	0	0	0	0	7
C3:						1	2	3	4	5	6								
R3:																			

Compatibility CRTC 3 & 4

Old C9	R0.1	R0.2	R9
4	57	0	5

	...	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	New C9
					R2														
C0:	...	47	48	49	50	51	52	53	54	55	56	57	0	0	0	0	0	0	5

C9=5

Old C9	R0.1	R0.2	R9
5	63		5
5	59	0	8
5	59	0	7
5	59	0	6
5	59	0	5
5	58	0	5
5	63		6
5	54	0	7

	...	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	New C9
					R2							OUT R0.2					OUT R0.1		
C0:	...	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	0
C0:	...	47	48	49	50	51	52	53	54	55	56	57	58	59	0	0	0	0	1
C0:	...	47	48	49	50	51	52	53	54	55	56	57	58	59	0	0	0	0	2
C0:	...	47	48	49	50	51	52	53	54	55	56	57	58	59	0	0	0	0	3
C0:	...	47	48	49	50	51	52	53	54	55	56	57	58	59	0	0	0	0	4
C0:	...	47	48	49	50	51	52	53	54	55	56	57	58	0	0	0	0	0	5
C0:	...	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	6
C0:	...	47	48	49	50	51	52	53	54	0	0	0	0	0	0	0	0	0	7
C3:						1	2	3	4	5	6								
R3:																			

Compatibility CRTC 3 & 4

Old C9	R0.1	R0.2	R9
5	56	0	6

	...	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	New C9
					R2														
C0:	...	47	48	49	50	51	52	53	54	55	56	0	0	0	0	0	0	0	6

C9=6

Old C9	R0.1	R0.2	R9
6	63		6
6	59	0	9
6	59	0	8
6	59	0	7
6	59	0	6
6	58	0	6
6	57	0	6
6	63		7

				R2								OUT R0.2					OUT R0.1		New C9
C0:	…	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	0
C0:	…	47	48	49	50	51	52	53	54	55	56	57	58	59	**0**	0	0	0	1
C0:	…	47	48	49	50	51	52	53	54	55	56	57	58	59	**0**	0	0	0	2
C0:	…	47	48	49	50	51	52	53	54	55	56	57	58	59	**0**	0	0	0	3
C0:	…	47	48	49	50	51	52	53	54	55	56	57	58	59	0	0	0	0	4
C0:	…	47	48	49	50	51	52	53	54	55	56	57	58	0	0	0	0	0	5
C0:	…	47	48	49	50	51	52	53	54	55	56	57	0	0	0	0	0	0	6
C0:	…	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	7
				C3:	1	2	3	4	5	6									
										R3									

Compatibility CRTC 3 & 4

Old C9	R0.1	R0.2	R9
6	55	0	7

				R2															New C9
C0:	…	47	48	49	50	51	52	53	54	55	**0**	0	0	0	0	0	0	0	7

C9=7

Old C9	R0.1	R0.2	R9
7	63		7
7	59	0	10
7	59	0	9
7	59	0	8
7	59	0	7
7	58	0	7
7	57	0	7
7	56	0	7

				R2								OUT R0.2					OUT R0.1		New C9
C0:	…	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	0
C0:	…	47	48	49	50	51	52	53	54	55	56	57	58	59	**0**	0	0	0	1
C0:	…	47	48	49	50	51	52	53	54	55	56	57	58	59	**0**	0	0	0	2
C0:	…	47	48	49	50	51	52	53	54	55	56	57	58	59	**0**	0	0	0	3
C0:	…	47	48	49	50	51	52	53	54	55	56	57	58	59	0	0	0	0	4
C0:	…	47	48	49	50	51	52	53	54	55	56	57	58	0	0	0	0	0	5
C0:	…	47	48	49	50	51	52	53	54	55	56	57	0	0	0	0	0	0	6
C0:	…	47	48	49	50	51	52	53	54	55	56	0	0	0	0	0	0	0	7
				C3:	1	2	3	4	5	6									
										R3									

13.4 CRTC 2

R0 accepte toutes les valeurs sans que cela pose de problème aux autres compteurs.

Cependant, une des particularités de ce CRTC est d'inhiber beaucoup de traitements durant la HSYNC. Voir chapitres 15 (page 132) et 14 (page 120).

En particulier :

- Une VSYNC FANTÔME est générée lorsque C4=R7 si cela se produit durant la HSYNC. Voir chapitre 15, page 132.
- La gestion de remise à 0 de C4 (lorsqu'on est sur la dernière ligne d'un frame) est désactivée dans certaines conditions. Voir chapitres 0 (page 88) et 9.3.2 (page 71).
- La condition de désactivation du border (lorsque C0=0) n'est également plus gérée, et le BORDER de la ligne précédente reste actif. Voir chapitre 14, page 120.

Ce CRTC affiche 1 octet (0.5µsec) de BORDER avant que C0=0, mais contrairement au CRTC 0, qui fait la même chose, la fonction R8 SKEW DISP n'existe pas pour contourner ce problème. On ne peut donc pas (à ma connaissance) empêcher l'apparition de 1 octet de BORDER.
Voir chapitre 19, page 169.

Le contenu de R12/R13 est "transféré" dans CRTC-VMA' lorsque C0 atteint R1.
Puis CRTC-VMA' est transféré dans CRTC-VMA en début de frame.
Ce qui implique qu'il est impossible de changer d'offset si C0 n'atteint pas R1.
Voir chapitre 14, page 120.

Changer l'offset implique donc qu'au moins une microseconde de BORDER existe avant que C0=R0, afin que C0 puisse être égal à R1.

Le traitement de C4 et C9 sur ce CRTC suit en partie la logique du CRTC 0, dans la mesure où il existe une notion d'armement de la remise à 0 ou de l'incrémentation de C4 sur la condition de « **Dernière Ligne** ». Le CRTC détermine si il est sur une dernière ligne sur la position C0=0 ou via une mise à jour de R4 et/ou R9 hors HSYNC et sous conditions (voir chapitre 12.4.1).
En fonction de l'état « **Dernière Ligne** », il arme la remise à 0 inconditionnelle de C9 (et de C4) pour la prochaine ligne, quelles que soient les valeurs qui sont programmées après dans R9 et R4 durant la ligne.
Cette notion à une importance capitale pour permettre la gestion d'une RVLL sur ce CRTC.

Lorsqu'il est nécessaire de créer les conditions permettant un changement d'offset à chaque « ligne », C4, R4, C9, R9 doivent tous être à 0 lorsque C0=0 et C0 doit avoir rencontré R1 après que R12 et/ou R13 soient mis à jour.

Sans une petite feinte de sioux, C4 va déborder sur la deuxième ligne.
Voir chapitre 10.3.3, page 74.

Si R0 est positionné à 3, par exemple, le CRTC 2 va générer 16 « lignes » de 4 µsec.
Dans cette situation, il faut veiller à ce que R2 soit supérieur à 0 :

- Afin que la HSYNC débute obligatoirement lorsque C0>0 afin que la VSYNC soit prise en compte lorsque C4 atteint R7. Ceci peut cependant être « contourné ».

Afin que le BORDER soit désactivé (ou du moins ne reste pas activé) lorsque C0=0, ce qui est une contrainte plus gênante.

13.4.1 CAS PRATIQUE : RUPTURE VERTICALE LAST LINE (R.V.L.L.)

Le CRTC 2 collectionne un florilège de contraintes diverses qu'il est nécessaire de prendre en compte si on souhaite réaliser une rupture verticale...

Disons le tout de suite, effectuer une R.V.I. « traditionnelle » sur ce CRTC est impossible pour une raison simple : Pour que l'offset soit pris en compte, il faut que C0 atteigne R1 **sur la dernière ligne**. **Modifier R12/R13 après que C0>R1 ne permet pas une prise en compte pour la ligne suivante**. Or, la R.V.I. est basée sur le principe d'atteindre la dernière ligne dans le bord, et il faudrait donc qu'une micro-ligne « dans le bord » ait un C0 qui atteint R1 (lorsque C9=R9).

Mettre R1=1 lorsque R0=1 activera le BORDER à partir de C0=1.
Sauf à être capable de modifier R0 et R1 en même temps, plus grand chose ne sera affiché....

Il faut donc que C0 puisse atteindre R1 sur la ligne visible, qui doit également être la dernière du frame. **Cela tombe bien car la R.V.L.L. fonctionne sur ce principe !**

Le CRTC 2 n'est pas du tout gêné pour le comptage de C9 et C4 lorsque R0=0.
À ce titre il dispose du même potentiel de ruptures « cachées » qu'un CRTC 1, soit 14 ruptures de 1 µsec, ce qui aurait dû permettre de régler le problème de centrage rencontré sur CRTC 0, dont le potentiel de ruptures cachées est de 7.
Mais, malheureusement, ceci est possible uniquement si on peut activer l'état de dernière ligne sur une première ligne.

R0-1	R0-2	R9		R2										OUT R0-2					OUT R0-1	New C9
63		0	C0:	... 47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	0
58	4	1	C0:	... 47	48	49	50	51	52	53	54	55	56	57	58	**0**	1	2	3	4	1
57	2	2	C0:	... 47	48	49	50	51	52	53	54	55	56	57	**0**	1	2	0	1	2	2
57	1	3	C0:	... 47	48	49	50	51	52	53	54	55	56	57	**0**	1	0	1	0	1	3
55	1	4	C0:	... 47	48	49	50	51	52	53	54	55	**0**	1	0	1	0	1	0	1	4
58	0	5	C0:	... 47	48	49	50	51	52	53	54	55	56	57	58	**0**	0	0	0	0	5
57	0	6	C0:	... 47	48	49	50	51	52	53	54	55	56	57	**0**	0	0	0	0	0	6
56	0	7	C0:	... 47	48	49	50	51	52	53	54	55	56	**0**	0	0	0	0	0	0	7
			C3:	1	2	3	4	5	6												
				R3																	

Tout comme le CRTC 0, il existe une notion de « **dernière ligne** » permettant de programmer une remise automatique à 0 de C9 et C4. Cependant, activer cet état sur une première ligne est possible uniquement si ce même état est faux sur la dernière position C0 de la HSYNC.

Autrement dit, 2 premières lignes C9=C4=0 sont impossibles sans modifier R9 et/ou R4 durant la HSYNC pour permettre la prise en compte de l'état « **dernière ligne** ». Sinon, C4 déborde salement. Il est impossible de modifier R9 durant la HSYNC pour permettre la prise en compte de cet état et permettre cette remise à 0 (comme on peut le faire dans une rupture ligne à ligne) car si on souhaite modifier R9 durant la HSYNC, aucun truc magique ne permet de modifier R0 en parallèle pour créer les ruptures cachées.

Cependant, il est fort utile de rappeler ici que R9 est un registre 5 bits dont seuls 3 bits participent au pointeur vidéo. Lorsque la ligne C9=8 est affichée, on visualise la ligne C9=0.

Il est donc possible de gérer les contraintes de contiguïté des C9 en utilisant ce principe, comme on peut le voir sur le tableau suivant :

R0-1	R0-2	R9
55	0	8
54	0	9
53	0	10
52	0	11
51	0	12
50	0	13
49	0	14
48	0	15

			R2						OUT R0-2								OUT R0-1		New C9
C0:	...	47	48	49	50	51	52	53	54	55	**0**	0	0	0	0	0	0	0	**8 (0)**
C0:	...	47	48	49	50	51	52	53	54	0	0	0	0	0	0	0	0	0	**9 (1)**
C0:	...	47	48	49	50	51	52	53	0	0	0	0	0	0	0	0	0	0	**10 (2)**
C0:	...	47	48	49	50	51	52	0	0	0	0	0	0	0	0	0	0	0	**11 (3)**
C0:	...	47	48	49	50	51	0	0	0	0	0	0	0	0	0	0	0	0	**12 (4)**
C0:	...	47	48	49	50	0	0	0	0	0	0	0	0	0	0	0	0	0	**13 (5)**
C0:	...	47	48	49	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**14 (6)**
C0:	...	47	48	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**15 (7)**
C3:			1	2	3	4	5	6											
								R3											

La démonstration fournie avec SHAKER utilise un frame sans C9 identiques contigus utilisant les 8 blocs. La première et la dernière ligne utilisent cependant ce principe « C9 and 7 » afin d'éviter cette contiguïté (en créant une ligne 0 via C9=8 et une ligne 1 via C9=9).

Il convient toutefois de noter que pour atteindre C9=6 et C9=7 (via C9=14 et 15), il est nécessaire que R0 atteigne 49 et 48 respectivement. Ce n'est pas une situation idéale si les lignes visibles doivent être centrées.

Plus C0=R2 se produit tôt, plus les « ruptures cachées » apparaissent à gauche sur l'écran.

Les plus curieux pourront toujours aller jeter un œil dans l'intro de « AMAZING DEMO 2021 ».
Une méthode simple pour contourner la contrainte des lignes contigües consiste à alterner 1 ligne sur 2 la table de gestion des C9 à atteindre. Ainsi une ligne 1 est alors forcément suivie d'une ligne C9+8 et le problème est ainsi réglé.

Remarques : Dans la mesure où le CRTC 0 ne peut pas créer des ruptures cachées de 1 µsec, cela signifie que les premières données à apparaitre à gauche sont le 3ème octet de la ligne, suivi d'une ½ µsec de BORDER. Sur un CRTC 2, ce serait le 1er octet de la ligne (lorsque R0=0) suivi d'une ½ µsec de BORDER. Enfin sur les CRTC 1, 3, 4 dans le même contexte R2, ce serait 2 octets qui apparaitraient à gauche.

13.5 CRTC 3, 4

R0 accepte toutes les valeurs sans que cela pose de problème aux autres compteurs.

Si R0 vaut 0, alors C9 et R4 continuent d'être gérés normalement.

L'offset est modifiable selon les timings indiqués dans les schémas des pages suivantes, chapitre 13.7.

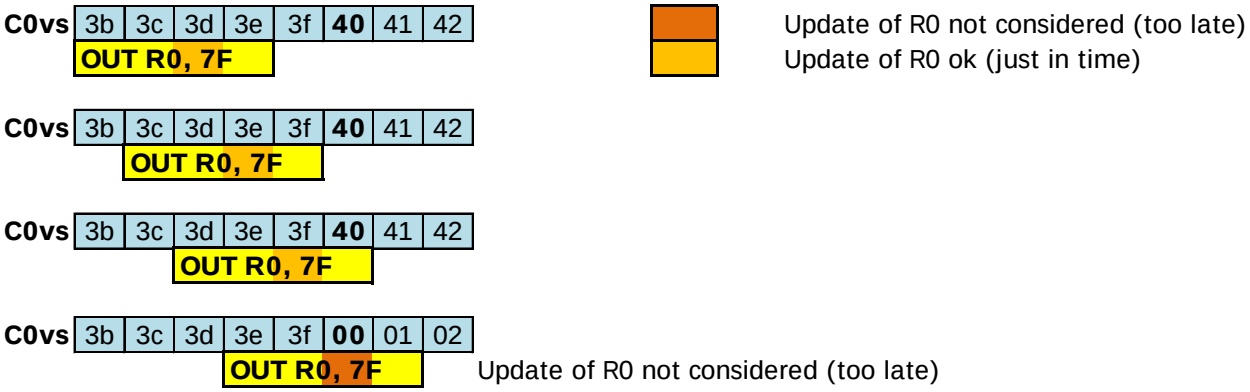
Les techniques de R.V.L.L. sont inapplicables sur ces CRTC. La principale « subtilité » de ces CRTC étant le passage de C9 à 0 si R9 est programmé est avec une valeur inférieure au C9 courant (qui permet une « certaine » compatibilité avec le CRTC 0). Il est possible de réaliser une R.V.I. avec la contrainte d'un bouclage de C9 à 0 pour que l'offset soit pris en compte. (voir chapitre 13.3.1). Attention néanmoins à ne pas oublier que l'I/O a lieu avec 1 µs de retard, ce qui implique de placer les OUT 1 µsec avant ceux qu'on ferait sur un CRTC 1.

13.6 MISE À JOUR DE R0

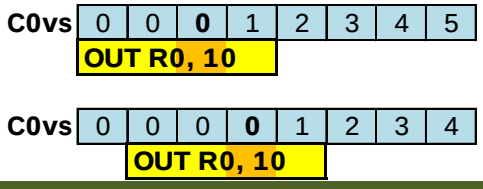
13.6.1 DÉLAIS DE PRISE EN COMPTE

CRTC 0, 1, 2

Previous R0 = #3f

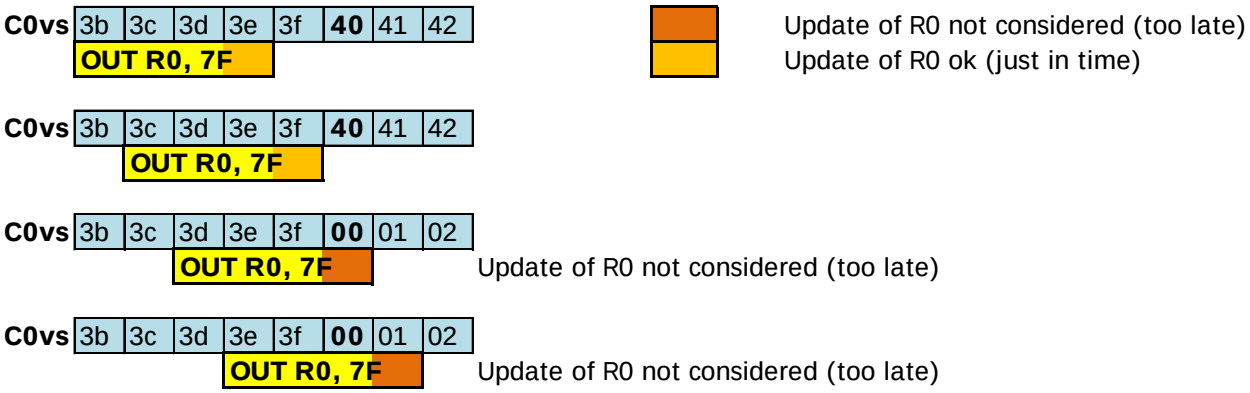


Previous R0 = 0

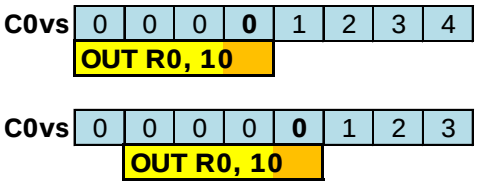


CRTC 3, 4

Previous R0 = #3f



Previous R0 = 0



La mise à jour du registre R0 est prise en compte immédiatement par le CRTC pour le comptage de C0. Le compteur C0 ne dépasse jamais la valeur de R0 lorsque R0 est mis à jour selon le timing décrit sur les schémas précédents.
Autrement dit, si R0=0 et que C0 vaut 0, il ne débordera pas, et C0 vaudra toujours 0.

En général, la valeur programmée dans R0 est 63 afin que 64 µsec s'écoulent avant que le compteur repasse à 0. Ce bouclage permet au compteur C0 de repasser sur la valeur de R2 dans une période de 64 µsec.

Lorsque R0 est reprogrammé durant le balayage d'une ligne, il faut prévoir de reprogrammer R2 pour que la HSYNC ait lieu au même endroit toutes les 64 µsec et qu'il n'y en ait qu'une seule de plus de 2 µsec.

En principe, il est plutôt conseillé de générer 1 HSYNC par ligne, mais un moniteur CTM supportera de ne pas en avoir 1 par ligne (bla, bla, hérétiques, bla, bla…, puristes, bla, bla,…).

Comme indiqué précédemment, sur le CRTC 0, la mise à jour de R0 avec une valeur inférieure à 2 empêche le CRTC de réaliser certains traitements spécifiques qui ont des conséquences sur le comptage.

Le délai minimum possible entre l'affection de R13 et R12 (dans cet ordre) est de 8 µsec.
Voir chapitre 23.1 page 228 217.

13.6.2 EXCEPTIONS

Il existe très certainement d'autres exceptions qui restent encore à établir dans une jolie table mais voici néanmoins quelques cas identifiés, selon les CRTC.

CRTC 1

Il existe une différence de prise en compte par le CRTC 1 de la modification de R0 sur une position précise de C0 selon l'instruction Z80A utilisée pour cette mise à jour. Les deux instructions concernées sont OUTI (I/O sur la 5ème µsec) ou OUT(C),reg8 (I/O sur la 3ème µsec). Ceci traduit une différence de délai de traitement interne au niveau des instructions Z80A.

C'est le cas lors de la mise à jour de R0 sur la position C0=0 lorsque C4=C9=0. Une RVI avec des ruptures de 1 NOP sur ce CRTC ne peut pas utiliser OUTI. Voir chapitre 13.3, page 105. Je n'ai pas (encore) identifié exactement ce qui se produit, mais cela semble avoir une action sur la désactivation du BORDER, et non sur le comptage de C0, n'ayant pas constaté de désynchronisation dans cette situation.

CRTC 0

Lorsque C0=0, le CRTC programme l'incrémentation ou le remise à zéro de C4 pour la prochaine ligne. Cette remise a 0 devient totalement effective partir de C0=2 lorsque le CRTC a évalué qu'il est sur une dernière ligne frame (C4=R4 et C9=R9) et qu'il n'y a pas de gestion additionnelle après cette dernière ligne (R5=0, R8=0). À noter que si une gestion additionnelle est en cours, la remise à 0 est programmée et se produira normalement à la fin de cette gestion.

Pour bien comprendre ce que je vais décrire, il faut avoir en tête que lorsque la condition C0=R0 se produit après C0=2 **dans le cadre d'une gestion additionnelle**, alors C4 est incrémenté quelle que soit la valeur de R4, et C9 passe à 0 lorsque C0 revient à 0. Le CRTC 0 fait l'économie d'un compteur spécifique C5 et utilise C9 lors de la gestion additionnelle.

Si R0 est programmé avec la valeur 1, alors le test de désactivation de gestion additionnelle n'est pas réalisé, et l'incrémentation de C4 reste programmée lorsque C9=R9 et C4=R4. Dans le cadre des délais de mise à jour de R0, cette gestion peut être **partiellement accomplie** si la mise à jour à lieu sur C0=1 alors que R0=1 et qu'on veut le modifier avec une valeur supérieure à 1 (par exemple pour refaire passer une ligne à 64 µsec...).

Si la mise à jour de R0 à lieu lorsque **C0=1 et C9=R9**, cela provoque un débordement de C4 sans que C0 et C9 soient remis à 0, **même si C4 n'est pas égal à R4**.
On est en gestion de ligne additionnelle :

- C4 est incrémenté inconditionnellement
- C9 ne peut plus repasser à 0, même lorsque C9=R9.

Le test d'évaluation C0=R0 (avec la valeur 1) a lieu avant que le registre soit mis à jour, ce qui provoque le « début d'une gestion additionnelle ». Mais le registre R0 a cependant été mis à jour assez tôt pour être pris en compte pour l'incrémentation de C0 tel qu'indiqué dans le chapitre précédent (voir chapitre 13.6.1) (dans l'exemple suivant, R0=59 avant le 1er OUT):

Screen Grid:		56	57	58	59	60	61	62	63	0	1	2	3	4	5	6	7
	C0:	...	56	57	58	59	**0**	1	0	1	0	**1**	2	3	4	5	6	7
R9=7	**C9:**	...	4	4	4	4	5	5	6	6	7	7	7	7	7	7	7	7
R4>=6	**C4:**	...	6	6	6	6	6	6	6	6	6	6	7	7	7	7	7	7

OUT R0,1 OUT R0,63

Dans cet exemple, deux situations sont possibles sur la ligne C9=7 lorsque R9=7 et lorsque C0 atteint R0 (qui vaut alors 63 grâce au second OUT).

13.6.2.1 On n'était pas sur une « dernière ligne frame» (C4<>R4, C9=R9)

La gestion additionnelle est désactivée d'office sur la position C0=2 (C4<>R4) car R5 vaut 0.
La conséquence est alors juste un débordement de C4 à partir de C0=2, sans que C4 repasse à 0 ensuite comme il l'aurait fait en gestion additionnelle.
À noter que **ce débordement n'aurait pas eu lieu si C9 avait été différent de R9.**
Dans l'exemple, C4 serait resté égal à 6.
Le contrôle du comptage C9/C4 reste classique.
Si, par exemple, il était nécessaire de compenser les deux « lignes » de 2 µsec créées après C0=59 afin que 8 lignes complètes soient affichées, il faudrait programmer R9 avec 9 pour ajouter 2 nouvelles lignes de 64 µsec.

13.6.2.2 On était sur une « dernière ligne frame » (C4=R4, C9=R9)

L'état dernière ligne a été activé avec C4=R4, C9=R9, quelle que soit la valeur de R4.
C'est vrai également si R4=0.
Comme dans le cas précédent, C4 est incrémenté à partir de C0=2.
La gestion additionnelle est évaluée sur la position C0=2.
Sur cette position, l'incrémentation de C4 sans que C9 repasse à 0 **laisse la gestion additionnelle activée**.
Si, par exemple, il était nécessaire de compenser les deux « lignes » de 2 µsec créées après C0=59 afin que 8 lignes complètes aient été affichées, **il faudrait programmer R5 avec 10** pour ajouter 2 nouvelles lignes 64 µsec. Si R5 n'est pas reprogrammé et valait 0, C9 va s'incrémenter pour afficher les lignes 8 à 31, jusqu'à ce qu'il atteigne R5-1.
Lorsque la gestion additionnelle est terminée, alors C4 repasse à 0.

Si on veut éviter que C4 soit incrémenté lors d'un « agrandissement » de R0 qui valait 1, **il faut donc le faire impérativement sur la position C0=0**.

13.7 OFFSET SELON C0

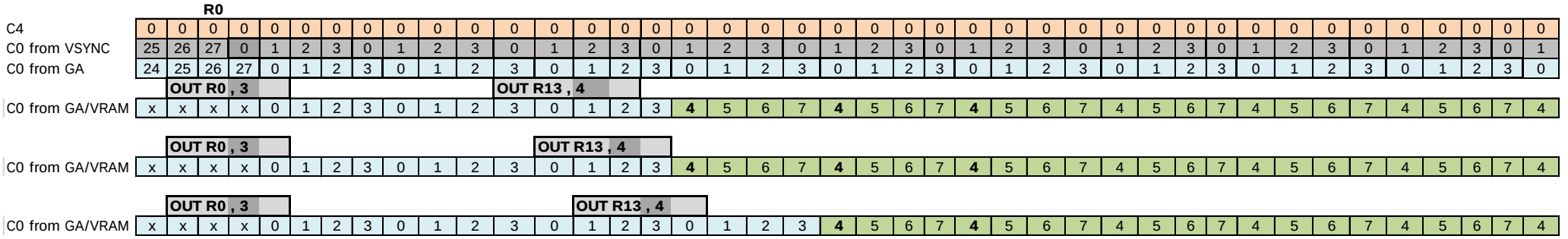
Les schémas suivants décrivent la prise en compte d'un changement d'offset (R12 et/ou R13) par rapport à la valeur courante de **C0vs**. Le CRTC 2 n'est pas représenté ici car ces tests ont été réalisés initialement avec une valeur de R1 supérieure à R0.

Les changements d'offset sont représentés avec la couleur verte.

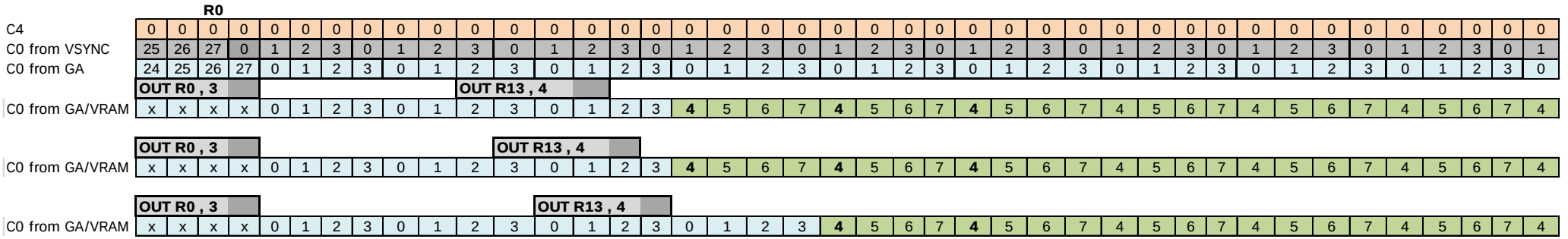
Données initiales :
CRTC-R4=0
CRTC-R9=0
CRTC-R1=4
CRTC-R13=0

13.7.1 ECRANS de 4 µsec (R0=3)

CRTC 0, 1

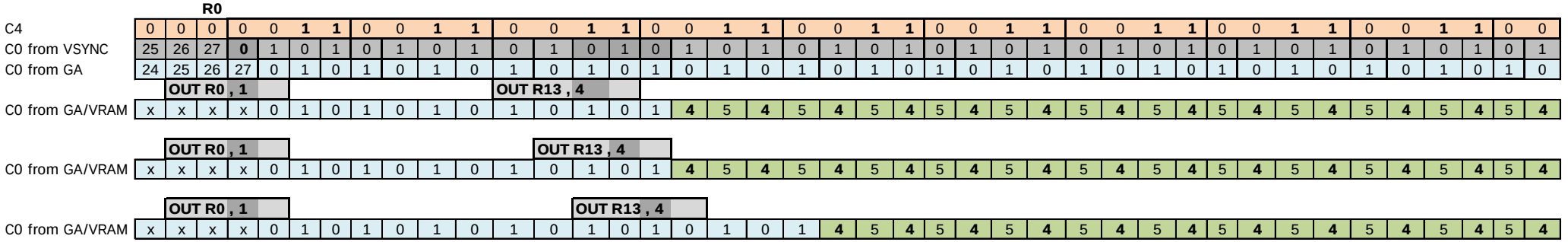


CRTC 3,4



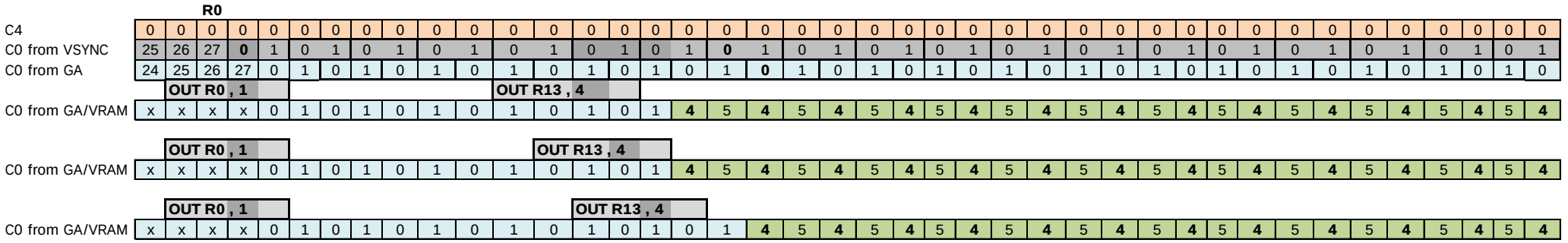
13.7.2 ECRANS de 2 µsec (R0=1)

CRTC 0

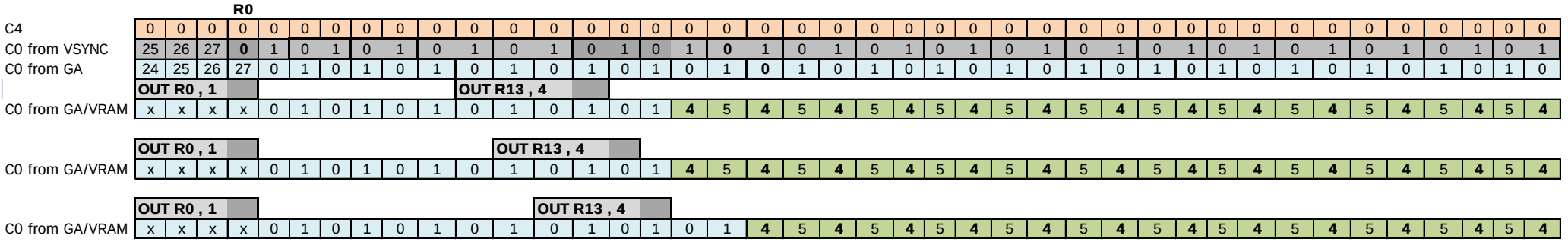


The event C0 = R0 after 2 µsec leaves C4 = 1 for the 2nd period of 2 µsec, which represents a vertical adjustment "not canceled" (because C0 is never equal to 2)

CRTC 1



CRTC 3, 4



13.7.3 ECRANS de 1 µsec (R0=0)

CRTC 0

	R0																																							
C4	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
C9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C0 from VSYNC	25	26	27	**0**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**0**	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
C0 from GA	23	24	25	26	**27**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		OUT R0 , 0																					**OUT R0 , 30**																	
C0 from GA/VRAM	x	x	x	x	x	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

R12 / R13 cannot be considered until C4 and C9 both go back to 0

CRTC 1

	R0																																							
C4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C0 from VSYNC	25	26	**27**	**0**	0	0	0	0	0	0	0	0	0	0	**0**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C0 from GA	24	25	26	27	0	0	0	0	0	0	0	0	0	0	0	**0**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		OUT R0 , 0										**OUT R13 , 4**																													
C0 from GA/VRAM	x	x	x	x	0	0	0	0	0	0	0	0	0	0	0	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**
		OUT R0 , 0												**OUT R13 , 4**																											
C0 from GA/VRAM	x	x	x	x	0	0	0	0	0	0	0	0	0	0	0	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**
		OUT R0 , 0													**OUT R13 , 4**																											
C0 from GA/VRAM	x	x	x	x	0	0	0	0	0	0	0	0	0	0	0	0	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**

CRTC 3,4

	R0																																							
C4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C0 from VSYNC	25	26	**27**	**0**	0	0	0	0	0	0	0	0	0	0	**0**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C0 from GA	24	25	26	**27**	0	0	0	0	0	0	0	0	0	0	0	**0**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	OUT R0 , 0											**OUT R13 , 4**																													
C0 from GA/VRAM	x	x	x	x	0	0	0	0	0	0	0	0	0	0	0	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**
	OUT R0 , 0											**OUT R13 , 4**																													
C0 from GA/VRAM	x	x	x	x	0	0	0	0	0	0	0	0	0	0	0	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**
	OUT R0 , 0													**OUT R13 , 4**																											
C0 from GA/VRAM	x	x	x	x	0	0	0	0	0	0	0	0	0	0	0	0	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**

14 SYNCHROS : REGISTRE R3

14.1 GÉNÉRALITÉS

Le registre 3 peut contenir 2 informations différentes, selon le modèle du CRTC 6845.

De manière générale, il permet de fixer :

- La durée de la HSYNC.
- La durée de la VSYNC pour certains CRTC.

Dans mes différents schémas, j'ai représenté la HSYNC à l'aide d'un compteur C3 qui débute à 1 sur C0=R2 et qui compte jusqu'à atteindre la valeur de R3. En interne, le circuit compte très certainement à partir de 0 et arrête la HSYNC lorsque C3=R3-1.

CRTC	7	6	5	4	3	2	1	0
0	Vsync	Vsync	Vsync	Vsync	Hsync	Hsync	Hsync	Hsync
1	x	x	x	x	Hsync	Hsync	Hsync	Hsync
2	x	x	x	x	Hsync	Hsync	Hsync	Hsync
3	Vsync	Vsync	Vsync	Vsync	Hsync	Hsync	Hsync	Hsync
4	Vsync	Vsync	Vsync	Vsync	Hsync	Hsync	Hsync	Hsync

Other CRTC	7	6	5	4	3	2	1	0
MC6845*1	Vsync	Vsync	Vsync	Vsync	Hsync	Hsync	Hsync	Hsync
UM6845E	Vsync	Vsync	Vsync	Vsync	Hsync	Hsync	Hsync	Hsync

Hsync CRTC				
0	0	0	0	**No Hsync**
0	0	0	1	1 nop
0	0	1	0	2 nop
0	0	1	1	3 nop
0	1	0	0	4 nop
0	1	0	1	5 nop
0	1	1	0	6 nop
0	1	1	1	7 nop
1	0	0	0	8 nop
1	0	0	1	9 nop
1	0	1	0	10 nop
1	0	1	1	11 nop
1	1	0	0	12 nop
1	1	0	1	13 nop
1	1	1	0	14 nop
1	1	1	1	15 nop

Hsync CRTC				
0	0	0	0	**16 nop**
0	0	0	1	1 nop
0	0	1	0	2 nop
0	0	1	1	3 nop
0	1	0	0	4 nop
0	1	0	1	5 nop
0	1	1	0	6 nop
0	1	1	1	7 nop
1	0	0	0	8 nop
1	0	0	1	9 nop
1	0	1	0	10 nop
1	0	1	1	11 nop
1	1	0	0	12 nop
1	1	0	1	13 nop
1	1	1	0	14 nop
1	1	1	1	15 nop

Vsync CRTC				
0	0	0	0	**16 lines**
0	0	0	1	1 line
0	0	1	0	2 lines
0	0	1	1	3 lines
0	1	0	0	4 lines
0	1	0	1	5 lines
0	1	1	0	6 lines
0	1	1	1	7 lines
1	0	0	0	8 lines
1	0	0	1	9 lines
1	0	1	0	10 lines
1	0	1	1	11 lines
1	1	0	0	12 lines
1	1	0	1	13 lines
1	1	1	0	14 lines
1	1	1	1	15 lines

14.2 LONGUEUR VSYNC

Les CRTC de première génération génèrent une VSYNC de 16 lignes.

Les constructeurs ont fait évoluer le circuit en ajoutant une fonction permettant de paramétrer le nombre de lignes, en utilisant les 4 bits de poids fort de R3.

Afin d'assurer une compatibilité des programmes créés pour la première génération de circuits, la valeur 0 pour les CRTC intégrant la nouvelle fonction correspond à 16 lignes de VSYNC.

L'intégralité des 4 bits de poids fort de R3 sert à indiquer un nombre de lignes exact, sauf pour 0 donc, qui signifie 16 lignes.

La ROM du CPC initialise VSYNC à 8 (R3=1000xxxx).
Ainsi, les CRTC 1 et 2, qui ne gèrent pas ces bits, génèrent 16 lignes de VSYNC alors que les CRTC 0, 3 et 4 en génèrent 8. (lorsque R7 est déjà programmé avant que C4=R7).

Étant donné que les CRTC 1 et 2 ne savent pas gérer une VSYNC paramétrable, **il est déconseillé d'utiliser cette fonction** sur les CRTC 0, 3 et 4 si l'objectif est la compatibilité.
Cela peut entrainer des incompatibilités pour un programme qui aurait la bonne idée de se synchroniser sur la fin de la VSYNC par exemple.

Cela implique de reprogrammer systématiquement ce registre car l'initialisation par la rom crée une différence qui pourrait être évitée.

Le jeu « 3D STARSTRIKE », édité par Realtime Games Software en 1985, a été développé sur le CRTC 0. Les développeurs n'ont pas modifié la valeur programmée en rom pour la longueur de la VSYNC. La conséquence est fâcheuse pour le jeu sur les CRTC 1 et 2, car le curseur de tir n'est pas en phase avec l'affichage et clignote, alors que ce n'est pas le cas sur les CRTC 0, 3 et 4.

14.3 HSYNC : GATE ARRAY VERSUS CRTC

Lorsque le GATE ARRAY reçoit un signal HSYNC de la part du CRTC, il arrête l'affichage pendant de 2 µsec (environ car il est possible de réduire cette période) et génère ensuite un signal HSYNC pour le moniteur dont la durée maximale est 4 µsec. Si la valeur programmée dans R3 est supérieure à 6, le GATE ARRAY affichera du noir pendant la période restante.

Si la HSYNC CRTC est supérieure à 6, cela n'a donc aucune incidence sur la synchronisation horizontale du moniteur. Avec une valeur inférieure à 6, la durée de la HSYNC pour le moniteur descend en-dessous des 4 µsec.

Le GATE ARRAY produit un signal de synchronisation composite utilisé par le moniteur.
Les signaux HSYNC et VSYNC produits par le CRTC sont fusionnés en 1 seul signal de synchronisation, qui est présent sur la broche 4 du connecteur vidéo DIN6.

Il peut être utile d'avoir une HSYNC « CRTC » supérieure à 6 car le CRTC ne prend pas en compte la mise à jour de tous ses registres de la même manière durant cette période et cela permet donc d'inhiber des traitements.

Remarque : Une HSYNC courte peut aussi avoir un effet colorimétrique sur certains écrans non CTM. C'est par exemple le cas si le CPC est branché sur un moniteur ATARI SC1425 (qui était livré avec l'ATARI 520STF). Comme indiqué plus haut, si la HSYNC est taillée à 6 µsec (R3=6), le GATE ARRAY envoie du noir pendant 2 µsec environ suivi du signal HSYNC de 4 µsec. Si la couleur du BORDER a été définie avec autre chose que du noir (que génèrerait une HSYNC plus longue que 6) cela a une incidence colorimétrique sur la ligne car chaque composante couleur est calibrée selon la couleur présente sur les 3 µsec qui suivent (position 7, 8 et 9). Si le BORDER n'est pas noir et que R3<9, les couleurs seront affectées sur la ligne. Sur ce type de moniteur, il est donc possible d'obtenir plus de couleurs.

14.4 HSYNC ET POSITION ECRAN

Il est possible d'utiliser la taille de la HSYNC-GA pour décaler le frame de environ 1/2 caractère CRTC sur l'écran (~ 1 octet soit 0.5 µsec). Ce "mécanisme" étant analogique, le résultat dépend du réglage du moniteur CTM et de sa tolérance.

La réduction de la taille de la HSYNC peut produire, sur d'autres écrans que les CTM, des variations de couleur ou des résultats imprévus et même créer l'apparition impromptue d'acronymes de modes graphiques.

Les valeurs utilisées en général, sans trop affecter la synchronisation, sont 5/6 ou 4/5.

Une valeur de 5 provoque un décalage de l'image à droite par rapport à une valeur de 6 (valeur maximale), quelle que soit la valeur de R2.

Dans la même logique, une valeur de 4 provoque un décalage de l'image à droite par rapport à une valeur de 5.

Remarque : Sur le jeu "*Skatewars*", Jon MENZIES utilise les valeurs &85 ,&8E, ce qui revient aux valeurs 5/6 pour la HSYNC-GA.

Une alternance continue (et consommatrice en CPU) de ces valeurs entre les lignes peut permettre de réduire la taille du décalage au 1/2 octet ou de créer de jolies trames.

Dans ce contexte, il est possible de faire bien mieux, puisque la technique **R3.JIT** permet de décaler la fin de la HSYNC de 0.25 µsec, permettant ainsi d'obtenir davantage de combinaisons. Le scroll au pixel en mode 1 n'est pas une idée farfelue…

14.5 HSYNC ET INTERRUPTIONS

La modification de la taille de la HSYNC modifie le moment ou une interruption est générée.
Le GATE ARRAY arme une interruption juste après la fin de la HSYNC, sous certaines conditions.
Voir chapitre 26, page 257

14.6 MISE À JOUR DE R3 DURANT LA HSYNC

Il est possible de modifier la valeur de R3 lorsque C3 compte, ce qui peut avoir une incidence sur la longueur de la HSYNC.

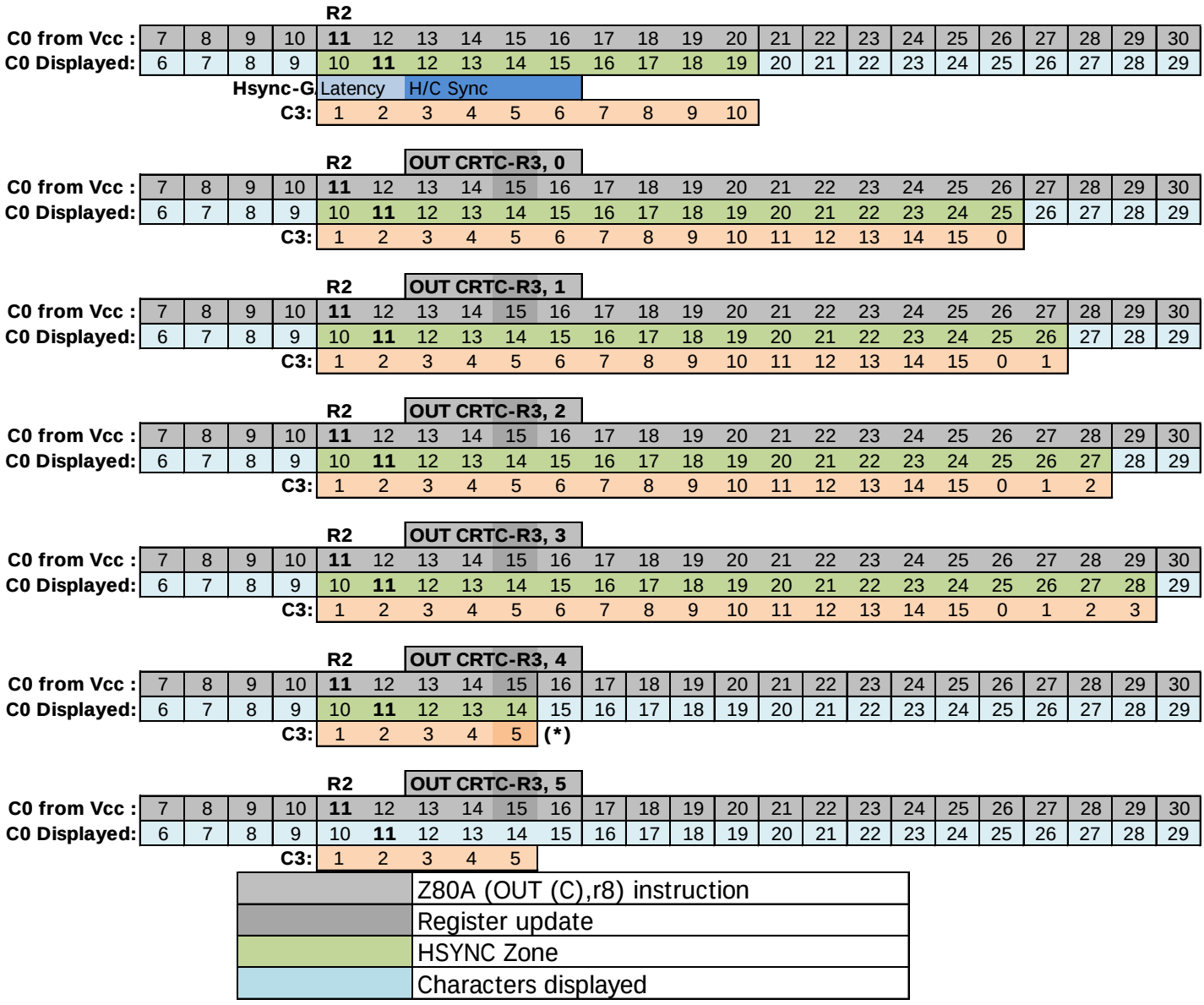
Si R3 est modifié avec une valeur inférieure à C3-1, alors C3 est en débordement, **sauf pour le CRTC 1 avec la valeur 0, qui annule la HSYNC en cours**.

Si R3 est modifié avec la valeur de C3-1 au moment ou C0 est sur la position C3, alors la HSYNC est interrompu sauvagement. Appelons donc cette technique **R3.JIT**.

La mise à jour de R3 durant ce décompte est prise en compte selon les situations décrites ci-après

14.6.1 CRTC 0, 2

CRTC-R2=11 / CRTC-R3=10 (HBL Size = 10 chars)



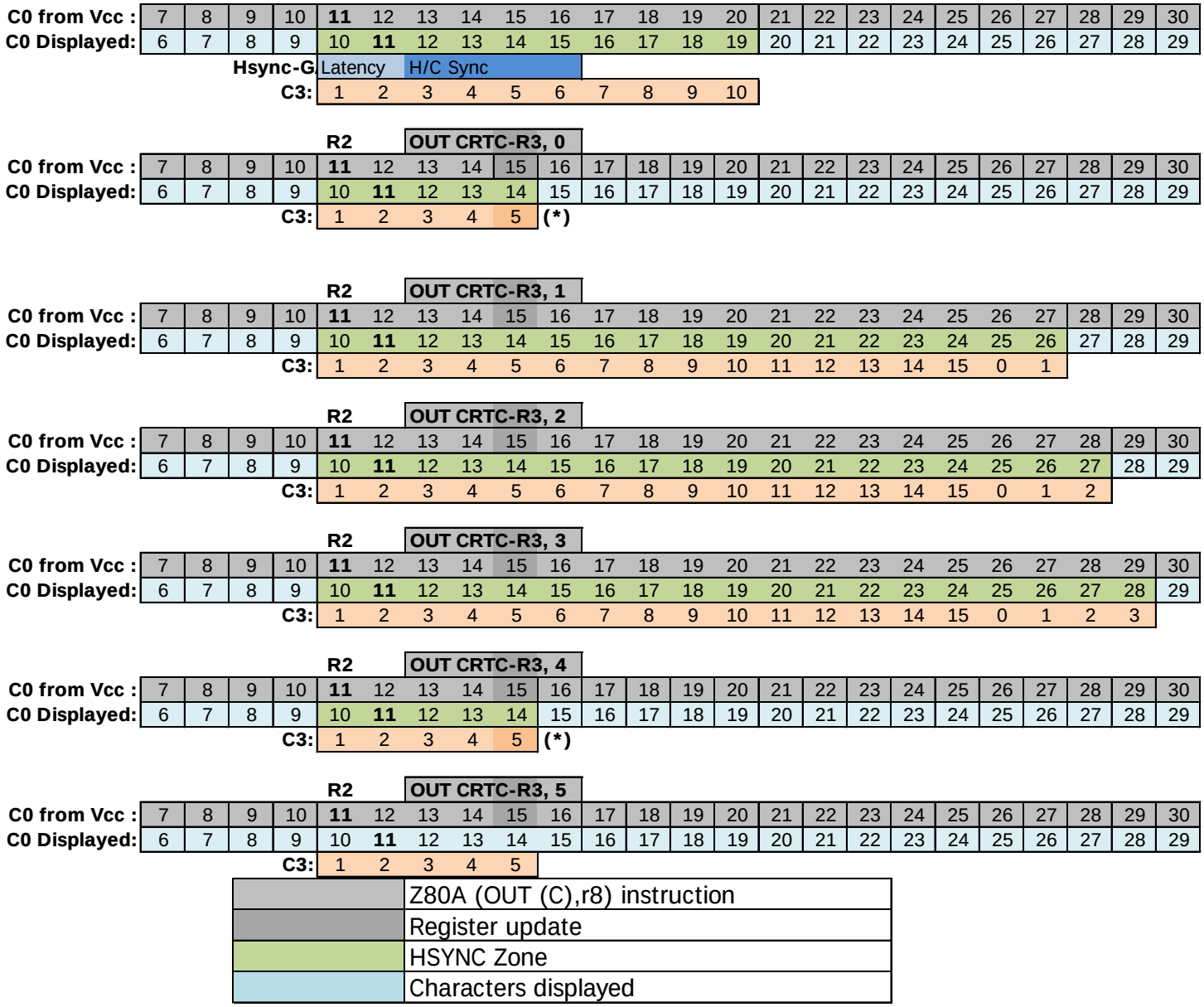
(*) R3.JIT : La mise à jour précise de R3 interrompt la HSYNC après 0.25 µsec après sa fin normale si R3 avait été programmé à l'avance. Voir chapitres suivants.

14.6.2 CRTC 1

CRTC-R2=11 / CRTC-R3=10 (HBL Size = 10 chars)

Durant une mise à jour de R3 pendant la HSYNC le CRTC 1 se comporte comme les CRTC 0 et 2, sauf lorsque R3 est mis à jour avec la valeur 0. En effet le traitement de R3=0 (absence de HSYNC) continue à être géré durant la HSYNC pour ce CRTC.

Sur les CRTC 0 et 2 dans cette condition, la HSYNC se poursuit et 0 est traité comme une valeur à atteindre.



(*) R3.JIT. La mise à jour précise de R3 interrompt la HSYNC après 0.25 µsec après sa fin normale si R3 avait été programmé à l'avance. Voir chapitres suivants.

14.6.3 CRTC 3, 4

CRTC-R2=11 / CRTC-R3=10 (HBL Size = 10 chars)

C0 from Vcc :	7	8	9	10	**11**	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
C0 Displayed:	6	7	8	9	10	**11**	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Hsync-GA :						Latency			H/C Sync																	
C3:						1	2	3	4	5	6	7	8	9	10											

R2					**OUT CRTC-R3, 0**																				
C0 from Vcc :	7	8	9	10	**11**	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
C0 Displayed:	6	7	8	9	10	**11**	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
C3:						1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0					

R2					**OUT CRTC-R3,1**																				
C0 from Vcc :	7	8	9	10	**11**	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
C0 Displayed:	6	7	8	9	10	**11**	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
C3:						1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0	1				

R2					**OUT CRTC-R3,2**																				
C0 from Vcc :	7	8	9	10	**11**	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
C0 Displayed:	6	7	8	9	10	**11**	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
C3:						1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0	1	2			

R2					**OUT CRTC-R3, 3**																				
C0 from Vcc :	7	8	9	10	**11**	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
C0 Displayed:	6	7	8	9	10	**11**	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
C3:						1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0	1	2	3		

R2					**OUT CRTC-R3, 4**																				
C0 from Vcc :	7	8	9	10	**11**	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
C0 Displayed:	6	7	8	9	10	**11**	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
C3:						1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0	1	2	3	4	

R2					**OUT CRTC-R3, 5**																				
C0 from Vcc :	7	8	9	10	**11**	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
C0 Displayed:	6	7	8	9	10	**11**	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
C3:						1	2	3	4	5																

R2					**OUT CRTC-R3, 6**																				
C0 from Vcc :	7	8	9	10	**11**	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
C0 Displayed:	6	7	8	9	10	**11**	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
C3:						1	2	3	4	5	6															

	Z80A (OUT (C),r8) instruction
	Register update
	HSYNC Zone
	Characters displayed

14.6.4 ZOOM SUR R3.JIT

Si R3 est modifié via un **OUT(C),r8** avec la valeur de C3-1 au moment ou C0 est sur la position C3 alors que R3 était supérieur à cette valeur, alors la HSYNC est interrompue sauvagement sur les CRTC 0, 1 et 2. Appelons donc cette technique **R3.JIT**.

L'utilisation de OUTI ne permet pas d'utiliser cette technique sur ces CRTC.

Que ce soit avec OUT(C),r8 ou OUTI, cette technique ne fonctionne pas sur les CRTC 3 et 4, qui synchronisent la HSYNC avec l'affichage.

Lorsqu'une HSYNC débute, elle le fait à différentes positions selon les CRTC.
Voir chapitre 9.3.2.252 pour plus de précisions sur ce sujet.

La modification de R3 peut être réalisée sur toutes les positions de C0 durant lesquelles la HSYNC a lieu.

La première µseconde de HSYNC est particulière, car son interruption avec la valeur 0 interrompt prématurément la HSYNC au lieu de retarder sa fin de 0.25 µsec.

Remarque : Interrompre R3 avec 0 à l'aide d'un OUTI a simplement pour effet d'empêcher la HSYNC de débuter.

Cependant, la valeur R3=0 pour un CRTC 2 signifie que la HSYNC fera 16 µsec.

En **R3.JIT** avec R3=0, la fin de la HSYNC a lieu après le dernier **Pixel-M2** de la µsec en cours.
Remarque : Sur CRTC 1, en R3.NJIT (ou avec OUTI), la HSYNC se termine 1 Pixel-M2 plus tard que pour les CRTC 0 et 2.

Sur un CRTC 0, la HSYNC débute sur le 5ème pixel-M2 et dure **4 pixel-M2**.
Sur un CRTC 1, la HSYNC débute sur le 6ème pixel-M2 et dure **3 pixel-M2**.

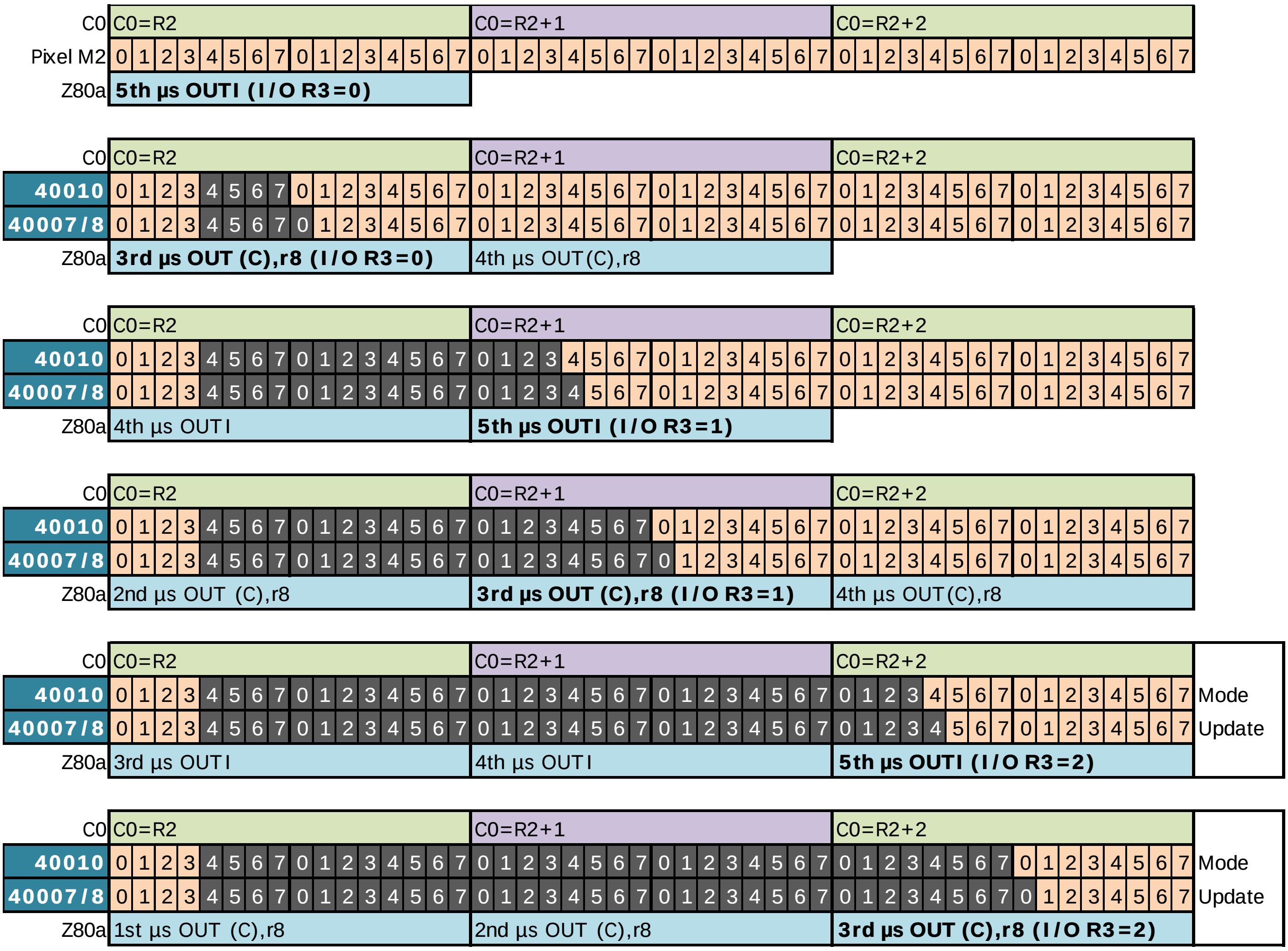
Les schémas suivants décrivent le positionnement de la HSYNC en R3.JIT.

La HSYNC est plus longue de 1 pixel-M2 sur les GA 40007 et 40008 par rapport au GA 40010.

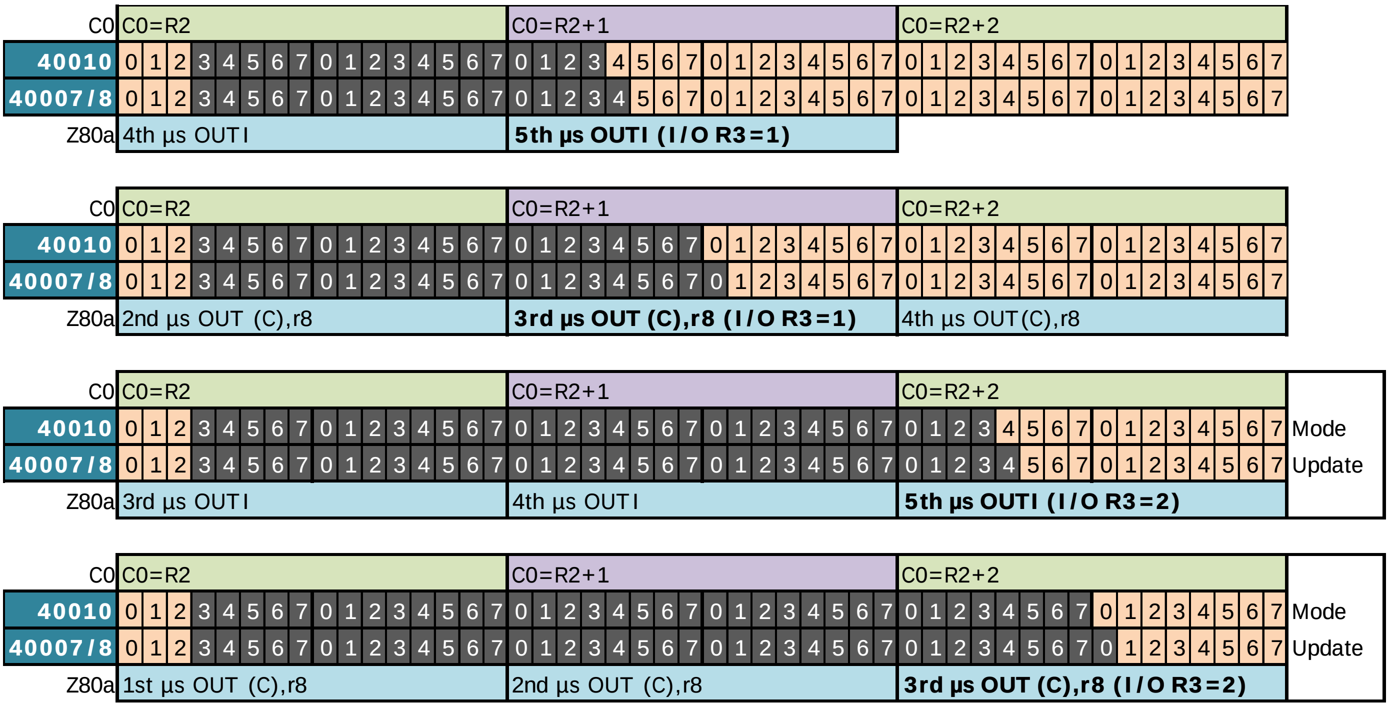
La possibilité de changement de mode est indiquée.

Il faut cependant tenir compte que passer du MODE 2 vers un autre MODE « ajoute » 1 pixel M2 (9 pixels sont générés à partir d'un octet) et passer d'un MODE 0.1.3 au MODE 2 « soustrait » 1 pixel M2 (7 pixels sont générés à partir d'un octet).
Voir chapitre 9.3.2, page 51.

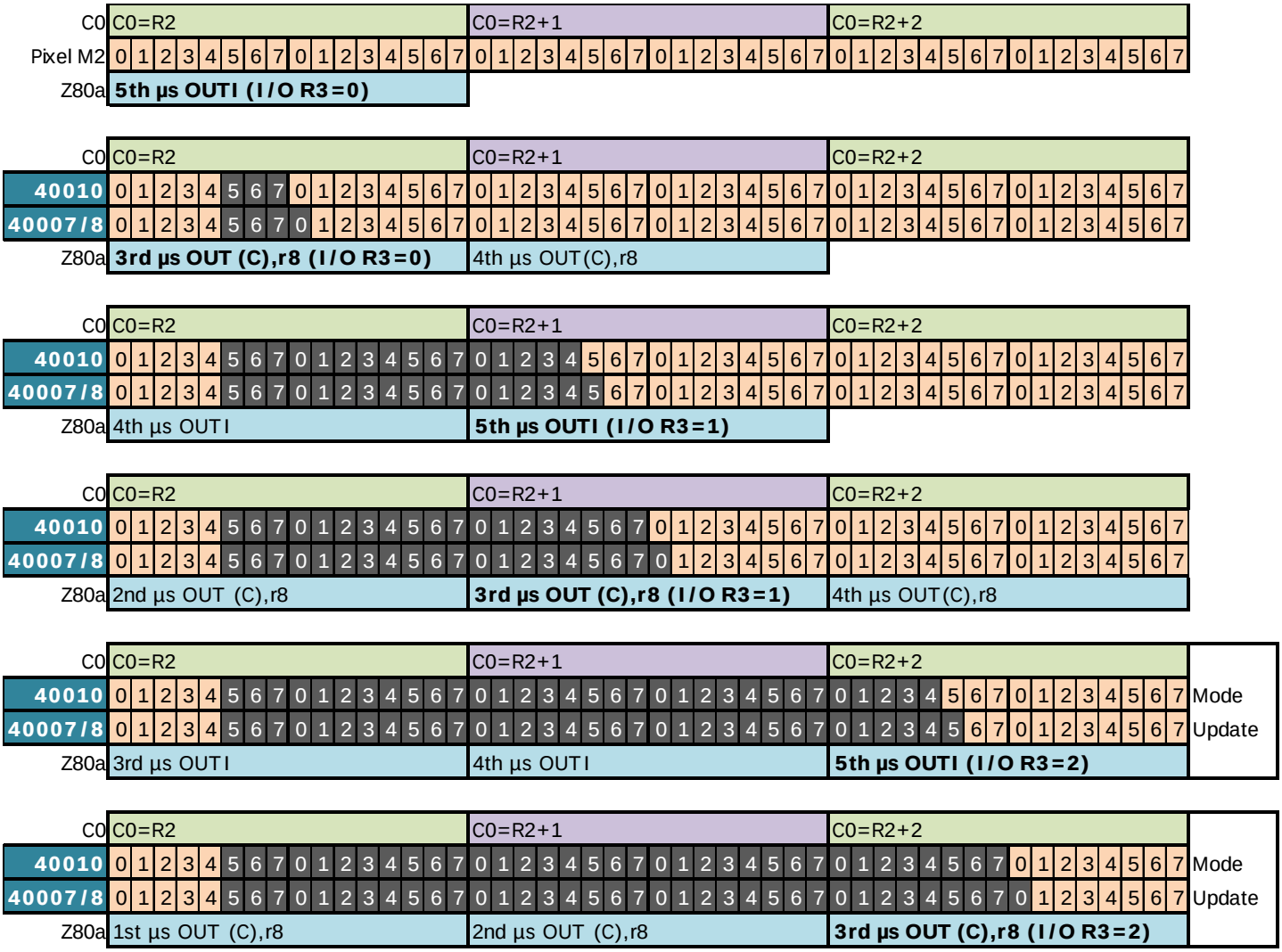
14.6.4.1 R3.JIT SUR CRTC 0



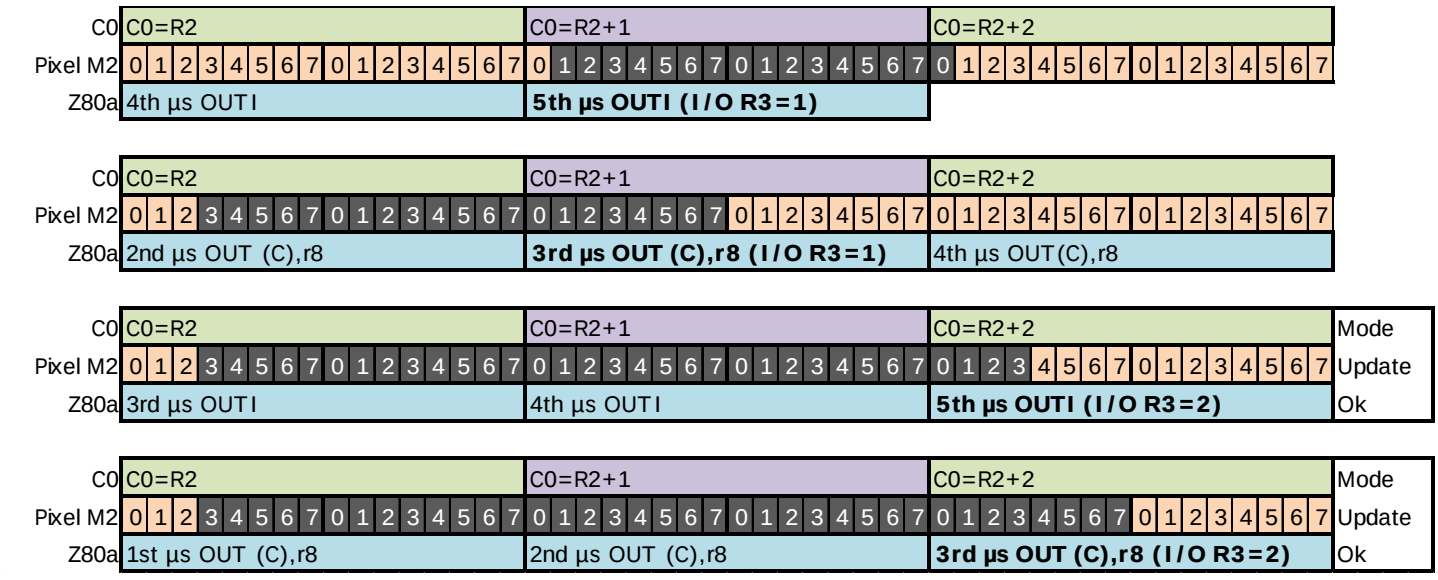
14.6.4.2 R3.JIT SUR CRTC 2



14.6.4.3 R3.JIT SUR CRTC 1



14.6.4.4 R3.JIT SUR CRTC 4



14.7 ABSENCE DE HSYNC

Lorsque R3=0, les CRTC 0 et 1 ne produisent pas de HSYNC (**et donc pas d'interruption**).

Sur CRTC 2, 3 et 4, il est impossible de ne pas générer de HSYNC si la condition C0=R2 est remplie. Une valeur de 0 dans R3 va générer une HSYNC de 16 µsec, sauf si elle est interrompue en modifiant R3 pendant la HSYNC.

14.8 DEMARRAGE HSYNC

14.8.1 CRTC 0, 1, 2

Lorsque C0=R2 alors une HSYNC est générée sur une longueur de R3 caractères CRTC.
La mise à jour de R2 a lieu pendant la 3ème µsec de l'instruction OUT(C),reg8.

Lorsque **la mise à jour de R2 a lieu avant que C0 atteigne R2**, la HSYNC est traitée par le GATE ARRAY pendant l'affichage du caractère précédent, le caractère C0=R2 n'ayant pas encore été affiché par le GATE ARRAY.

Dans cette situation, la zone noire HSYNC ne débute pas exactement sur une frontière de caractère, et pas à la même position selon les CRTC.

- CRTC 0 : Le non affichage lié à la HSYNC débute à partir du début du 5ème pixel mode 2 après le début du caractère CRTC R2-1 affiché. Il a été cependant observé l'affichage de la moitié du 4ème Pixel-M2.
- CRTC 1 : Le non affichage lié à la HSYNC débute à partir du début du 6ème pixel mode 2 après le début du caractère CRTC R2-1 affiché.
- CRTC 2 : Le non affichage lié à la HSYNC débute à partir du début du 4ème pixel mode 2 après le début du caractère CRTC R2-1 affiché. Il a été cependant observé l'affichage de la moitié du 3ème Pixel-M2.

Si la mise à jour de R2 intervient **pendant que C0=R2** (durant la 3ème µsec d'un OUT(C),r8) alors le CRTC envoie le signal HSYNC plus tardivement au GATE ARRAY. Cela traduit un délai entre l'exécution de l'instruction en Z80A et la prise en compte par le CRTC.

Cependant, si cette mise à jour est réalisée via l'instruction OUTI (durant la 5ème µsec de l'instruction) alors le signal HSYNC est envoyé plus rapidement au GATE ARRAY de 0.25 µsec, qui se comporte alors comme si R2 avait été programmé avant que C0=R2.

Une modification **R2.JIT** (« Just In Time ») réalisée avec un OUT(C),r8 provoque l'apparition de la zone noire HSYNC plus tard que dans les autres situations (R2 programmé avant C0=R2 ou i/o OUTI sur C0=R2).

Dans cette situation, la position d'arrêt de l'affichage dépend du type de CRTC :

- CRTC 0, 1 : Le non affichage lié à la HSYNC débute à partir du début du 9ème pixel mode 2 après le début du caractère CRTC R2-1 affiché. Il a été cependant observé l'affichage de la moitié du 8ème Pixel-M2.
- CRTC 2 : Le non affichage lié à la HSYNC débute à partir du début du 8ème pixel mode 2 après le début du caractère CRTC R2-1 affiché. Il a été cependant observé l'affichage de la moitié du 7ème Pixel-M2.

La durée de la HSYNC reste décomptée avec la valeur programmée dans R3 (sauf si R3 est modifié durant la HSYNC). L'affichage retardé de la zone noire HSYNC ne modifie pas la synchronisation du moniteur par rapport à une programmation de R2 anticipée.

Ce cas particulier de mise à jour (**R2.JIT**) permet de retarder le début de la HSYNC affichée de 4 pixels mode 2 (soit 0.25 µsec) (1 T-State) sur les CRTC 0 et 2, et 3 pixels mode 2 (0 .1875 µsec) sur CRTC 1.

Cela permet de retarder l'arrêt d'affichage par le GATE ARRAY de la zone noire de 0.1875 à 0.25 µsec. C'est intéressant en cas de changement de mode graphique en cours de ligne pour limiter la zone d'absence d'affichage. Il faut néanmoins tenir compte que le mode 2 est affiché 1 pixel (0.0625 µsec) plus tôt par le GATE ARRAY que pour les autres modes graphiques.

Sur les schémas pages suivantes, R3 est fixé à 2.
R2 avant modification est supérieur à 10.

14.8.2 CRTC 3, 4

Lorsque C0vs=R2 alors une HSYNC va être générée.

Cette mise à jour est prise en compte de manière à correspondre à l'affichage du caractère correspondant à C0 par le GATE ARRAY, mais le test est néanmoins réalisé par rapport à C0vs.

Une modification **R2.JIT** (« Just In Time ») réalisée avec un OUT(C),r8 ne provoque pas l'apparition de la zone noire HSYNC plus tard que dans les autres situations puisque la HSYNC est différée.

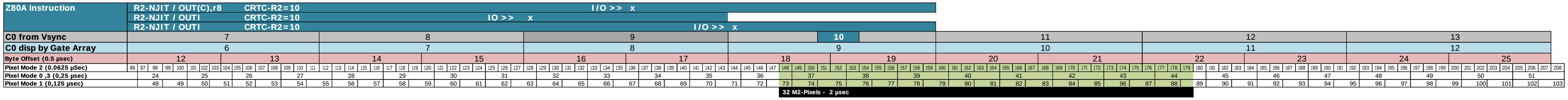
L'ASIC du CRTC 4 simule le GATE ARRAY, dans la mesure où l'affichage des pixels en mode 2 débute 1 Pixel-M2 avant l'affichage des pixels des autres modes graphiques.
Sur ce CRTC, l'arrêt d'affichage lié à la HSYNC débute à partir du 19ème pixel mode 2 après le début du caractère CRTC R2-1 affiché. Il a été cependant observé l'affichage de la moitié de ce 19ème pixel Pixel-M2.

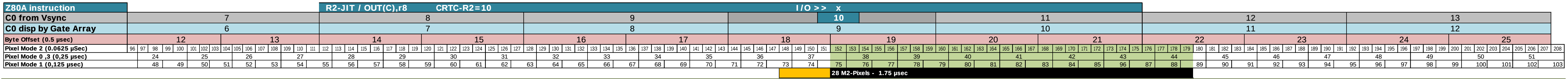
Pour rappel, une I/O via un OUT(C),r8 a lieu durant la 4ème µseconde de l'instruction sur les CRTC 3 et 4. Une I/O sur l'instruction OUTI a lieu durant la 5ème µseconde de l'instruction, comme pour les autres CRTC.

Les schémas ci-après illustrent les propos précédents en montrant le positionnement de la HSYNC selon le moment ou la mise à jour de R2 a lieu et selon l'instruction utilisée.

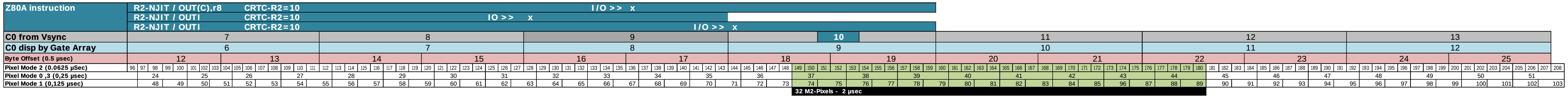
Le schéma CRTC 3 sera ajouté dans une version ultérieure. Il est actuellement supposé qu'il s'agit de la même chose que pour le CRTC 4, mais avec une HSYNC débutant sur le 17ème pixel après le début du caractère CRTC R2-1 affiché.

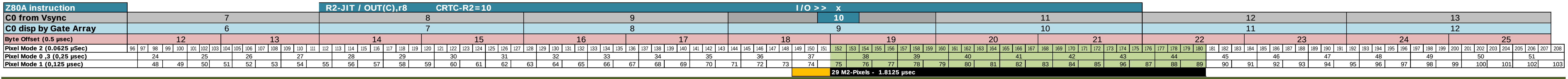
CRTC 0



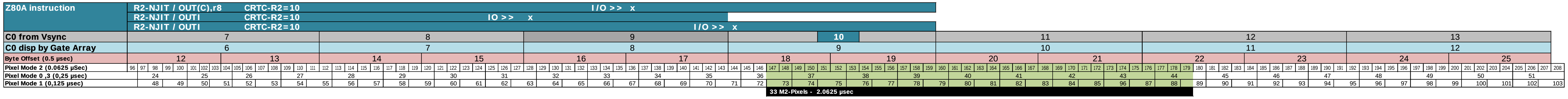


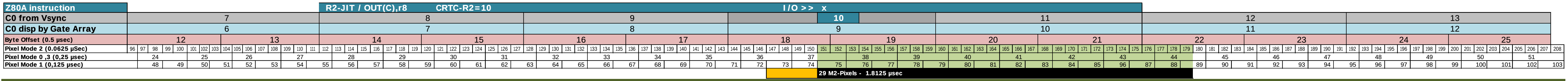
CRTC 1



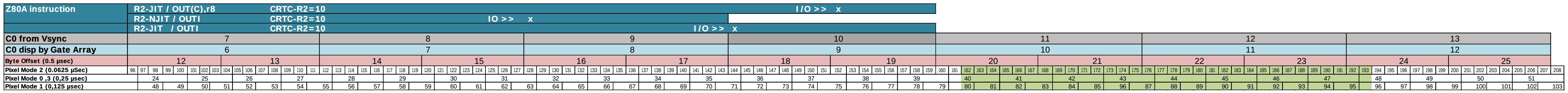


CRTC 2





CRTC 4



CRTC 3

to come.

15 SYNCHROS : REGISTRE R7

15.1 GÉNÉRALITÉS

Le registre R7 sert à fixer le moment ou le CRTC génère son signal VSYNC.
Ce signal est envoyé au GATE ARRAY.

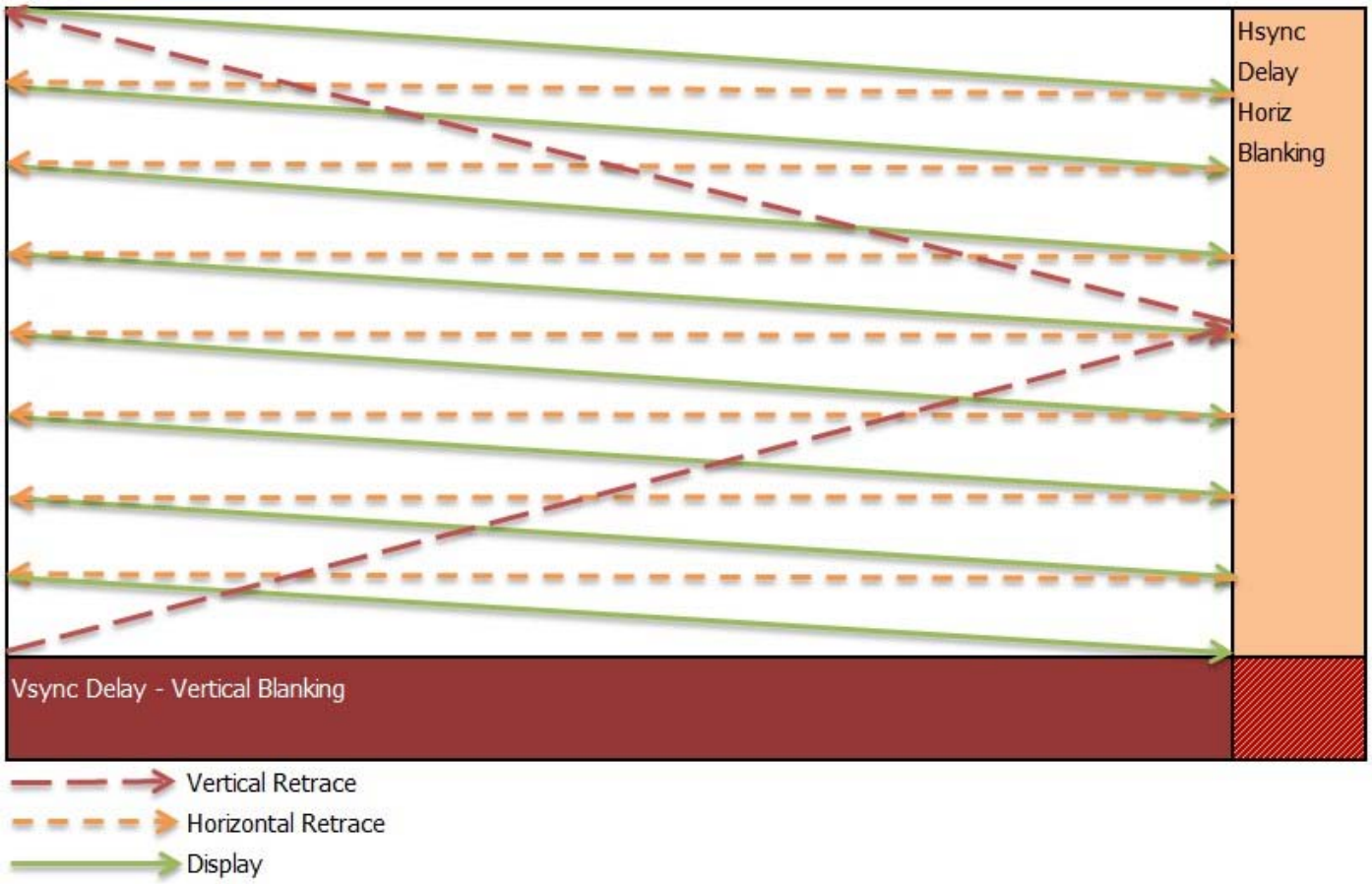
Ce signal est en général envoyé lorsque C4=R7.

Lorsque la VSYNC CRTC débute, le GATE ARRAY comptabilise le nombre de HSYNC rencontrées.

À la fin de la 2ème HSYNC, le GATE ARRAY envoie son signal VSYNC au moniteur.

Remarque : Ce comptage de HSYNC par le GATE ARRAY a lieu quelle que soit la taille de la HSYNC rencontrée. Si par exemple, R3=1 après le début de la VSYNC, les HSYNC de 1 µsec rencontrées lorsque C0=R2 mettront à jour le compteur HSYNC du GATE ARRAY.

C'est à partir du signal VSYNC envoyé par le GATE ARRAY que le faisceau du moniteur remonte en diagonale vers le haut en zigzaguant pour se repositionner sur le caractère ou le signal a été reçu du GATE ARRAY.



Le déflecteur vertical du moniteur descend constamment à vitesse constante. Bien que cela soit volontairement exagéré sur le schéma ci-dessus, le faisceau descend lorsqu'il affiche l'image de la gauche vers la droite. Cela ne se voit pas grâce à des réglages internes d'inclinaison. Ce phénomène est moins perceptible lors du retour du faisceau de la droite vers la gauche car le déflecteur horizontal va plus vite lors d'une HBL et la pente est moindre.

Durant la période VSYNC, le GATE ARRAY n'affiche pas de caractère mais le CRTC continue à gérer ses compteurs et pointeurs. Ce retour physique du canon à électrons par le moniteur est aussi appelé Vertical Blanking (VBL). À la différence de la période de latence pendant laquelle le GATE ARRAY génère une couleur noire, la période de VBL représente une absence réelle d'affichage.

La taille de la VSYNC est exprimée en nombre de HSYNC.
Ce nombre de HSYNC est programmable sur CRTC 0, 3 et 4 (via le registre R3) et est fixé à 16 pour les CRTC 1 et 2.

Comme pour les HSYNC, il est possible de générer plusieurs VSYNC durant un frame, mais le moniteur ne saura se caler que sur un signal VSYNC.

Il est impossible de déclencher ou inhiber une VSYNC durant une VSYNC. Ainsi, modifier la valeur de R7 avec une valeur de C4 atteinte durant la VSYNC ne provoque pas une nouvelle VSYNC. Modifier R7 avec une valeur de C4 différente de la valeur de C4 initiale n'interrompt pas la VSYNC courante.

Enfin, sur un moniteur CTM, l'image commence à être visible à partir de la 34ème ligne (Ce qui représente la deuxième ligne du 5ème caractère de 8 lignes à partir du début de la VSYNC).

15.2 CONDITIONS DE PRISE EN COMPTE

Pour tous les CRTC, la mise à jour de R7 avec C4 peut avoir lieu jusqu'à la dernière µseconde qui précède C4=R7. Autrement dit, si R7 est modifié avec la valeur de C4 sur la ligne C4-1, C9=R9 et C0=R0, alors la VSYNC sera active dès que C4=R7.

15.2.1 CRTC 0

La VSYNC CRTC débute lorsque C4=R7.
Si C4 devient égal à R7 lorsque C0vs atteint 0, alors la VSYNC débute sur C0vs=0.

15.2.1.1 Modification de R7

Si R7 est modifié avec la valeur de C4, alors la VSYNC est déclenchée immédiatement si elle n'était pas déjà en cours, **sauf si cette modification survient lorsque C0vs=0 ou C0vs=1**.

Si la modification de R7 avec la valeur de C4 a eu lieu lorsque C0vs<2, on est dans une situation de **VSYNC BLOQUÉE POUR C4**. Cela signifie que la VSYNC ne pourra plus se produire sur la valeur de C4=R7 tant qu'une situation de déblocage ne sera pas produite.

Pour les modifications R7=C4 qui surviennent lorsque C0vs>1, la VSYNC est "déclenchée" en cours de ligne.

Dans ce cas, le compteur de ligne débute à 0.
Ce compteur de lignes VSYNC est initialisé en début de ligne suivante lorsque C0=0.
N'importe quelle ligne de la VSYNC est concernée par ce démarrage en cours de ligne.
Par conséquent, la durée totale de la VSYNC est augmentée du nombre de µsec correspondant au calcul R0 - C0vs (du moment où R7 a été mis à jour).

Ainsi si une VSYNC est déclenchée en cours de ligne numéro 1, alors la VSYNC se termine à la fin de la ligne 17.

Exemples :

- Si R7 est modifié sur C0vs=#36, alors la prochaine lecture du port B du PPI (lue 6 µsec après sur C0vs=#3C) renvoie un statut actif de la VSYNC.
- Si R7 est modifié sur C0vs=#00, alors la prochaine lecture du port B du PPI (lue 6 µsec après sur C0vs=#06) renvoie un statut inactif de la VSYNC.

15.2.1.2 Modification de R0

Le compteur C0 a besoin d'atteindre la valeur 2 sur la ligne précédent celle ou C4=R7 pour qu'une VSYNC soit validée. Ainsi, si longueur de la ligne précédente est limitée à 0 ou 1 µsec (R0=0/1), la VSYNC sera bloquée pour C4=R7 de la ligne courante tant qu'une situation de déblocage ne se sera pas produite.

15.2.1.3 Déblocage de VSYNC CRTC

Lorsqu'une VSYNC est bloquée dans les deux cas de figures évoqués précédemment, il existe deux conditions de déblocage permettant à une VSYNC de se produire de nouveau :

- C4 doit changer de valeur. Il faut préciser ici qu'il ne s'agit pas d'un nouveau calcul C4 lorsque C9=R9 mais bien d'une modification de sa valeur. En effet, si on est dans le cadre d'une **VSYNC BLOQUEE** lorsque R4=0 (R7 a été modifié avec 0 lorsque C4=0 et C0=0 ou 1), alors C4 va rester à 0 à chaque fois que C9=R9, mais plus aucune VSYNC ne pourra se produire.
- C4<>R7 lorsque C9 repasse à 0 après avoir atteint R9. Ainsi, si C4=R4=0, la mise à jour de R7 avec une valeur différente de 0 débloquera la VSYNC.

Pour rappel, une VSYNC débutée ne peut plus être arrêtée avant son issue.

15.2.2 CRTC 1

La VSYNC débute lorsque C4=R7.

Si R7 est modifié avec la valeur de C4, alors la VSYNC est déclenchée immédiatement.
Si R7 est modifié sur C0vs=#36, par exemple, alors la prochaine lecture du PPI (au plus tôt 5 µsec après, donc sur C0=#3B) renvoie un statut actif de la VSYNC.

L'activation "déclenchée" de la VSYNC comptabilise la ligne comme si la VSYNC avait débuté en C0=0 de la ligne courante.

En conséquence, la durée totale de la VSYNC est réduite du nombre de µsec correspondant à la valeur de C0+1 du moment où R7 a été mis à jour.

Si une VSYNC est déclenchée en cours de ligne numéro 1, alors la VSYNC se termine à la fin de la ligne 16.

15.2.3 CRTC 2

La VSYNC est prise en compte sur toutes les valeurs de C0 et C9 lorsque C4=R7. Si la condition de la VSYNC a lieu durant une HSYNC de C0=R2 à C0=R2+R3 (1 µsec de plus que la taille visuelle de la HSYNC) alors le CRTC génère une **VSYNC FANTÔME**.

Si R7 est modifié avec la valeur de C4, alors la VSYNC est déclenchée immédiatement, sauf durant la période HSYNC (C0=R2 à C0=R2+R3) qui déclenche cette **VSYNC FANTÔME.**

Une **VSYNC FANTÔME** signifie que le CRTC comptabilise les lignes comme si une VSYNC avait lieu en empêchant une nouvelle VSYNC de se produire, mais sans que la broche VSYNC soit activée.

L'activation "déclenchée" de la VSYNC comptabilise la ligne comme si la VSYNC avait débuté en C0=0 de la ligne courante.

En conséquence, la durée totale de la VSYNC est réduite du nombre de µsec correspondant à la valeur de C0+1 du moment où R7 a été mis à jour.

Si une VSYNC est déclenchée en cours de ligne numéro 1, alors la VSYNC se termine à la fin de la ligne 16.

Pour contourner la problématique d'absence de VSYNC sur ce CRTC, il suffit juste d'éviter de générer une **VSYNC FANTÔME**. Ainsi, en positionnant R7 loin dans le cosmos (par exemple 127), il suffit ensuite de mettre à jour R7 avec C4 lorsque C0 n'est plus présent dans la période de HSYNC de la ligne considérée. Il est aussi possible de réduire R3 au dernier moment, mais c'est plus sportif. Dans ce cas, il ne faut pas oublier que si la HSYNC déborde sur C0=0, alors C4 déborde sur la dernière ligne du frame et le BORDER reste activé.

La **VSYNC** peut être différée lorsqu'un des modes « Interlace » est actif. Un retard d'une demi-ligne (R0/2) survient lorsque la parité d'un frame est paire.

15.2.4 CRTC 3, 4

La VSYNC débute lorsque C4=R7 et C9=C0=0.

Si R7 est modifié avec la valeur de C4 alors que C0>0, cela ne déclenchera pas de VSYNC CRTC.

15.3 VSYNC DIFFÉRÉE

15.3.1 CRTC 0

La VSYNC peut être différée lorsqu'un des modes « Interlace » est actif à partir de C4==R7.

- Un retard d'une demi-ligne lorsque la parité d'un frame est paire. La VSYNC se produit alors sur C4==R7 et C0=R0/2.
- Un retard d'une ligne complète dans le cas particulier ou R9 est impair en mode IVM (R8=3) sur un frame impair et un C4 impair. Voir chapitre 19.5.2

15.3.2 CRTC 1

La VSYNC peut être différée lorsqu'un des modes « Interlace » est actif à partir de C4==R7.

- Un retard d'une demi-ligne lorsque la parité d'un frame est paire. La VSYNC se produit alors sur C4==R7 et C0=R0/2.

Contrairement aux CRTC 0, 3 et 4, ce CRTC ne peut pas synchroniser correctement une image en mode IVM lorsque R7 est impair sur une image composée de caractères avec un nombre impair de lignes (R9 pair).

15.3.3 CRTC 2

La VSYNC peut être différée lorsqu'un des modes « Interlace » est actif à partir de C4==R7.

- Un retard d'une demi-ligne lorsque la parité d'un frame est paire. La VSYNC se produit alors sur C4==R7 et C0=R0/2.

15.3.4 CRTC 3 & 4

La VSYNC peut être différée lorsqu'un des modes « Interlace » est actif à partir de C4==R7.

- Un retard d'une demi-ligne lorsque la parité d'un frame est paire. La VSYNC se produit alors sur C4==R7 et C0=R0/2.
- Un retard d'un ligne complète dans le cas particulier ou R9 est impair en mode IVM (R8=3) sur un frame impair et un C4 impair. Voir chapitre 19.5.5

15.4 VSYNC SANS LIMITES !

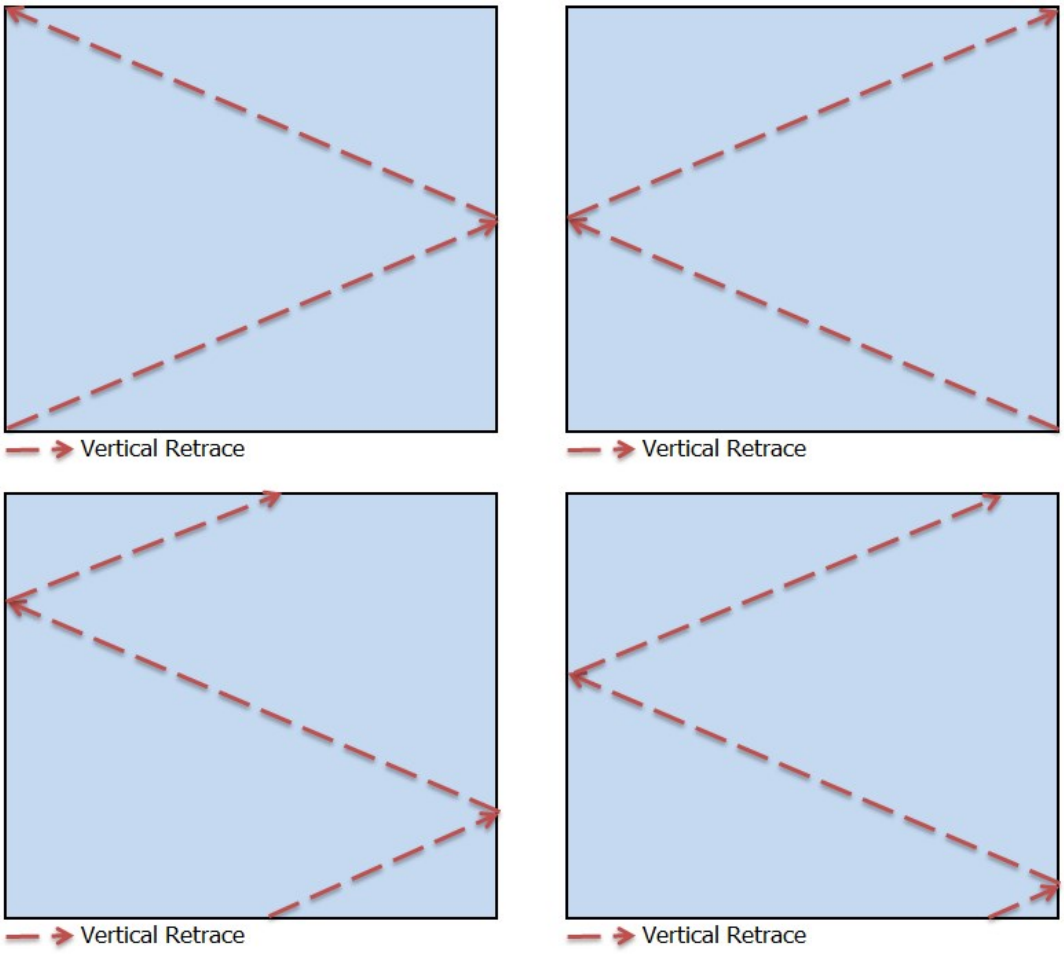
Dans un système sans GATE ARRAY, si le signal VSYNC d'un CRTC était envoyé directement au moniteur, ce dernier serait capable de débuter sa phase de retour vers le haut du moniteur pour satisfaire aux contraintes du mode interlacé.

Dans le cadre d'une image composée de 2 frame, le CRTC en mode « Interlace » génère un frame impair et pair de 312 lignes chacun, séparés par une ligne sur laquelle la VSYNC sera générée au milieu de la ligne (R0/2). Voir chapitre 19.3, page 175.

Malheureusement (ou heureusement, nous allons le voir) c'est le GATE ARRAY qui filtre les signaux HSYNC et VSYNC du CRTC et qui génère un signal mixte HSYNC+VSYNC vers le moniteur dès qu'il arrive sur la 2ème HSYNC qui suit la VSYNC générée par le CRTC. Il court-circuite ainsi le timing « natif » des CRTC pour le fonctionnement « Interlace ».

Il est malgré tout possible d'obliger le GATE ARRAY à envoyer le signal VSYNC au moniteur au bon moment si on souhaite obtenir une image « Interlace ». Il suffit pour ça de placer la 2ème HSYNC qui survient après la VSYNC CRTC à la moitié de la distance de la ligne par rapport à celle qui a lieu habituellement (soit un délai de R0/2 µsec). Il ne faut cependant pas oublier de compenser le déficit ainsi créé pour le moniteur. Il est donc possible d'afficher une image « Interlace » assez simplement.

Le faisceau remonte en diagonale pour rejoindre la position horizontale quittée en bas de l'écran. Selon la position horizontale du faisceau lorsque la demande est réalisée, ce dernier zigzague pour atteindre la position horizontale qu'il vient de quitter.
Les schémas suivants montrent les différentes situations qui peuvent se présenter.

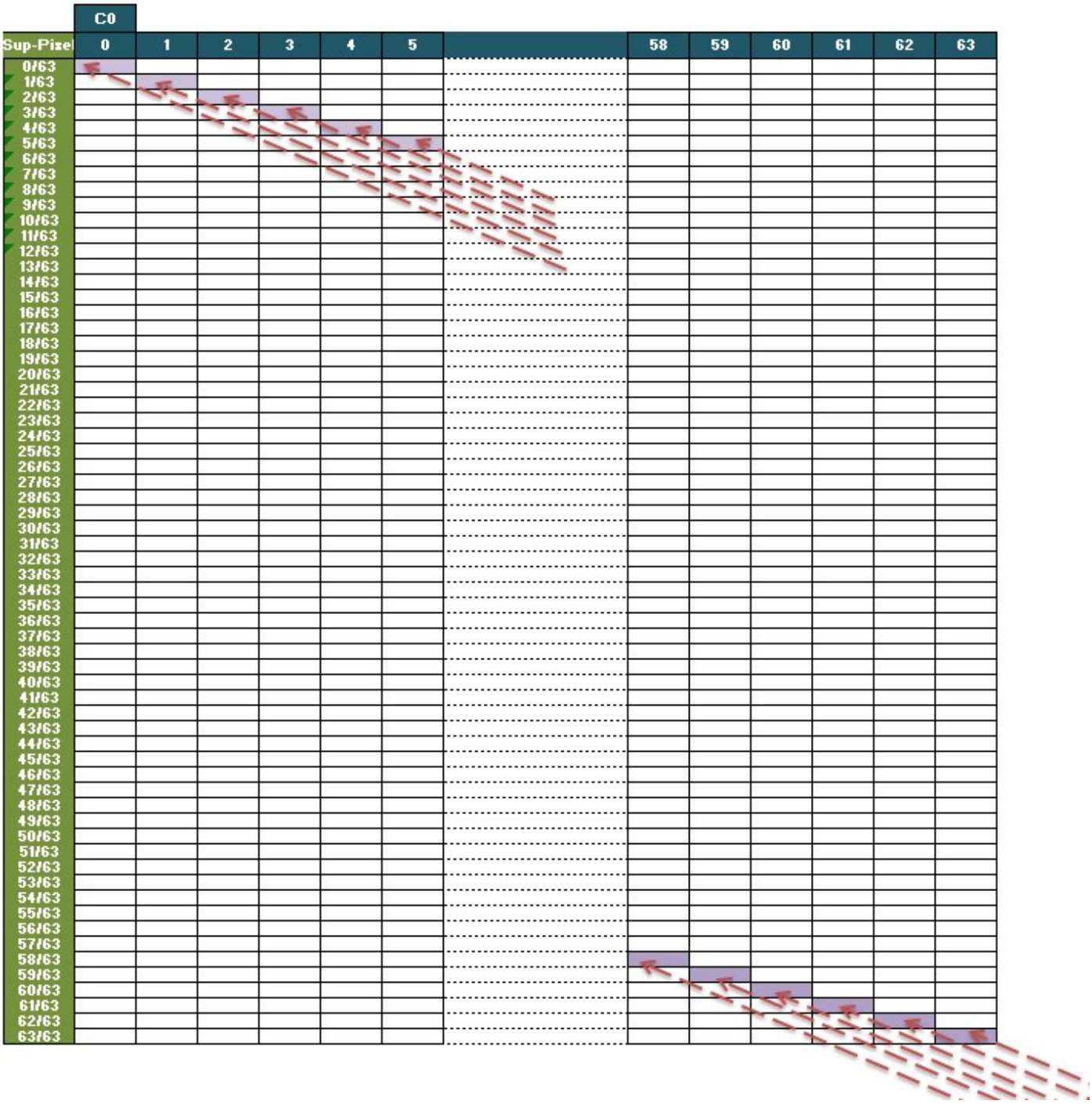


Plus le faisceau démarre tard, plus le faisceau remonte haut.

Ainsi, si le faisceau commence à remonter à partir du milieu d'une ligne, il va remonter :

- Plus haut de ½ pixel que si le faisceau était remonté à partir de la position C0=0.
- Plus bas de ½ pixel que si le faisceau était remonté à partir de la position C0=63.

Une idée (pas si farfelue que ça, dirait Cheshirecat) consiste à déclencher la 2ème HSYNC-VSYNC sur chacune des 64 positions du frame afin d'obtenir une précision de positionnement de 1/64ème de pixel.



Ce schéma montre, d'un point de vue « CRTC », le positionnement du faisceau au niveau vertical d'un pixel lorsqu'il est arrivé en haut de l'écran, en partant d'une VSYNC activée sur les différentes positions de C0 d'une ligne du frame.

Le CTM est assez précis pour gérer ça. Magie du CPC ! Meuglement du C64 et de l'Amiga!

L'oeil humain n'est cependant pas capable de voir le mouvement à ce degré de précision.
Ainsi, le CPC sur moniteur CTM est capable de gérer un scrolling vertical fluide au 1/64ème de pixel (et toutes les déclinaisons inférieures).

Ceci est démontré dans SHAKER à partir de la version 2.1.

Il devrait être possible d'augmenter encore cette précision à 1/128ème pixel, grâce à l'aide d'une autre technique décrite dans ce document…

15.5 LE BON MOMENT…

Si R7 est mis à jour sans « précautions », une nouvelle VSYNC peut survenir durant ou après le frame courant. Dans cette situation, le moniteur doit gérer plusieurs VSYNC ou l'absence de VSYNC après la mise à jour, et cela se traduit par des sauts d'images.

Le moniteur essaie simplement de caler son image sur la nouvelle position de C4=R7.
Cette opération peut avoir lieu plus ou moins vite selon le réglage « v-hold » du moniteur.

Il est possible d'éviter ce décrochage de synchronisation du moniteur en agissant pour que C4 repasse à la nouvelle valeur de R7 au même endroit que l'ancienne valeur de C4=R7. Cela nécessite de modifier R4 pour qu'il repasse à 0 plus tôt si le nouveau C4=R7 à atteindre est plus élevé que l'ancien, ou inversement d'agrandir R4 afin que C4 repasse à 0 plus tard si le nouveau C4=R7 à atteindre est moins élevé que l'ancien. Il faut juste imaginer que ce sont les compteurs qui viennent se placer où ils devraient se trouver.

Cette petite gymnastique de repositionnement des compteurs évite au moniteur de perdre la VSYNC, ce qui peut être ennuyeux pour un programme ne pouvant souffrir ce type d'artefact visuel.

À défaut, si votre code a le temps, il est toujours possible de dissimuler ce décrochage en agissant sur R1 et/ou R6, voir même en mettant toutes les encres noires un « certain temps », mais il existe quelques pervers qui titillent le V-Hold de leur moniteur afin de traquer les déviants insoumis du diktat des puristes intégristes.

16 SYNCHROS : REGISTRE R2

16.1 GÉNÉRALITÉS

Le registre CRTC R2 définit quand se produit la **HSYNC**.
La **HSYNC** se produit par rapport à la position du caractère **C0vs** (au sens CRTC).
Pendant la **HSYNC**, en principe, plus rien n'est affiché.

La longueur en µsec de la HSYNC-CRTC est fixée avec les 4 bits de poids faible du registre R3.
Voir chapitre 14.1, page 120.

Le moment ou se produit la HSYNC est différent selon les CRTC et n'est pas calé sur le début d'un caractère CRTC. Voir chapitre 14.4, page 122.

L'affichage de pixels ne débute pas (et ne s'arrête pas) sur une frontière de mot, voir d'octet (selon conditions) pour la HSYNC. Voir chapitre 14.8, page 129.

Le GATE ARRAY est plus rapide pour gérer la HSYNC que pour afficher les caractères lus par les CRTC 0, 1, 2. La HSYNC débute donc plus tôt et est "visible" sur le caractère précédent celui pointé par R2, qui n'a pas encore été affiché.

Les ASIC (CRTC 3 et 4) gèrent une HSYNC en cohérence avec la valeur de C0 affichée, en retardant l'affichage de la HSYNC de 1 µsec.

Exemples :

- Si R2 vaut 10, sur un CPC avec CRTC (0, 1, 2) alors la HSYNC débute à partir de C0 affiché=9 (R2-1) sur une longueur de R3 µsec.
- Si R2 vaut 10, sur un CPC avec "CRTC" (3, 4) alors la HSYNC débute à partir de C0 affiché=10 sur une longueur R3 µsec.

À cause de ce décalage, l'étalonnage d'un moniteur CM14 (464+/6128+) est différente de celui d'un CTM 640/644 (464/664/6128) pour les CRTC 0, 1 et 2.

Le CRTC 4 n'est pas livré avec un CM14, mais il se comporte comme un CPC+ au niveau du signal HSYNC. AMSTRAD calibrait les CTM livrés avec ce CPC pour que l'image soit centrée. Branché sur le CTM 640/644 d'un autre CPC (avec un CRTC 0, 1, 2), l'image est décalée à gauche car la HSYNC se produit 1 µsec plus tard. Afin d'assurer une compatibilité visuelle avec les autres CRTC, il est possible d'en tenir compte en programmant R2 avec une valeur inférieure de 1 à celle programmée pour les autres CRTC. Mais sans un menu de calibrage, il est impossible de savoir si le CRTC 4 est branché sur son moniteur d'origine.

Inversement, brancher un CPC CRTC 0, 1, 2 sur un moniteur CM14 ou le CTM d'un CRTC 4 doit provoquer un décalage à droite de l'image.

Sur un moniteur CTM, le premier caractère visible à gauche est le 15ème caractère à partir de la position C0=R2 si R3>5. (Si R2=49 (avec R0=63), le 1er caractère visible est C0=63).

Cette différence entre la prise en compte et l'affichage ne change rien au comportement de gestion du compteur du CRTC lorsque R2 et R3 sont modifiés.

Nous l’avons vu, le GATE ARRAY est plus lent pour afficher les caractères que pour prendre en considération le signal HSYNC, mais il existe également **un délai entre l'instruction du Z80A qui met à jour le CRTC et la vitesse de ce dernier pour en tenir compte**.

Ainsi, si R2 est modifié très exactement sur le caractère (CRTC) visé, le CRTC reçoit l'information légèrement plus tard, traduisant ici les délais internes propres à l’instruction OUT du Z80A. Cela permet de réduire légèrement la taille de la HSYNC « affichée » Voir chapitre 14.2, page 121.

Le CRTC dispose d'une période pendant laquelle il "accepte" de prendre en considération R2.

Le GATE ARRAY dans cette situation, reçoit le signal HSYNC légèrement plus tard, ce qui a un impact sur la longueur "**affichée**" de cette dernière.

Le traitement HSYNC débute avec le traitement HSYNC-GA, qui dure au maximum 6 µsec.
C'est sur la HSYNC-GA que le moniteur se synchronise pour chaque ligne.

Si cette HSYNC change de position ou de longueur (et si cette longueur dépasse 2 µsec) plusieurs fois en cours de balayage cela entraine une distorsion de l'image par le moniteur.

Pendant les 2 premières µsec le GATE ARRAY n'envoie pas de signal de synchronisation (H/Csync) au moniteur. La HSYNC-GA est dépendante de la HSYNC-CRTC (début et longueur):

- Elle débute en même temps que la HSYNC du CRTC.
- Elle s'arrête en même temps si la HSYNC du CRTC dure moins de 6 µsec.

Une HSYNC de 2 µsec ne provoque pas d'envoi de signal de synchronisation au moniteur.
Au-dessus de cette durée de 2 µsec, le moniteur va essayer de synchroniser la ligne affichée.

Des effets de distorsion peuvent apparaitre :

- Si la HSYNC est trop courte (>2 µsec et < 6 µsec).
- Si plusieurs HSYNC de longueur > 2 µsec sont générées sur une même ligne.
- Si la ou les HSYNC ne sont pas alignées verticalement.

Durant une HSYNC, les différents CRTC ne gèrent plus la condition C0=R2, ce qui interdit à une HSYNC de redémarrer au sein d'une HSYNC.

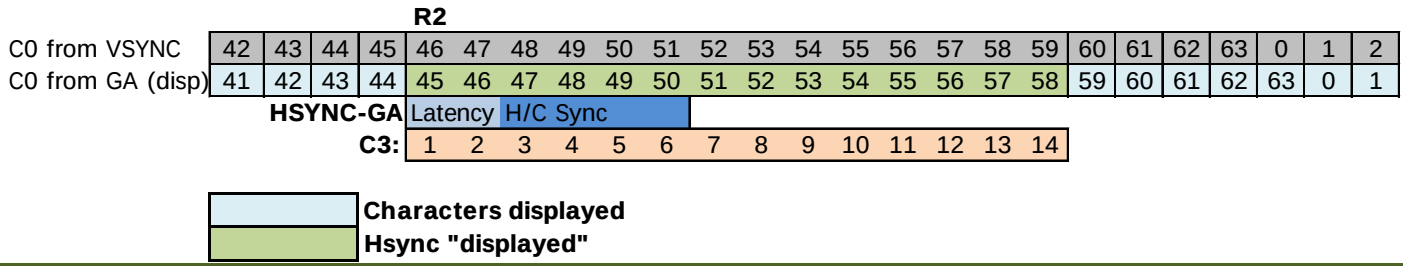
16.2 HSYNC LORSQUE R2 EST PRÉDÉFINI

Les schémas suivants montrent la génération de la HSYNC, telle qu'elle survient lorsque R2 a été programmé avant que C0vs=R2, ce qui est en principe le cas général.

Les caractères sont indiqués selon deux échelles, puisque la HSYNC est gérée sans « délai » par le GATE ARRAY.

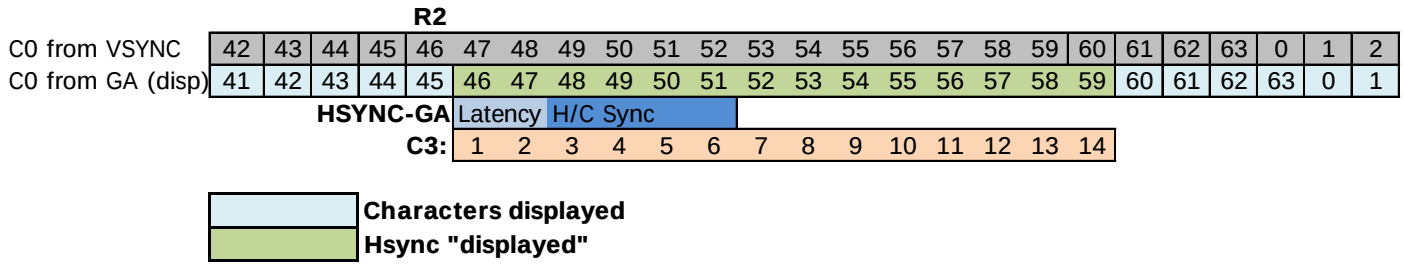
CRTC 0, 1, 2

CRTC-R2=46
CRTC-R3=14



CRTC 3, 4

CRTC-R2=46
CRTC-R3=14



16.3 MISE À JOUR DE R2 DURANT LA HSYNC

Durant le traitement de la HSYNC CRTC, une mise à jour de R2 n'est plus prise en compte si cette modification a pour objet de débuter une nouvelle HSYNC durant la HSYNC.

Remarque :

- Cette absence de prise en compte évite un plantage du circuit, qui ne génèrerait plus que des HSYNC dans la situation où R0 serait inférieur à R3. C'est notamment le cas lorsque R0 est inférieur à R3, ce qui implique que C0 peut repasser plusieurs fois sur la même valeur (égale à R2).
- Sur le CRTC 0, deux HSYNC ne peuvent pas être collées, alors qu'il y a un bug de gestion sur les autres CRTC dans ce cas précis (ils évaluent mal C3=R3).

HSYNC INFINIE :

Le bug de gestion de R2 sur les CRTC 1, 2, 3, 4 permet de créer une HSYNC infinie (et au-delà).

Si, par exemple, on place R0=0, R2=0 et R3=1, on va demander au CRTC de générer une HSYNC de 1 µsec à la position C0=R2=0, et lui demander de faire une HSYNC toutes les µsec.

Sur le 2ème caractère, C0 est encore égal à R2.

Sur un CRTC 0, la HSYNC n'aura pas lieu. Elle se produira sur le 3ème C0=0.

Sur les autres CRTC, la HSYNC ne se termine pas et C3 va déborder.
C3 va s'incrémenter jusqu'à 15, revenir à 0 puis à 1.

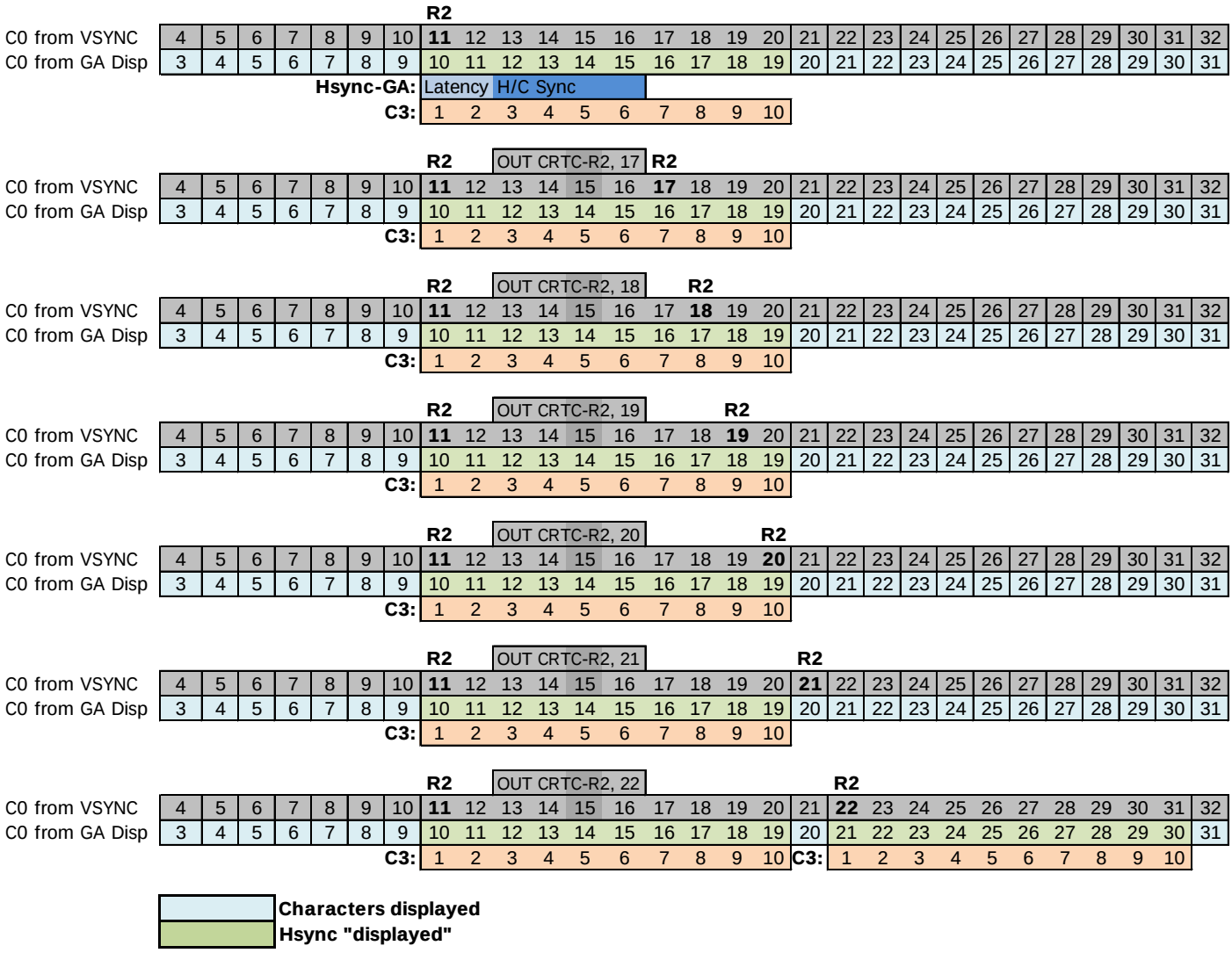
À la fin du débordement de C3, si C0 est encore égal à R2 (=0), alors la HSYNC va de nouveau continuer sa route, et C3 déborder de nouveau, et ainsi de suite.

Les différences de prise en compte de R2 selon les CRTC sont détaillées sur les schémas ci-après et décrivent la mise à jour de R2 sur différentes valeurs de **C0vs** durant la HSYNC, et l'impact sur cette dernière.

CRTC 0

CRTC-R2=11

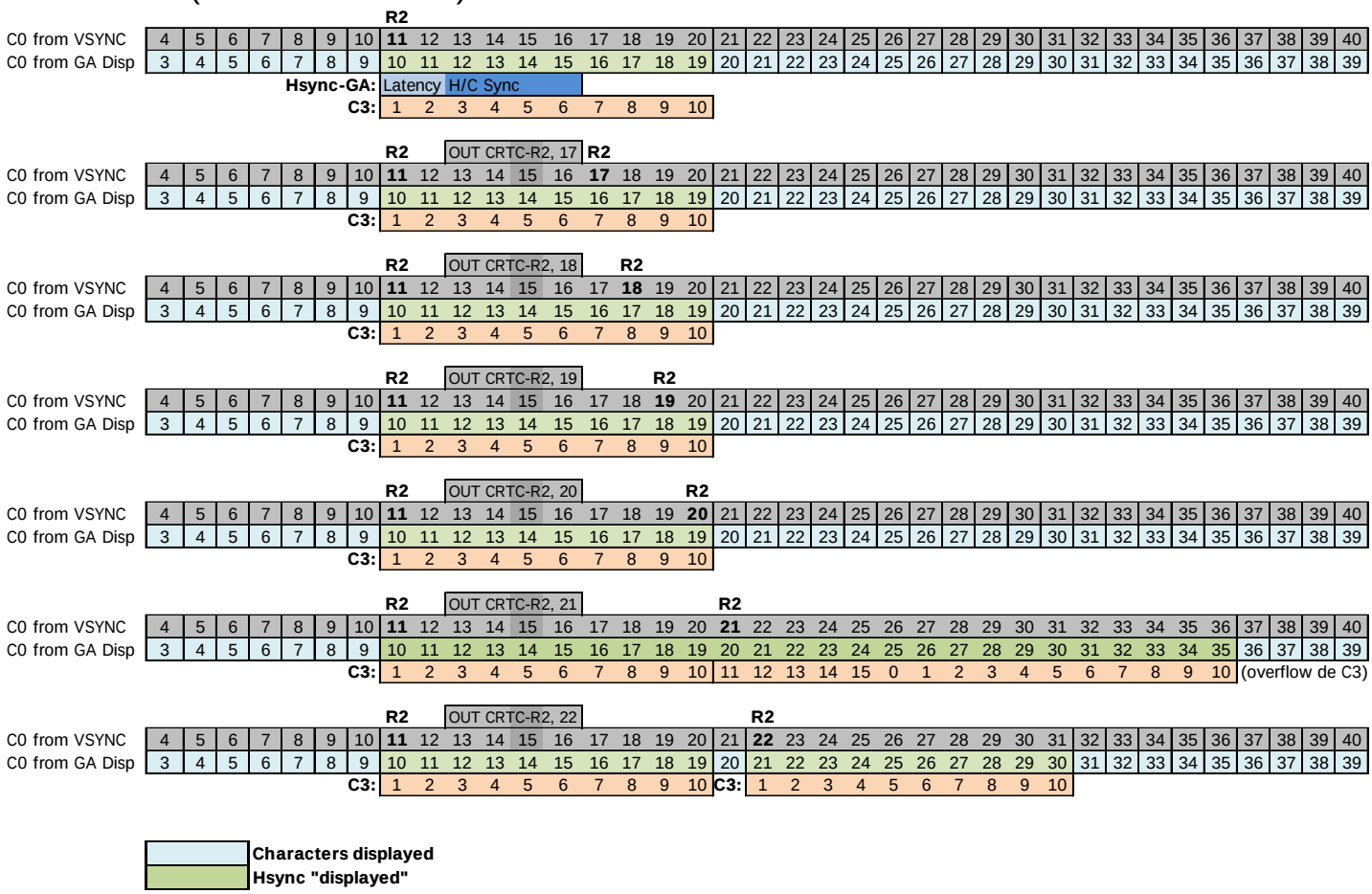
CRTC-R3=10 (HBL Size=10 chars)



CRTC 1, 2

CRTC-R2=11

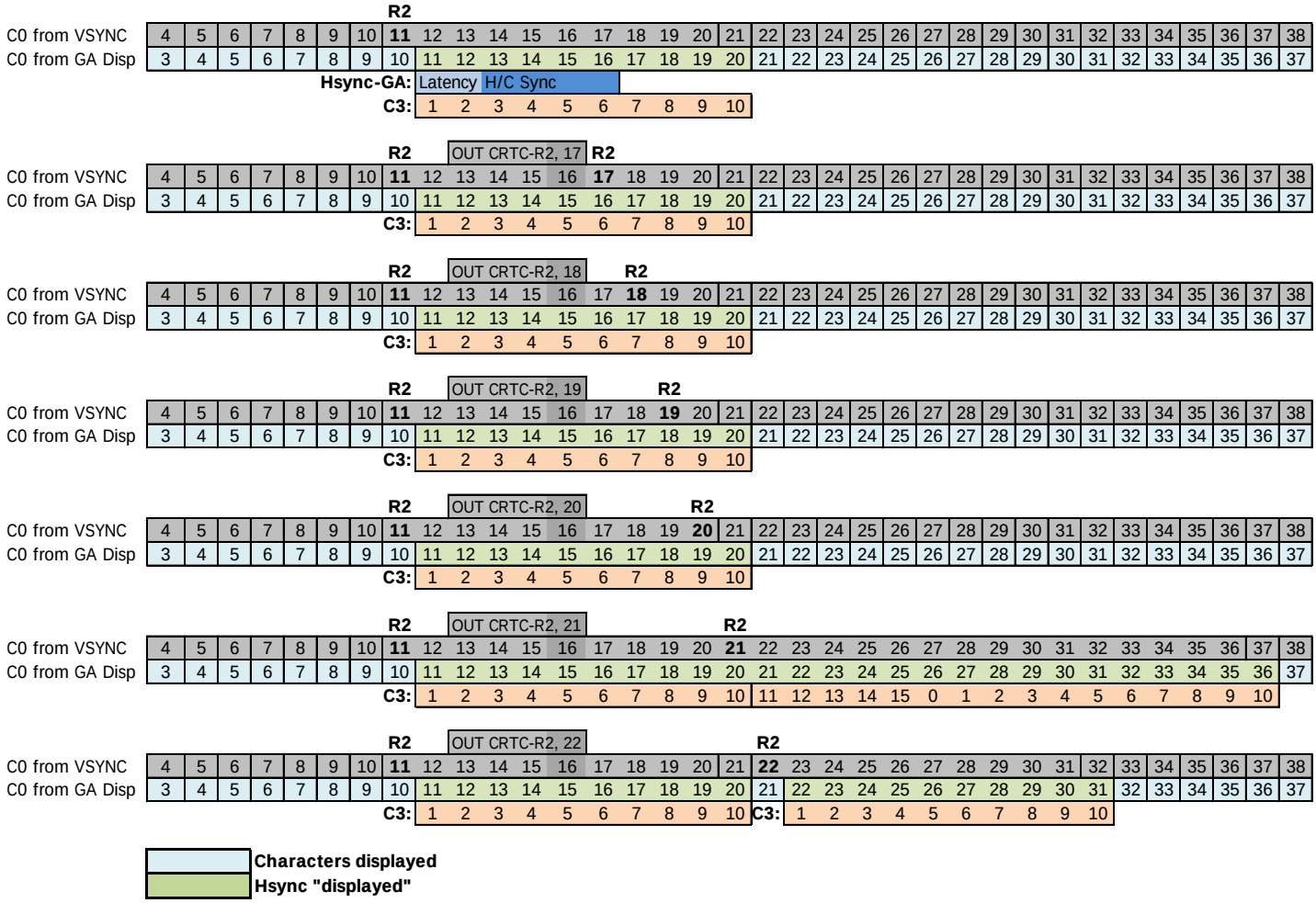
CRTC-R3=10 (HBL Size=10 chars)



CRTC 3, 4

CRTC-R2=11

CRTC-R3=10 (HBL Size=10 chars)



16.4 PRISE EN COMPTE VSYNC DURANT LA HSYNC

16.4.1 GÉNÉRALITÉS

L'évaluation de la condition VSYNC est réalisée quelle que soit la valeur de C0 pour les CRTC 0, 1 et 2, mais **uniquement lorsque C0=0 pour les CRTC 3 et 4**.

Dans tous les cas, C4 doit être égal à R7, soit parce que C4 y arrive, soit parce que R7 a été programmé avec la valeur de C4. C'est vrai également lorsque C4 atteint R7 à partir d'une gestion de ligne additionnelle.

Pour tous les CRTC sauf le 2, une condition de VSYNC se produisant durant la HSYNC ne pose aucun problème particulier. Voir le paragraphe ci-après pour les détails concernant le CRTC 2.

La valeur de R2 a été appliquée sur un frame complet dans les schémas suivants et décrit des HSYNC de différentes longueurs qui empiètent sur la zone de test de VSYNC.

La partie de la ligne sur les schémas est celle qui correspond à C4=R7-1 (ou précédent si R7 vaut 0), C9=7 (lorsque R9=7).

16.4.2 CRTC 0, 1

		R2														Vsync			
C0 from VSYNC	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	0	1	2	3
C0 from GA	48	**49**	50	51	52	53	54	55	56	57	58	59	60	61	62	63	0	1	2
R3=12	**C3:**	1	2	3	4	5	6	7	8	9	10	11	12						

		R2														Vsync			
C0 from VSYNC	49	**50**	51	52	53	54	55	56	57	58	59	60	61	62	63	0	1	2	3
C0 from GA	48	**49**	50	51	52	53	54	55	56	57	58	59	60	61	62	63	0	1	2
R3=13	**C3:**	1	2	3	4	5	6	7	8	9	10	11	12	13					

		R2														Vsync			
C0 from VSYNC	49	**50**	51	52	53	54	55	56	57	58	59	60	61	62	63	0	1	2	3
C0 from GA	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	0	1	2
R3=14	**C3:**	1	2	3	4	5	6	7	8	9	10	11	12	13	14				

		R2														Vsync			
C0 from VSYNC	49	**50**	51	52	53	54	55	56	57	58	59	60	61	62	63	0	1	2	3
C0 from GA	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	0	1	2
R3=15	**C3:**	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15			

16.4.3 CRTC 3, 4

		R2													Vsync			
C0 from VSYNC	49	**50**	51	52	53	54	55	56	57	58	59	60	61	62	63	0	1	2	3
C0 from GA	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	0	1	2
R3=12		**C3:**	1	2	3	4	5	6	7	8	9	10	11	12					

		R2													Vsync			
C0 from VSYNC	49	**50**	51	52	53	54	55	56	57	58	59	60	61	62	63	0	1	2	3
C0 from GA	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	0	1	2
R3=13		**C3:**	1	2	3	4	5	6	7	8	9	10	11	12	13				

		R2													Vsync			
C0 from VSYNC	49	**50**	51	52	53	54	55	56	57	58	59	60	61	62	63	0	1	2	3
C0 from GA	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	0	1	2
R3=14		**C3:**	1	2	3	4	5	6	7	8	9	10	11	12	13	14			

		R2													Vsync			
C0 from VSYNC	49	**50**	51	52	53	54	55	56	57	58	59	60	61	62	63	0	1	2	3
C0 from GA	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	0	1	2
R3=15		**C3:**	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15		

16.4.4 CRTC 2

La condition **VSYNC** est évaluée pour toutes les valeurs de C0 pendant lesquelles C4=R7.

Si cette évaluation a lieu lorsque la **HSYNC** est en cours alors une **VSYNC FANTÔME** débute.

Comme on a pu le voir dans le chapitre précédent, 2 **HSYNC** ne peuvent pas être collées sans que la première déborde, ce qui implique que le traitement de la HSYNC n'est pas totalement achevé sur la position C0=R2+R3-1 car le compteur n'a pas encore été réinitialisé.

Une condition de VSYNC ayant lieu sur une position comprise entre **C0=R2 et la position de C0 après R3+1 caractères** va donc déclencher une **VSYNC FANTÔME, sauf si R2=0.**

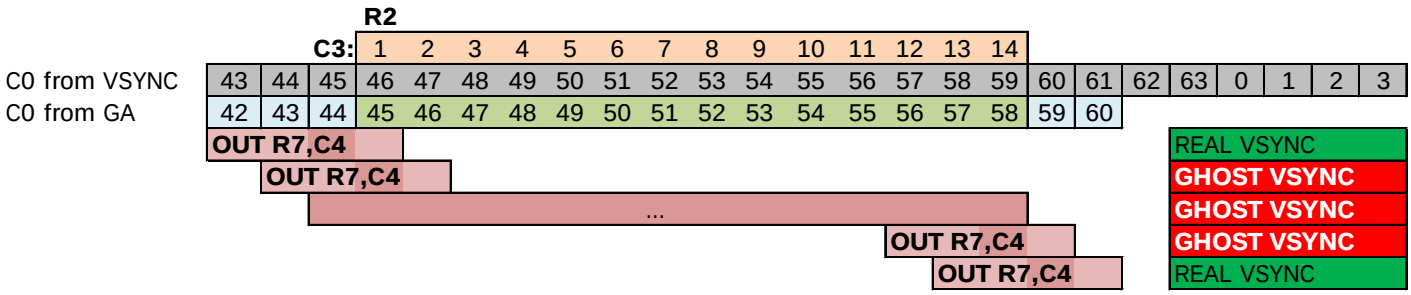
Lorsque **R2=0**, la HSYNC débute sur C0=0, mais la VSYNC a eu le temps d'être traitée et elle se produit normalement.

Si la condition VSYNC était seulement « ignorée » durant la HSYNC, elle se produirait immédiatement en sortie de HSYNC, puisque la condition C4=R7 est encore vraie et l'évaluation a lieu sur toutes les valeurs de C9 et C0.

Il se produit très certainement un joli conflit sur l'activation de la broche VSYNC (broche 40) lorsque la broche HSYNC (broche 39) est encore à l'état haut.

La VSYNC ne peut plus survenir car le CRTC a activé son compteur de HSYNC et ne pourra donc accepter une nouvelle condition de VSYNC que lorsque la VSYNC FANTÔME sera terminée.

Le schéma suivant décrit précisément l'inhibition de la VSYNC lorsque R7 est modifié avec le C4 courant :



Les schémas suivants décrivent les différentes valeurs de C0 sur lesquelles la condition VSYNC est évaluée en fonction de R2 et R3 :

	R2														Vsync			
C0 from VSYNC	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	0	1	2	3
C0 from GA	48	**49**	50	51	52	53	54	55	56	57	58	59	60	61	62	63	0	1	2
R3=12	**C3:**	1	2	3	4	5	6	7	8	9	10	11	12						

	R2															Vsync			
C0 from VSYNC	49	**50**	51	52	53	54	55	56	57	58	59	60	61	62	63	0	1	2	3
C0 from GA	48	**49**	50	51	52	53	54	55	56	57	58	59	60	61	62	63	0	1	2
R3=13	**C3:**	1	2	3	4	5	6	7	8	9	10	11	12	13					

	R2															No Vsync			
C0 from VSYNC	49	**50**	51	52	53	54	55	56	57	58	59	60	61	62	63	0	1	2	3
C0 from GA	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	0	1	2
R3=14	**C3:**	1	2	3	4	5	6	7	8	9	10	11	12	13	14				

	R2															No Vsync			
C0 from VSYNC	49	**50**	51	52	53	54	55	56	57	58	59	60	61	62	63	0	1	2	3
C0 from GA	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	0	1	2
R3=15	**C3:**	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15			

Il est possible de contourner la limitation du CRTC 2 à traiter sa VSYNC de plusieurs manières si on souhaite positionner R2 et R3 librement.

Une solution consiste à gérer soi-même R7, afin que la VSYNC ne puisse pas être traitée durant la HSYNC, et positionner R7 avec C4 une fois la HSYNC dépassée.

La FAKE VSYNC évoquée dans le chapitre 7.3 ne fonctionne pas correctement sur tous les CPC que j'ai pu tester. C'est donc une solution à éviter tant qu'on ne sait pas à quoi cette différence est liée et si on peut y pallier. De mes observations, ce n'est pas la mise à 1 du bit 0 du port B du PPI qui génère une VSYNC, mais plutôt le moment précis ou le CRTC annule la VSYNC FANTÔME.

Une autre solution consiste à modifier la taille de la HSYNC au bon moment.
La valeur de R3 est ainsi réduite au moment ou la condition VSYNC doit être évaluée.

En procédant ainsi, la VSYNC a lieu normalement.

		R2				OUT R3, 12										Vsync			
C0 from VSYNC	49	**50**	51	52	53	54	55	56	57	58	59	60	61	62	63	0	1	2	3
C0 from GA	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	0	1	2
R3 init=14	**C3:**	1	2	3	4	5	6	7	8	9	10	11	12	*13*	*14*				

		R2				OUT R3, 12										Vsync			
C0 from VSYNC	49	**50**	51	52	53	54	55	56	57	58	59	60	61	62	63	0	1	2	3
C0 from GA	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	0	1	2
R3 init=15	**C3:**	1	2	3	4	5	6	7	8	9	10	11	12	*13*	*14*	*15*			

Remarque : Si R2 et R3 sont programmés de manière à ce que la HSYNC s'achève sur la ligne suivante, cela signifie qu'il y a 2 HSYNC par ligne. Mais on a également une HSYNC active sur C0=0 et la condition « **Dernière Ligne** » n'est alors pas évaluée. Étant donné qu'elle est fausse en début de ligne, cela provoque le débordement de C4 en fin de frame (voir chapitre 10.3.3).

16.5 DISPEN ET HSYNC

16.5.1 CRTC 0, 1, 3, 4

La gestion d'affichage du fond ou du BORDER, avec les conditions C0=0 et C0=R1, est prise en compte pour tous les CRTC sauf le 2.

16.5.2 CRTC 2

La condition permettant de rétablir l'affichage du fond a lieu lorsque C0=0 (et que C4 n'a jamais atteint R6).
Le BORDER est activé lorsque C0=R1.
Cependant, durant la HSYNC, ce test C0=0 n'est pas effectué.
(à voir si c'est la même chose avec C0=R1 pour activer le BORDER)

Dans cette condition, **le BORDER n'est pas désactivé**.
Ce qui est BORDERLINE, on peut le dire !

16.6 CRTC 2 ET HSYNC

Le CRTC 2 est le champion dans la différence de gestion de certaines conditions survenant sur des valeurs précises de C0 pendant une HSYNC:

- Lorsque la programmation de R2 et R3 amène C0 à atteindre la position qui précède C0=0. Si une HSYNC a lieu sur cette position, alors la VSYNC sur C4=R7 est considérée comme activée mais sans que le signal VSYNC soit transmis au GATE ARRAY (VSYNC FANTÔME). Si R7 est programmé avec la valeur de C4 durant la HSYNC, cela déclenche une VSYNC FANTÔME.
- Si R2 est programmé avec 0, la condition VSYNC est détectée assez tôt pour se produire normalement.
- Une HSYNC placée sur la position C0=0 ne permet pas de désactiver le BORDER avant un prochain C0=0.
- Une condition de dernière ligne effective sur la position C0=0 est annulée si une HSYNC est placée sur la position C0=0. Aussi C4 s'incrémente au lieu de passer à 0 en fin de frame.
- Une mise à jour de R4 ou R9 est ignorée si elle a lieu durant une HSYNC afin de positionner un état « dernière ligne » à vrai.

Quelques exemples :

- Si R0=63, R2=50 et R3=14, alors le frame défile sans bande noire. L'affichage est activé mais il n'y a plus de VSYNC.
- Si R0=63, R2=50 et R3=15, alors le frame n'est plus affiché, il défile sans bande noire. L'affichage est désactivé (Border), il n'y a plus de VSYNC et C4 déborde continuellement.
- Si R0=63, R2=0 et R3=6, alors le frame n'est plus affiché mais une bande noire défile. L'affichage est désactivé (Border), la VSYNC est présente mais C4 déborde et donc le frame défile.

16.7 LE BON MOMENT…

Si R2 est mis à jour sans « précautions », une nouvelle HSYNC peut survenir durant ou après la fin de la ligne courante. Dans cette situation, le moniteur doit gérer plusieurs HSYNC ou l'absence de HSYNC après la mise à jour, et cela se traduit par une distorsion horizontale de l'image.

Le moniteur essaie de caler son image sur la nouvelle position de C0=R2.
Étant donné qu'il met plusieurs lignes pour y parvenir, cela se traduit, visuellement, par un décalage progressif des lignes. C'est à peu près le même principe que lorsque la longueur de la HSYNC est modifiée avec une valeur inférieure à 6 pour décaler l'image.

De nombreuses démos ont utilisé ces principes (R2 et/ou R3) pour effectuer des déformations horizontales de l'image ou réaliser des scrollings.

Tout comme pour R7 au niveau vertical, il est possible d'éviter ce décrochage de synchronisation horizontale du moniteur en agissant pour que C0 repasse à la nouvelle valeur de R2 au même endroit que l'ancienne valeur de C0=R2. Cela nécessite de modifier R0 pour qu'il repasse à 0 plus tôt si le nouveau C0=R2 à atteindre est plus élevé que l'ancien, ou inversement d'agrandir R0 afin que C0 repasse à 0 plus tard si le nouveau C0=R2 à atteindre est moins élevé que l'ancien. Il faut juste imaginer que ce sont les compteurs qui viennent se placer où ils devraient se trouver.

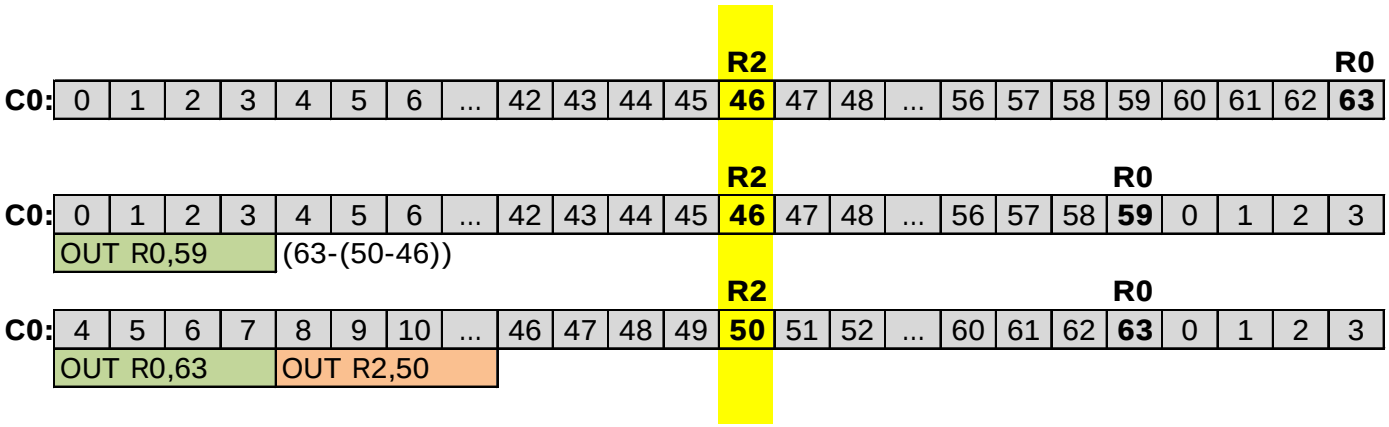
Cette petite gymnastique de repositionnement des compteurs évite au moniteur de perdre la HSYNC (ou d'en avoir deux en moins de 64 µsec), ce qui peut être ennuyeux pour un programme ne pouvant souffrir ce type d'artefact visuel.

À défaut, si le CPC n'a pas vocation à servir le diktat de diffusion vidéo des organisateurs de gros meetings à concours (auquel est soumis le diktat évoqué dans le chapitre 15.4) ou s'adapter à de jolis écrans plats LCD, il est toujours possible de dissimuler ce décrochage en modifiant R2 dans une zone non affichée (par exemple durant la VSYNC).

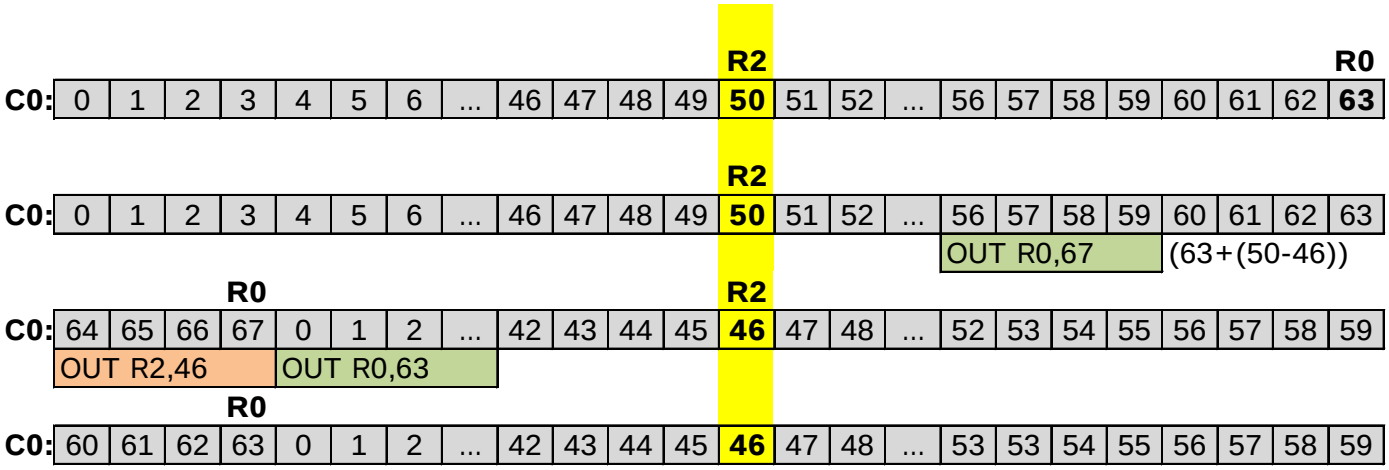
Il est aussi possible, comme pour la synchro verticale, d'agir sur R1 et/ou R6, ou même mettre toutes les encres noires un « certain temps ».

Le réglage du potentiomètre H-Hold nécessite toutefois un petit tournevis plat pour être réglé sur un moniteur CTM.
Voici néanmoins 2 schémas pour traduire la gymnastique des compteurs nécessaire pour passer de R2 de 46 à 50 et inversement, de 50 à 46 (le choix de ces valeurs n'étant ni fortuit, ni indépendant de ma volonté) sans traumatiser l'écran.

16.7.1 PASSER DE R2=46 À R2=50 SUR DES LIGNES DE 64 µSEC



16.7.2 PASSER de R2=50 À R2=46 SUR DES LIGNES DE 64 µSEC



17 AFFICHAGE : REGISTRE R1

17.1 GÉNÉRALITÉS

Ce registre a pour fonction de définir le nombre de caractères horizontaux affichés sur une ligne.
Sa valeur est exprimée en nombre de caractères CRTC (pour rappel, chaque caractère CRTC occupe 2 octets en RAM).

Il joue également un rôle important pour la mémorisation du pointeur vidéo courant.

De manière générale, le CRTC génère R0+1 caractères sur une ligne, dont R1 caractères seront affichés.
Lorsque R1 caractères ont été affichés (DISPLAY ENABLE ON) alors l'affichage est inhibé, et du "BORDER" est affiché sur CPC (DISPLAY ENABLE OFF) via le GATE ARRAY.

À noter que la broche DISPLAY ENABLE porte divers nom, et on pourra la retrouver sous d'autres appellations exotiques, comme DISPTMG ou DE selon les différentes documentations CRTC.

Pour afficher l'intégralité des caractères que le CRTC peut générer pour une ligne, R1 devrait en principe valoir R0+1. En effet, le "flag" DISPLAY ENABLE est positionné à OFF pour générer du BORDER lorsque C0=R1.

Or, C0 ne peut jamais dépasser R0.
Pour afficher tous les caractères programmés avec R0, il est possible d'empêcher que C0 atteigne R1, soit en positionnant R1>R0, soit en changeant la valeur de R1 en cours de ligne.

Dans cette situation, un problème se pose car l'égalité entre C0 et R1 sert à la mise à jour du pointeur vidéo lorsque C9=R9 (dernière ligne d'un "caractère").

Ce défaut de mise à jour provoque dès lors une répétition des lignes de caractères, dans des conditions propres à chacun des CRTC.

Remarque :
Ce défaut lié à la gestion dynamique des compteurs (dite de bornes et de poteaux) est une plaie car il ne permet pas au compteur vidéo d'atteindre 128 octets pour 1 ligne (64 µsec), ce qui peut conduire certains malheureux à utiliser des frames de 65 µsec par ligne en positionnant R0 à 64 avec une position de synchro horizontale unique pour ces lignes de 65µsec...
Cela permet d'ajouter un sifflement aux musiques jouées dans les démos et vérifier que le moniteur est mal réglé.
Sans artifice cela ne permet pas de disposer d'un pointeur vidéo dont le poids fort ne varie pas durant l'affichage d'une ligne. Ce « truc » permet de gagner de la CPU lorsqu'il est question d'afficher des données, car on peut se contenter d'incrémenter un registre 8 bits au lieu de 16 bits pour le pointeur vidéo. C'est pourquoi, malheureusement, tant de frames sont formatés avec des lignes de 64 octets de large en « minimized screen », soit 16 octets de moins que la largeur standard.

Lorsque C0 repasse à 0 (suite à une condition C0=R0) alors l'affichage est autorisé (DISPLAY ENABLE ON). À noter que si C0 repasse à 0 car il a atteint 255 en ayant débordé, cela n'autorise pas l'affichage (au moins sur le CRTC 0. À voir sur les autres CRTC).
À noter également que **cette condition n'est pas gérée par le CRTC 2 durant la HSYNC.**

Si R1 est positionné à 0, alors plus aucun caractère n'est affiché, quel que soit le CRTC d'un CPC.

Le BORDER débute sur la position C0=R1 (en considérant, pour le CRTC 0, que la fonction SKEW DISP n'est pas utilisée).

Remarque :
À priori ce ne serait pas le cas sur le BBC avec le Hitachi HD6845SP (type 0) soit avec Samsung KS68C45S ou un VLSI VL68C45S23PC. (mais pourquoi je parle de ça dans un document dédié au CPC, moi ?). (pourquoi pas le NEC PD7220 tant que j'y suis ?)

Dans les schémas ci-après, et dans quelques propos liminaires, je fais référence à deux pointeurs mémoires dans le CRTC, que j'ai nommé **VMA** et **VMA'**.

Lorsque le CRTC affiche des caractères, il se sert toujours du pointeur **VMA**.

Ce pointeur est incrémenté à chaque fois qu'un caractère est traité par le CRTC, qu'il soit affichable ou non. Dans certaines « préversions » de CRTC, ce pointeur n'était pas géré durant la VSYNC, mais cela ne concerne pas les CRTC des CPC à ma connaissance.

Lorsque C0=R1 et C9=R9, alors le pointeur courant **VMA** est transféré dans le pointeur **VMA'**.

Lorsque C0=0, à chaque début de ligne, le pointeur de ligne **VMA'** est transféré dans le pointeur courant **VMA**, sauf pour le premier caractère (C4=0).

17.2 AFFICHAGES SELON R1

17.2.1 AFFICHAGE AVEC R1 <= R0

Le schéma suivant décrit la gestion R1 dans un cadre de programmation "standard" du CRTC sur un CPC qui vient d'être allumé avec une rom Basic standard.

Données initiales : CRTC-R0=63 / **CRTC-R1=40 / CRTC-R9=7 /** CRTC-R12=0 / CRTC-R13=0

C0=0	**C0=R1 & C9=R9**	**VRAM-C9-Bit 0..2**		**C0:**	0	1	2	3	...	37	38	39	**40**	41	42	43	44	45	...	63
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9 :	0		0	1	2	3	...	37	38	39	DISP-OFF							
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9 :	1		0	1	2	3	...	37	38	39	DISP-OFF							
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9 :	2		0	1	2	3	...	37	38	39	DISP-OFF							
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9 :	3		0	1	2	3	...	37	38	39	DISP-OFF				**BORDER**			
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9 :	4		0	1	2	3	...	37	38	39	DISP-OFF							
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9 :	5		0	1	2	3	...	37	38	39	DISP-OFF							
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9 :	6		0	1	2	3	...	37	38	39	DISP-OFF							
CRTC-VMA=CRTC-VMA'	**CRTC-VMA'=CRTC-VMA**	CRTC-VMA C9 :	7		0	1	2	3	...	37	38	39	DISP-OFF							
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9 :	0		40	41	42	43	...	77	78	79	DISP-OFF							
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9 :	1		40	41	42	43	...	77	78	79	DISP-OFF							
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9 :	2		40	41	42	43	...	77	78	79	DISP-OFF							
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9 :	3		40	41	42	43	...	77	78	79	DISP-OFF							
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9 :	4		40	41	42	43	...	77	78	79	DISP-OFF							
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9 :	5		40	41	42	43	...	77	78	79	DISP-OFF							
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9 :	6		40	41	42	43	...	77	78	79	DISP-OFF							
CRTC-VMA=CRTC-VMA'	**CRTC-VMA'=CRTC-VMA**	CRTC-VMA C9 :	7		40	41	42	43	...	77	78	79	DISP-OFF							

17.2.2 AFFFICHAGE AVEC R1 > R0

Le schéma suivant décrit l'affichage lorsque R1 est programmé avec une valeur supérieure à 63.

Données initiales : CRTC-R0=63 / CRTC-R1=**64 /** CRTC-R9=7 / CRTC-R12=0 / CRTC-R13=0

C0=0	**C0=R1 & C9=R9**	**VRAM-C9-Bit 0..2**		**C0:**	0	1	2	3	...	53	54	55	56	57	58	59	60	61	62	63
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9:	0		0	1	2	3	...	53	54	55	56	57	58	59	60	61	62	63
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9:	1		0	1	2	3	...	53	54	55	56	57	58	59	60	61	62	63
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9:	2		0	1	2	3	...	53	54	55	56	57	58	59	60	61	62	63
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9:	3		0	1	2	3	...	53	54	55	56	57	58	59	60	61	62	63
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9:	4		0	1	2	3	...	53	54	55	56	57	58	59	60	61	62	63
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9:	5		0	1	2	3	...	53	54	55	56	57	58	59	60	61	62	63
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9:	6		0	1	2	3	...	53	54	55	56	57	58	59	60	61	62	63
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9:	7		0	1	2	3	...	53	54	55	56	57	58	59	60	61	62	63
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9:	0		0	1	2	3	...	53	54	55	56	57	58	59	60	61	62	63
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9:	1		0	1	2	3	...	53	54	55	56	57	58	59	60	61	62	63
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9:	2		0	1	2	3	...	53	54	55	56	57	58	59	60	61	62	63
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9:	3		0	1	2	3	...	53	54	55	56	57	58	59	60	61	62	63
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9:	4		0	1	2	3	...	53	54	55	56	57	58	59	60	61	62	63
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9:	5		0	1	2	3	...	53	54	55	56	57	58	59	60	61	62	63
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9:	6		0	1	2	3	...	53	54	55	56	57	58	59	60	61	62	63
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9:	7		0	1	2	3	...	53	54	55	56	57	58	59	60	61	62	63

R0 **R1**

Si R1 > R0, alors C0 n'est jamais égal à R1 (lorsque C9=R9), et **VMA'** n'est donc pas mis à jour avec **VMA.**

Cela provoque une répétition de caractères car l'adresse courante n'est pas mise à jour lors du changement de ligne-caractère (lorsque C9=R9).

Remarque 1 :
Seuls les 10 premiers bits ne sont pas mis à jour dans ce contexte.
Les bits qui déterminent le numéro de bloc (ligne caractère) (C9) continuent de "participer" à l'adresse. Voir chapitre 20, page 217.

Remarque 2 :
Dans la mesure où les conditions permettent une prise en compte de l'adresse, modifier R12/R13 permet d'éviter cette répétition. En pratique cela permet donc de coller les "lignes" lorsque C0 repasse à 0 plusieurs fois durant une "ligne". Cependant les CRTC 0 et 2 génèrent un octet de BORDER, sachant que le CRTC 0 peut tout de même empêcher la génération de cet octet en consommant de la CPU. Voir chapitre 19.2, page 170.

Remarque 3 :
Selon les CRTC, la mise à jour initiale de CRTC-VMA'/CRTC-VMA via R12/R13 n'est pas la même.
Voir chapitres suivants et chapitre 19.2, page 170.

17.3 MISE JOUR DYNAMIQUE DE R1

La condition C0=R1 est prise en compte immédiatement sur une ligne.
Elle peut survenir plusieurs fois sur une même ligne si R1 est reprogrammé.

Lorsque la première condition C0=R1 est vraie, il n'y a plus d'affichage des caractères et du BORDER est affiché.

Cependant, même si du BORDER est affiché, le pointeur **VMA continue de compter** (Le CRTC n'étant qu'un vaste champ de compteurs en fleurs...).

Si R1 est modifié une nouvelle fois durant la ligne pour satisfaire de nouveau la condition C0=R1 lorsque C9=R9 alors cela **entrainera une mise à jour du pointeur vidéo**.

Dis autrement, la modification de R1 durant l'affichage de BORDER R1 permet de mettre à jour le pointeur vidéo **sans que les données soient affichées**.

Les schémas ci-après montrent ces comportements lorsque C4>0.

C0=0	C0=R1 & C9=R9	VRAM-C9-Bit 0..2		Upd C0:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9 :	0	R1=26	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	DISP-OFF				
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9 :	1	R1=25	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	DISP-OFF					
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9 :	2	R1=24	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	DISP-OFF						
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9 :	3	R1=23	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	DISP-OFF							
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9 :	4	R1=22	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	DISP-OFF								
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9 :	5	R1=21	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	DISP-OFF									
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9 :	6	R1=20	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	DISP-OFF										
CRTC-VMA=CRTC-VMA'	**CRTC-VMA'=CRTC-VMA**	CRTC-VMA C9 :	7	R1=19	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	DISP-OFF											
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9 :	0	R1=18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	DISP-OFF												
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9 :	1	R1=17	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	DISP-OFF													
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9 :	2	R1=16	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	DISP-OFF														
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9 :	3	R1=15	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	DISP-OFF															
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9 :	4	R1=14	19	20	21	22	23	24	25	26	27	28	29	30	31	32	DISP-OFF																
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9 :	5	R1=13	19	20	21	22	23	24	25	26	27	28	29	30	31	DISP-OFF																	
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9 :	6	R1=12	19	20	21	22	23	24	25	26	27	28	29	30	DISP-OFF																		
CRTC-VMA=CRTC-VMA'	**CRTC-VMA'=CRTC-VMA**	CRTC-VMA C9 :	7	R1=11	19	20	21	22	23	24	25	26	27	28	29	DISP-OFF																			
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9 :	0	R1=12	30	31	32	33	34	35	36	37	38	39	40	41	DISP-OFF																		

R0

C0=0	C0=R1 & C9=R9	VRAM-C9-Bit 0..2		Upd C0:	0	1	2	3	...	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	**63**
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9 :	0	R1=64	0	1	2	3	...	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9 :	1		0	1	2	3	...	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9 :	2		0	1	2	3	...	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9 :	3		0	1	2	3	...	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9 :	4		0	1	2	3	...	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9 :	5		0	1	2	3	...	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9 :	6		0	1	2	3	...	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
CRTC-VMA=CRTC-VMA'	**CRTC-VMA'=CRTC-VMA**	CRTC-VMA C9 :	7	R1=40	0	1	2	3	...	37	38	39	DISP-OFF																									
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9 :	0		40	41	42	43	...	77	78	79	DISP-OFF																									
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9 :	1		40	41	42	43	...	77	78	79	DISP-OFF																									
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9 :	2		40	41	42	43	...	77	78	79	DISP-OFF																									
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9 :	3		40	41	42	43	...	77	78	79	DISP-OFF																									
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9 :	4		40	41	42	43	...	77	78	79	DISP-OFF																									
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9 :	5		40	41	42	43	...	77	78	79	DISP-OFF																									
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9 :	6		40	41	42	43	...	77	78	79	DISP-OFF																									
CRTC-VMA=CRTC-VMA'	**CRTC-VMA'=CRTC-VMA**	CRTC-VMA C9 :	7		40	41	42	43	...	77	78	79	DISP-OFF																									

													R1																							R0
C0=0	**C0=R1 & C9=R9**	**VRAM-C9-Bit 0..2**		**Upd C0:**	0	1	2	3	...	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	**63**
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9 :	0	R1=40	0	1	2	3	...	37	38	39	DISP-OFF																							
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9 :	1		0	1	2	3	...	37	38	39	DISP-OFF																							
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9 :	2		0	1	2	3	...	37	38	39	DISP-OFF																							
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9 :	3		0	1	2	3	...	37	38	39	DISP-OFF																							
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9 :	4		0	1	2	3	...	37	38	39	DISP-OFF																							
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9 :	5		0	1	2	3	...	37	38	39	DISP-OFF																							
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9 :	6		0	1	2	3	...	37	38	39	DISP-OFF																							
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9 :	7	R1=64	0	1	2	3	...	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9 :	0		0	1	2	3	...	37	38	39	DISP-OFF																							
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9 :	1		0	1	2	3	...	37	38	39	DISP-OFF																							
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9 :	2		0	1	2	3	...	37	38	39	DISP-OFF																							
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9 :	3		0	1	2	3	...	37	38	39	DISP-OFF																							
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9 :	4		0	1	2	3	...	37	38	39	DISP-OFF																							
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9 :	5		0	1	2	3	...	37	38	39	DISP-OFF																							
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9 :	6		0	1	2	3	...	37	38	39	DISP-OFF																							
CRTC-VMA=CRTC-VMA'	**CRTC-VMA'=CRTC-VMA**	CRTC-VMA C9 :	7		0	1	2	3	...	37	38	39	DISP-OFF																							

																		R1																		R0
C0=0	**C0=R1 & C9=R9**	**VRAM-C9-Bit 0..2**		**Upd C0:**	0	1	2	...	32	33	34	35	36	37	38	39	40	41	42	43	44	45	...	51	52	53	54	55	56	57	58	59	60	61	62	**63**
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9 :	0	R1=40	0	1	2	...	32	33	34	35	36	37	38	39	DISP-OFF																			
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9 :	1		0	1	2	...	32	33	34	35	36	37	38	39	DISP-OFF																			
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9 :	2		0	1	2	...	32	33	34	35	36	37	38	39	DISP-OFF																			
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9 :	3		0	1	2	...	32	33	34	35	36	37	38	39	DISP-OFF																			
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9 :	4		0	1	2	...	32	33	34	35	36	37	38	39	DISP-OFF																			
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9 :	5		0	1	2	...	32	33	34	35	36	37	38	39	DISP-OFF																			
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9 :	6		0	1	2	...	32	33	34	35	36	37	38	39	DISP-OFF																			
CRTC-VMA=CRTC-VMA'	**CRTC-VMA'=CRTC-VMA**	CRTC-VMA C9 :	7		0	1	2	...	32	33	34	35	36	37	38	39	DISP-OFF															**OUT R1,0**			
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9 :	0		DISP-OFF			**CRTC-VMA +++++**					**OUT R1,40**					**CRTC-VMA +++++**															**OUT R1,0**			
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9 :	1		DISP-OFF			**CRTC-VMA +++++**					**OUT R1,40**					**CRTC-VMA +++++**															**OUT R1,0**			
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9 :	2		DISP-OFF			**CRTC-VMA +++++**					**OUT R1,40**					**CRTC-VMA +++++**															**OUT R1,0**			
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9 :	3		DISP-OFF			**CRTC-VMA +++++**					**OUT R1,40**					**CRTC-VMA +++++**															**OUT R1,0**			
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9 :	4		DISP-OFF			**CRTC-VMA +++++**					**OUT R1,40**					**CRTC-VMA +++++**															**OUT R1,0**			
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9 :	5		DISP-OFF			**CRTC-VMA +++++**					**OUT R1,40**					**CRTC-VMA +++++**															**OUT R1,0**			
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9 :	6		DISP-OFF			**CRTC-VMA +++++**					**OUT R1,40**					**CRTC-VMA +++++**															**OUT R1,0**			
CRTC-VMA=CRTC-VMA'	**CRTC-VMA'=CRTC-VMA**	CRTC-VMA C9 :	7		DISP-OFF			**CRTC-VMA +++++**					**OUT R1,40**					**CRTC-VMA +++++**																		
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9 :	0		80	81	82	...	112	113	114	##	##	##	##	##	DISP-OFF																			
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9 :	1		80	81	82	...	112	113	114	##	##	##	##	##	DISP-OFF																			
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9 :	2		80	81	82	...	112	113	114	##	##	##	##	##	DISP-OFF																			
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9 :	3		80	81	82	...	112	113	114	##	##	##	##	##	DISP-OFF																			
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9 :	4		80	81	82	...	112	113	114	##	##	##	##	##	DISP-OFF																			
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9 :	5		80	81	82	...	112	113	114	##	##	##	##	##	DISP-OFF																			
CRTC-VMA=CRTC-VMA'		CRTC-VMA C9 :	6		80	81	82	...	112	113	114	##	##	##	##	##	DISP-OFF																			
CRTC-VMA=CRTC-VMA'	**CRTC-VMA'=CRTC-VMA**	CRTC-VMA C9 :	7		80	81	82	...	112	113	114	##	##	##	##	##	DISP-OFF																			

17.4 VMA'/VMA LORSQUE C4=0

Il existe des différences entre les CRTC pour l'affectation du pointeur vidéo avec les valeurs programmées dans R12 et/ou R13 lorsque C4=0 et C0=0.

17.4.1 CRTC 0, 3, 4

La première ligne caractère commence avec l'adresse définie par R12/R13, quelle que soit la valeur de R1.
Sur la première ligne du premier caractère d'un "frame" (C4=C9=C0=0), **le pointeur VMA' est mis à jour à l'aide du contenu de R12/R13.**
Cette mise à jour est suivie de la **mise à jour du pointeur VMA avec VMA'**.

Si R1>R0, le pointeur VMA' n'est plus mis à jour et les lignes se répètent.

Dans cette circonstance, lorsqu'un nouveau frame débute, **VMA' est mis à jour et toutes les lignes affichées deviennent identiques et égales au pointeur R12/R13**.

17.4.2 CRTC 1

La première ligne caractère commence avec l'adresse définie par R12/R13, quelle que soit la valeur de R1.
Lorsqu'on est sur le premier caractère d'un "frame" (C4=C0=0), **le pointeur VMA** (et non VMA' comme sur CRTC 0) **est mis à jour à l'aide du contenu de R12/R13**.

Remarque : cette particularité de mise à jour permet de modifier l'offset via R12 et/ou R13 sur chaque ligne (C9=0 à R9) d'un caractère pour lequel C4=0.

Cette mise à jour de VMA a cependant une conséquence lorsque R1>R0 durant tout le frame.

En effet, la condition C0=R1 ne se produit plus et le pointeur **VMA' n'est plus mis à jour**.
VMA' est "figé" sur le dernier pointeur connu lorsque C0 a atteint R1 lorsque C9=R9.
Lorsque C9=0, le pointeur VMA est rechargé avec VMA' lorsque C4>0.

On a donc, lorsque R1>R0, une première ligne caractère qui contient le pointeur défini dans R12/R13 et sur les suivantes, le dernier pointeur mis à jour dans VMA'.

Le pointeur vidéo continue cependant d'être incrémenté, même lorsque du BORDER est affiché.

C'est vrai au niveau horizontal (gestion R1) mais aussi en vertical (gestion R6).

Le pointeur VMA' est sur 14 bits. Lorsqu'il continue de s'incrémenter et dépasse la limite de définition des 10 bits, il peut modifier les Overscan Bits ™ (voir chapitre 20.5, page 220) et provoquer des changements de page.

Lorsque R1 devient supérieur à R0, il faut toutefois noter que si la mise à jour de R1 a lieu très exactement lorsque C0=R1 (R1.JIT), alors la condition C0=R1 n'est plus vraie et VMA' n'est pas mis à jour. Si la modification de R1 intervient lorsque C0=R1+1 (C0>R1), alors la condition C0=R1 a eu le temps d'être prise en compte et VMA' est mis à jour.

Exemple :
Si la largeur d'affichage d'une ligne est de 40 caractères (&28), le CRTC va pouvoir afficher 1024/40 lignes caractères différentes, soit 25 lignes complètes.

Lorsque C4=R6=25, le CRTC cesse d'afficher les lignes-caractères mais il continue cependant d'incrémenter son pointeur et de gérer les mises à jour de VMA' lorsque C0=R1 & C9=R9.

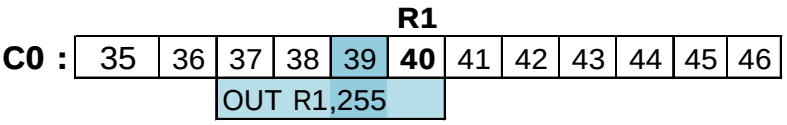
Si la modification de R1 (>R0) intervient durant cette période, le pointeur vidéo aura débordé.

Si R1 est modifié après que C4=24, C9=R9 et C0>R1, alors le pointeur sera égal à 40 x 25=1000 (la ligne répétée contiendra les caractères 1000 à 1023, suivi de 0000 à 0015).

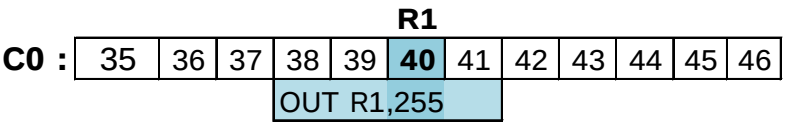
Si R1 est modifié après que C4=25, C9=R9 et C0>R1, alors le pointeur sera égal à 40 x 26=(1040 and 1023)=16 (la ligne répétée contiendra les caractères 0016 à 0055).

Pointeur VMA' répété = 960 :

C4=24 / C9=R9
CRTC-VMA'=40x24=960

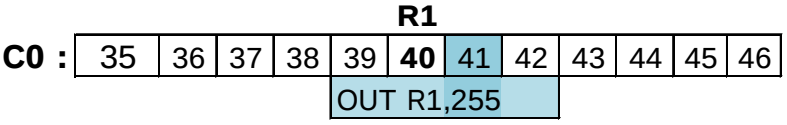


C4=24 / C9=R9
CRTC-VMA'=40x24=960



Pointeur VMA' répété = 1000 (40 x 25) :
La condition C0=R1 a eu lieu, le pointeur VMA' a été chargé avec VMA

C4=24 / C9=R9
CRTC-VMA'=40x24=960



17.4.3 CRTC 2

Lorsque R1>R0, **aucun des deux pointeurs n'est mis à jour avec R12/R13.**
Toutes les lignes sont identiques dans cette situation.

Sur la première ligne, VMA est affecté par VMA' (qui a lui-même été affecté par R12/R13 lorsque C0=R1 sur la dernière ligne du frame).

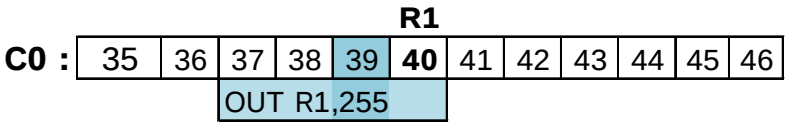
L'adresse répliquée suit la même logique que celle décrite pour le CRTC 1.

Donc, selon le moment ou R1 devient supérieur à R0, c'est le pointeur au moment ou le BORDER R1 est géré qui est pris en compte.

Il existe toutefois une différence avec le CRTC 1 lorsqu'on modifie R1 sur la position exacte de C0=R1. Le CRTC 1 est plus rapide que le CRTC 2 pour prendre en compte la mise à jour de R1 lorsque C0=R1. Sur le CRTC 2 une mise à jour de R1 sur la position C0=R1 arrive trop tard. La mise à jour de VMA' a déjà eu lieu en utilisant l'ancienne valeur de R1.

Pointeur VMA' répété = 960 :

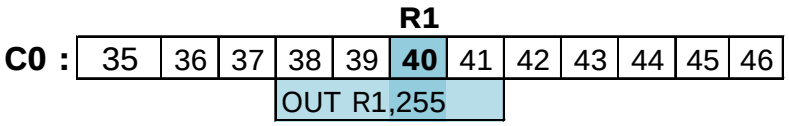
C4=24 / C9=R9
CRTC-VMA'=40x24=960



Pointeur VMA' répété = 1000 (40 x 25) :

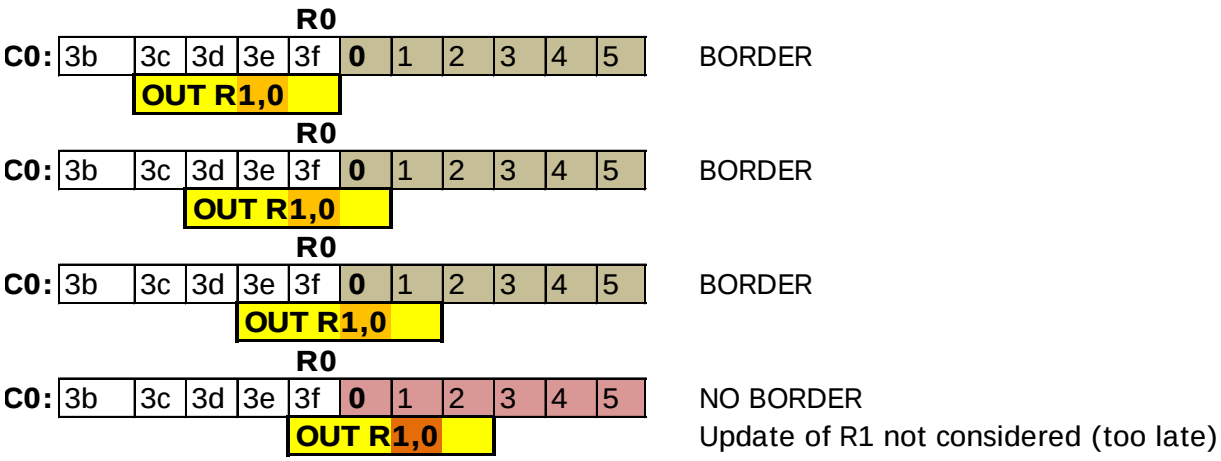
La condition C0=R1 a eu lieu, le pointeur CRTC-VMA' a été chargé avec CRTC-VMA.

C4=24 / C9=R9
CRTC-VMA'=40x24=960



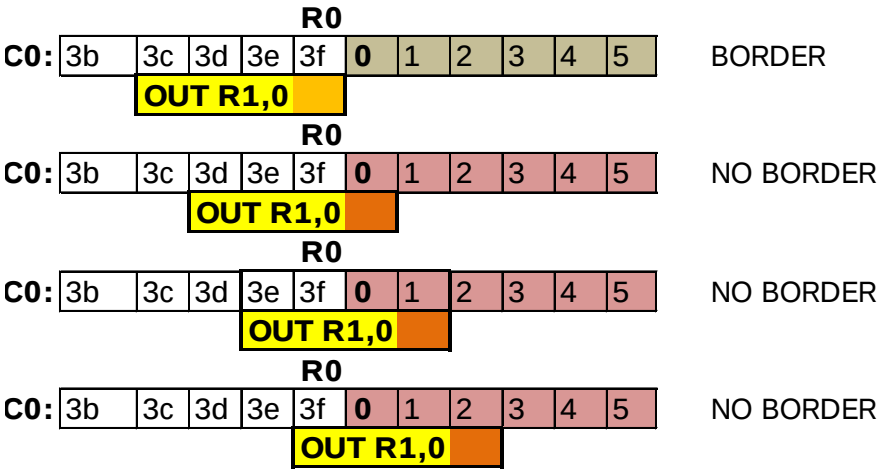
17.5 PRISE EN COMPTE R1=0

17.5.1 CRTC 0, 1, 2



Update of R1 not considered (too late)
Update of R1 ok (just in time)

17.5.2 CRTC 3, 4



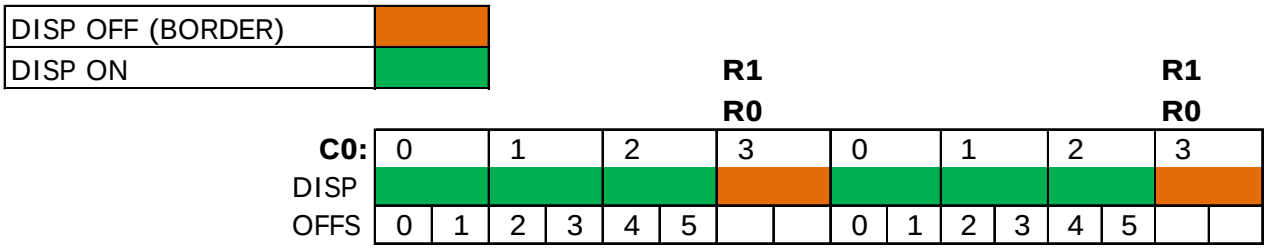
Update of R1 not considered (too late)
Update of R1 ok (just in time)

17.6 BORDER INTERLIGNE

17.6.1 R1=R0 ET C0=R0

CRTC 0, 1, 2, 3, 4

Si R1=R0, tous les CRTC vont afficher 1µs de BORDER sur le dernier caractère (C0=R0).
CRTC-R0=3
CRTC-R1=3
CRTC-R12/R13=0

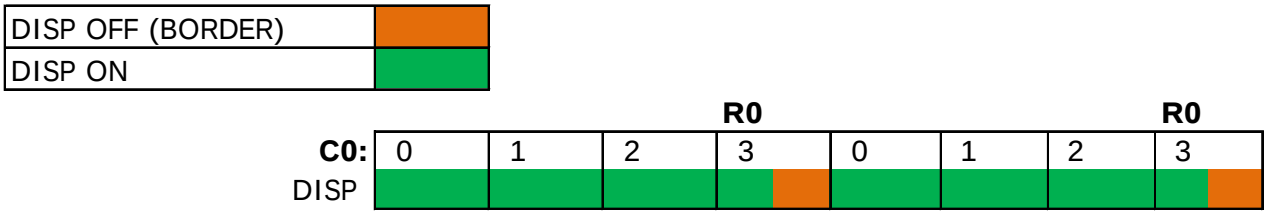


Remarque : Le pointeur en VRAM continuera sur l'offset 6 sur la prochaine "ligne caractère" (lorsque C9 repasse à 0)

17.6.2 R1>R0 ET C0=R0

CRTC 0, 2

La prise en compte du BORDER est anticipée.
1 octet de BORDER est généré (pendant exactement 0.5 µsec) avant que C0 passe à 0.
CRTC-R0=3
CRTC-R1=4
CRTC-R12/R13=0



Remarque 1 : Ce comportement reste vrai quelle que soit la valeur de R0.
Si R0=0, alors l'affichage alterne entre 1 octet DISP ON, et 1 octet DISP OFF.
Lorsque C0 atteint R0 sans que R1 ait été atteint, les CRTC 0 et 2 envoie un signal "BORDER ON" au GATE ARRAY, qui le prend en charge immédiatement. Ces CRTC sont en "avance" sur les caractères affichés par le GATE ARRAY et envoient le signal de BORDER 0.5 µsec trop tôt. Le signal « BORDER OFF » est envoyé sur le caractère qui suit C0=R0.

Remarque 2 : Le pointeur en VRAM restera "coincé" sur VMA'. Si les conditions le permettent (selon le CRTC et l'état des compteurs), VMA' peut être rechargé en mettant à jour R12 et/ou R13.

Remarque 3 : Le CRTC 0 dispose d'une fonction permettant de générer des conditions de prise en compte plus tardives du BORDER via son registre R8. En jouant sur la modification des conditions au moment de leur application, il est possible d'éviter la génération de cet octet de BORDER. Voir chapitre 19.2, page 170.

Remarque 4 : Le CRTC 2 ne dispose pas de la fonction SKEW DISP (R8) et l'octet de BORDER ne peut pas être "annulé" de cette façon. Mais LOGON doit-il vraiment révéler tous ses secrets ?

CRTC 1, 3, 4

CRTC-R0=3
CRTC-R1=4

Sur ces CRTC, aucun octet de BORDER n'est généré entre les "frames".
Des frames peuvent donc être créés dans la zone affichable sans qu'un octet apparaisse entre les frames. Ceci peut permettre de réaliser des plagiats hargneux...

DISP OFF (BORDER)	
DISP ON	

				R0				R0
C0:	0	1	2	3	0	1	2	3
DISP								

18 AFFICHAGE : REGISTRE R6

18.1 GÉNÉRALITÉS

La fonction de ce registre est de fixer le nombre de lignes-caractères affichées verticalement.
Lorsque ce nombre est atteint, du BORDER est affiché.

Le BORDER est affiché lorsque C4=R6 (1ère ligne-caractère R6).
Cette règle est vraie quelle que soit la valeur de C9.

Sauf sur CRTC 3 et 4, la prise en compte de R6 est immédiate sur le C0 courant.

La broche DISPLAY ENABLE du CRTC passe à "OFF" lorsque C4=R6, de la même manière que pour le registre R1 lorsque C0=R1.

En général lorsque la condition est remplie pour afficher du BORDER, il faut attendre un nouveau frame (C4=C9=C0=0) pour rétablir l'affichage des caractères.

Avec le registre R1, le BORDER est désactivé lorsque C0=0 et activé lorsque C0=R1.

Que le border soit généré via R1 ou R6, le pointeur en VRAM continue à être mis à jour.
En l'occurrence, si le BORDER a été activé par la condition sur R6, le pointeur VMA' continue à être mis à jour.

18.2 DÉLAIS ET PRIORITÉS DE BORDER R6

18.2.1 GÉNÉRALITÉS

L'état de la broche DISPLAY ENABLE dépend de 2 groupes de conditions internes du CRTC.

Tant que les "conditions R6" ne sont pas satisfaites, ce sont les "conditions R1" qui définissent l'état de la broche DISPLAY ENABLE.

Lorsque les "conditions R6" sont satisfaites, les "conditions R1" ne sont plus prises en compte.
La condition commune au rétablissement de l'affichage du fond est C4=C9=C0=0.

En général, la condition BORDER R6 est prioritaire sur la condition BORDER R1, mais il y cependant quelques différences selon les CRTC.

18.2.2 CRTC 0, 2

À l'exception de la première ligne d'un frame (C4=C9=0), positionner R6 avec C4 provoque l'activation immédiate et définitive du BORDER jusqu'au prochain frame.

La condition C4=R6 est prise en compte immédiatement (quelle que soit la valeur de C0), et BORDER R6 est prioritaire sur BORDER R1.

Ceci est également vrai durant le (ou les) caractères C4 générés pendant un ajustement vertical.

La valeur R6=0 utilisée sur la première ligne (C4=C9=0) est traitée spécifiquement (voir chapitre CONFLITS R6) et il est possible, sous conditions, d'utiliser ce conflit pour **annuler BORDER R6**.
Voir chapitre 18.3.

18.2.3 CRTC 1

Positionner R6 avec C4 provoque l'activation immédiate du BORDER, qui devient définitif jusqu'au prochain frame.

La condition C4=R6 est prise en compte immédiatement (quelle que soit la valeur de C0), et BORDER R6 est prioritaire sur BORDER R1.

Comme pour d'autres registres de ce CRTC (par exemple R3) la valeur 0 est prise en compte spécifiquement, et déclenche un BORDER sans que la condition C4=R6 soit requise.

Cependant, si la condition C4=R6 est également remplie lorsque C4=0, alors ce BORDER devient définitif jusqu'à ce que C4=C9=C0=0.

Il existe cependant une exception à cette règle : si la condition C4=R6=0 est vraie sur le dernier caractère du frame, alors le border n'est pas irréversible. Une mise à jour de R6<>C4 permet de l'annuler. En effet, si la condition C4=R6=0 a déjà été évaluée sur le frame précédent, **elle n'est plus réévaluée sur C4=R6=0 du nouveau frame**. Le passage sur le nouveau frame a cependant ré-autorisé le fond et le border est alors uniquement affiché à cause de R6=0 (et non plus à cause de l'équivalence C4=R6). À noter que la mise à jour de R6=0 sur le frame précédent est prise en compte jusqu'à C0=R0 qui précède le nouveau frame.

Dans les autres cas, lorsque R6 est mis à 0 alors que C4<>0, le BORDER est activé tant que R6=0 et il est désactivé dès que R6>0 et que sa nouvelle valeur est différente de C4 (auquel cas, le BORDER R6 est activé pour le frame).

18.2.4 CRTC 3, 4

Le test de R6 est fait au début de la ligne uniquement.

La mise à jour de R6 **en cours de ligne n'est donc pas prise en compte**.
Si R6 est mis à 0 lorsque C4=0 mais que C0>0, le BORDER n'est pas activé.

Le BORDER est activé uniquement au moment ou C0 passe à 0 lorsque C4=R6.

Ceci est également vrai durant un ajustement vertical.

Remarque : C4 n'est pas incrémenté sur ces 2 CRTC pendant l'ajustement vertical. Cela implique que si R6=R4, alors le BORDER concernera le dernier caractère ainsi que les lignes d'ajustement vertical définies avec R5.

La valeur de R6=0 n'est pas traitée spécifiquement et ne permet pas d'activer temporairement le BORDER comme sur le CRTC 1.

18.3 CONFLITS R6

18.3.1 GÉNÉRALITÉS

Afin de gérer correctement certains conflits de condition, un traitement particulier est souvent réalisé lorsqu'un registre vaut 0 (notamment sur CRTC 1, le spécialiste du domaine).

C'est notamment le cas lorsque R1=0, car cette valeur crée un conflit potentiel : désactivation du BORDER (C0=0 suite à condition C0=R0), activation du BORDER (C0=R1=0))

[Dans cette situation, c'est la désactivation du BORDER qui est activée]

Pour R6, la situation a été beaucoup moins bien gérée, selon les CRTC.

18.3.2 CRTC 0, 2

À l'exception de la première ligne d'un frame (C4=C9=0), positionner R6 avec C4 provoque l'activation immédiate et définitive du BORDER jusqu'au prochain frame.

Lorsque C4=R6=0 et C9=0 sur les CRTC 0 et 2, un conflit survient (ce conflit n'existe pas lorsque R6>0).

Lorsque R6 est égal à 0 dans cette situation, l'état de DISPLAY ENABLE passe à ON au début du caractère CRTC et repasse à OFF 0.5 µsec après.

Autrement dit, sur la première ligne du frame, il y a une alternance d'octets de BORDER et de caractères affichables (le pointeur vidéo continuant à compter normalement).

À chaque caractère CRTC :

- BORDER R6 passe à vrai car C4=R6 et C9=0
- BORDER R6 repasse à faux car on est sur un nouveau frame (C4=C9=0)

Remarque :
Cette alternance n'a lieu que lorsque la condition R1 est remplie (BORDER R1 est faux) et que les conditions du conflit existent (C4=R6=C9=0).

Lorsque le conflit est annulé (R6 mis à jour avec une valeur > 0 sur la première ligne du frame), le BORDER ne reste pas activé, contrairement aux autres lignes.
Dans cette situation cependant, si R6 vaut 0 lorsque C0=R1, le BORDER devient définitif.

Si on empêche la condition C0=R1 sur la ligne C4=C9=0 (par exemple R1=R0+1) et que R6 n'est plus égal à 0, alors le BORDER est désactivé sur la ligne suivante.

Autrement dit, **la condition C4=R6=0=BORDER est annulable sur la première ligne.**

Remarque :
La prise en compte sur chaque valeur de C0 de la valeur de R6 permet de cibler précisément des zones ou créer cette alternance BORDER/CARACTERES à l'octet.

Moyennant une jolie rupture ligne à ligne cela permet d'ajouter ponctuellement de jolies couleurs, via du BORDER, au mode graphique 2, par exemple, qu'on appellera le **mode JMLPAJIDA** pour les plus modestes, sinon directement Mode <votre pseudo>.

À noter qu'on peut parvenir à une alternance BORDER/CARACTERE sur CRTC 0 avec R0=0, mais sans que les compteurs C4 et C9 puissent avancer.

18.3.3 CRTC 1

Lorsque R6 est mis à jour avec 0, du BORDER est activé tant que la valeur du registre vaut 0.

La prise en compte sur chaque valeur de C0 de la valeur de R6 permet cibler précisément des zones ou créer ce BORDER.

Cependant, si C4=R6=0 (1ère ligne-caractère d'un nouveau frame) durant cette mise à jour, BORDER R6 devient vrai prioritairement pour tout le reste du frame, jusqu'au nouveau frame (C4=C9=C0=0).

18.3.4 CRTC 3, 4

Aucun conflit n'existe puisque la gestion de R6=0 n'existe pas durant la ligne et est testée une seule fois.

Positionner R6 à 0 lorsque C4 et C9 valent 0, mais que C0>0 n'aura aucune incidence avant le nouveau frame (ou le BORDER sera activé).

19 AFFICHAGE : REGISTRE R8

19.1 GÉNÉRALITÉS

Le registre R8 contient des paramètres pour la gestion du mode « Interlace » pour tous les CRTC.

Sur les CRTC 0, 3 et 4, une fonction complémentaire existe, qui permet de retarder la gestion d'activation/désactivation du BORDER, ou désactiver l'affichage comme le fait R6=0 sur CRTC 1 (hors C4=0 sur CRTC 1)).

CRTC	7	6	5	4	3	2	1	0
0	Sc	Sc	Sd	Sd	x	x	i	i
1	x	x	x	x	x	x	i	i
2	x	x	x	x	x	x	i	i
3	x	x	Sd	Sd	x	x	i	i
4	x	x	Sd	Sd	x	x	i	i

Other CRTC	7	6	5	4	3	2	1	0
MC6845*1	Sc	Sc	Sd	Sd	x	x	i	i
UM6845E	UpdM	US	Sc	Sd	Vdrac	Vdrad	i	i

Interlace		
0	0	No interlace
0	1	Interlace Sync Mode
1	0	No interlace
1	1	Interlace Sync & Video Mode

Skew DISPTMG		
0	0	Non Skew
0	1	One-character skew
1	0	Two-character skew
1	1	Non-output

Skew CUDISP		
0	0	Non Skew
0	1	One-character skew
1	0	Two-character skew
1	1	Non-output

19.2 FONCTIONS « SKEW-DISPTMG »

Ces fonctions sont disponibles uniquement sur les CRTC 0, 3 et 4.
Elles permettent d'activer le BORDER ou de générer un délai sur la gestion du BORDER R1.

Remarque : Si la fonction BORDER ON est activée, la fonction « Interlace » sur les 2 bits de poids faible n'est pas prise en compte. (ce point demande des investigations complémentaires).

19.2.1 BORDER ON

Cette fonction est activée avec **00110 0xx** sur R8.
Elle permet de désactiver l'affichage des caractères.
Le GATE ARRAY génère du BORDER dans cette situation.
La mise à jour du signal DISPEN (DISPLAY OFF) est immédiatement prise en compte.
Le pointeur de VRAM continue néanmoins d'être incrémenté et le pointeur courant est mis à jour lorsque C0=R1 et C9=C0=0.
Cette fonction n'affecte pas l'état BORDER R6 (lorsque C4=R6) et il est donc possible de basculer sur un des 3 autres états disponibles.

Remarque : Sur CRTC 1, cette affectation directe de DISPEN est possible en mettant R6 à 0.
Cependant, si C4=0, alors la condition C4=R6 est remplie et cela provoque la fin d'affichage jusqu'à ce que C4=C9=C0=0. Ce problème n'existe pas avec la fonction BORDER ON.

19.2.2 BORDER OFF

Cette fonction est activée avec **00000 0xx** sur R8.
Elle indique de cesser la gestion des autres fonctions "SKEW".

À tester : positionner la fonction 001100xx , puis attendre que C4=R6 pour activation du BORDER, et repositionner R8 000000xx pour voir si cela peut annuler un BORDER R6.

19.2.3 BORDER DELAI +1 / +2

Ces fonctions sont activées avec **00010 0xx** & **00100 0xx**
Elles permettent de gérer un délai sur la gestion du BORDER R1.

Sans que cette fonction soit activée, on a:

- La condition de désactivation du BORDER (début de ligne) est réalisée sur le caractère qui suit **C0=R0** (C0=0).
- La condition d'activation du BORDER (fin de ligne) est réalisée sur le caractère **C0=R1**.

Le délai peut être de 1 ou 2 caractères CRTC (donc 1 ou 2 µsec) selon la fonction utilisée.
Les conditions de gestion du BORDER **restent les mêmes** que celles décrites précédemment.
Cependant, un compteur de prise en compte à partir de C0 est appliqué selon la fonction :

Lorsque le "délai" programmé est de 1 µsec :

- Le BORDER est désactivé sur le 2ème caractère après celui ou C0=R0 (C0=1).
- Le BORDER est activé sur le 1er caractère après celui ou C0=R1.
  Remarque : Si R1=R0, alors le BORDER est activé sur C0=0.

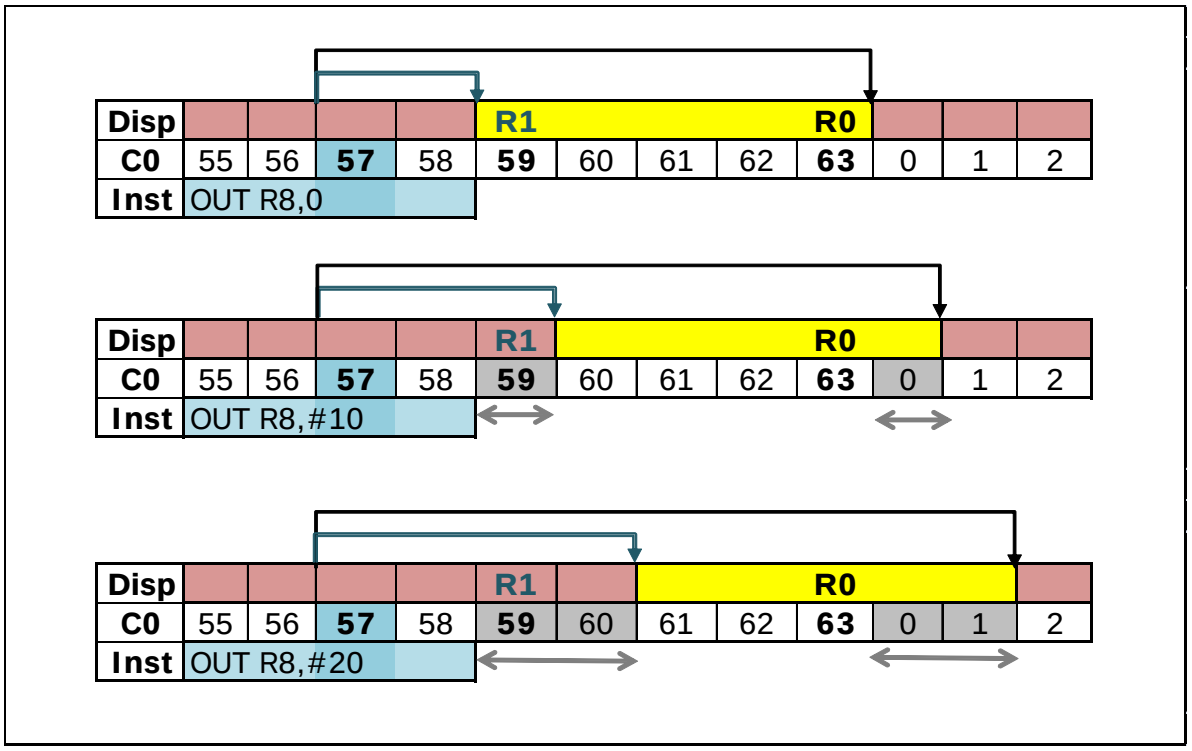
Lorsque le "délai" programmé est de 2 µsec :

- Le BORDER est désactivé sur le 3ème caractère après celui ou C0=R0 (C0=2).
- Le BORDER est activé sur le 2ème caractère après celui ou C0=R1.
  Remarque : Si R1=R0, alors le BORDER est activé sur C0=1.

L'affectation du pointeur vidéo, lorsque C9=R9, suit la même logique lorsque C0 atteint la valeur relative de C0 par rapport à R1.
Autrement dit, le pointeur vidéo est mémorisé en relation avec la nouvelle position du BORDER.

La prise en compte de ces nouvelles "règles" de test lorsque R8 est modifié est immédiate durant la ligne.

Le schéma ci-dessous décrit la position du BORDER selon la programmation des registres, pour un cas standard. R1=59, R0=63, avec une modification de R8 intervenue ailleurs que sur les positions correspondantes de C0.



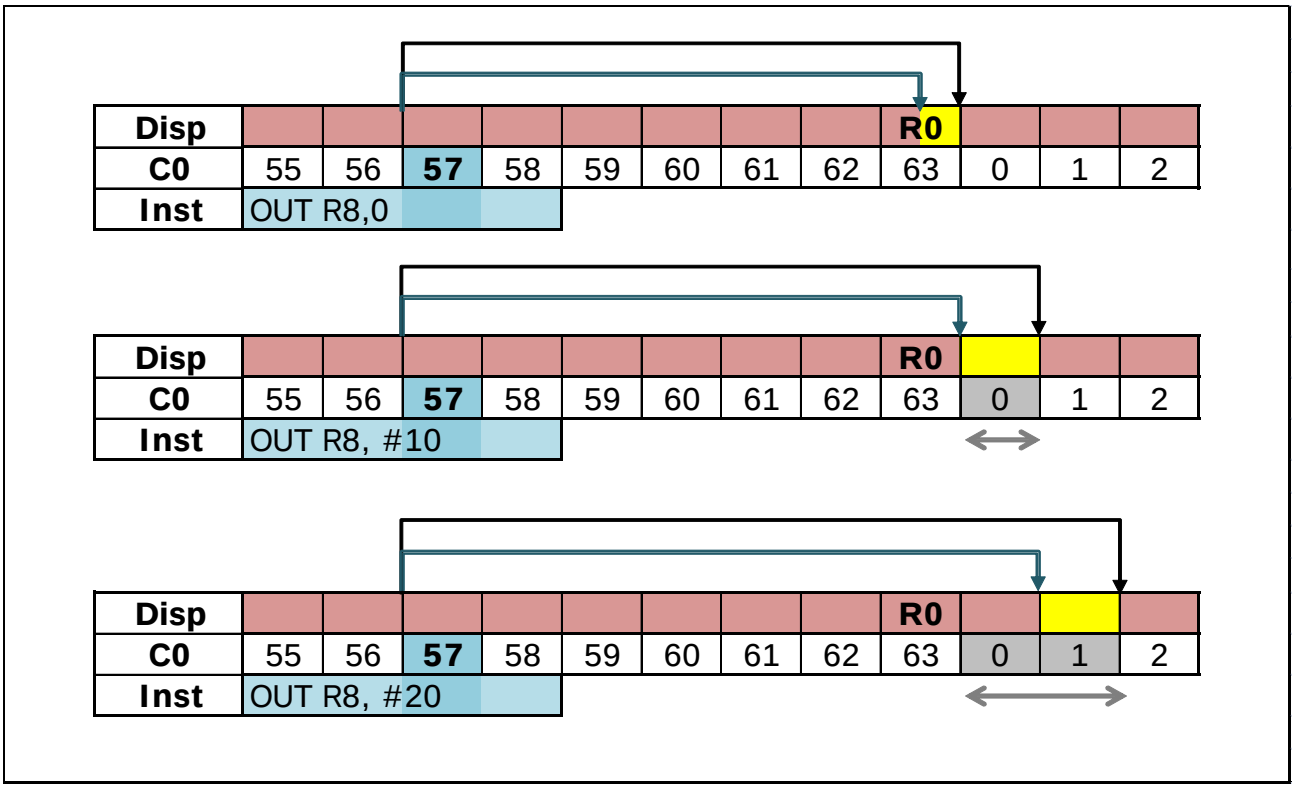
	Character Display
	Border Display
	Gap Skew ⟷
→	Border ON on C0=R1 applied
- - - →	Border ON on C0=R1 not applied
→	Border OFF on C0=R0 applied
- - - - - - →	Border OFF on C0=R1 not applied

19.2.4 ABSENCE DE CONDITION C0=R1

Dans le chapitre consacré au registre R1 (17.2) nous avons vu que sur les CRTC 0 et 2, un octet de BORDER est généré entre "2 lignes" lorsque R1 est supérieur à R0.

La condition C0=R1 n'étant pas rencontrée durant la ligne, le signal BORDER est alors envoyé 0.5µsec après que la condition C0=R0 soit satisfaite. Le BORDER est désactivé sur le caractère suivant, 0.5µsec plus tard. La condition C0=R0 se substitue donc à la condition C0=R1.

Remarque : Sur les CRTC 1, 3 et 4, dans le même contexte (R1>R0, avec par exemple R0=63, R1=64), la condition C0=R1 n'est pas rencontrée, et le CRTC n'envoie pas de signal BORDER ON au GATE ARRAY.
Lorsqu'un retard est programmé à l'aide des fonctions SKEW DISP, le délai est comptabilisé sur la base des transitions de C0. Autrement dit, le BORDER « différé » sera affiché au début du caractère, comme il l'aurait été sur la condition C0=R1.



19.2.5 DÉSINTÉGRATION DU BORDER SUR CRTC 0

Il existe une période de 1 ou 2 µs, selon la fonction utilisée, durant laquelle il est possible "d'annuler" les 2 conditions. Le CRTC 0 a la possibilité de supprimer cet octet de BORDER en annulant les 2 conditions. Le CRTC 2 ne dispose pas des fonctions SKEWDISP et ne peut donc pas utiliser cette méthode pour faire disparaitre cet octet de BORDER entre 2 lignes.

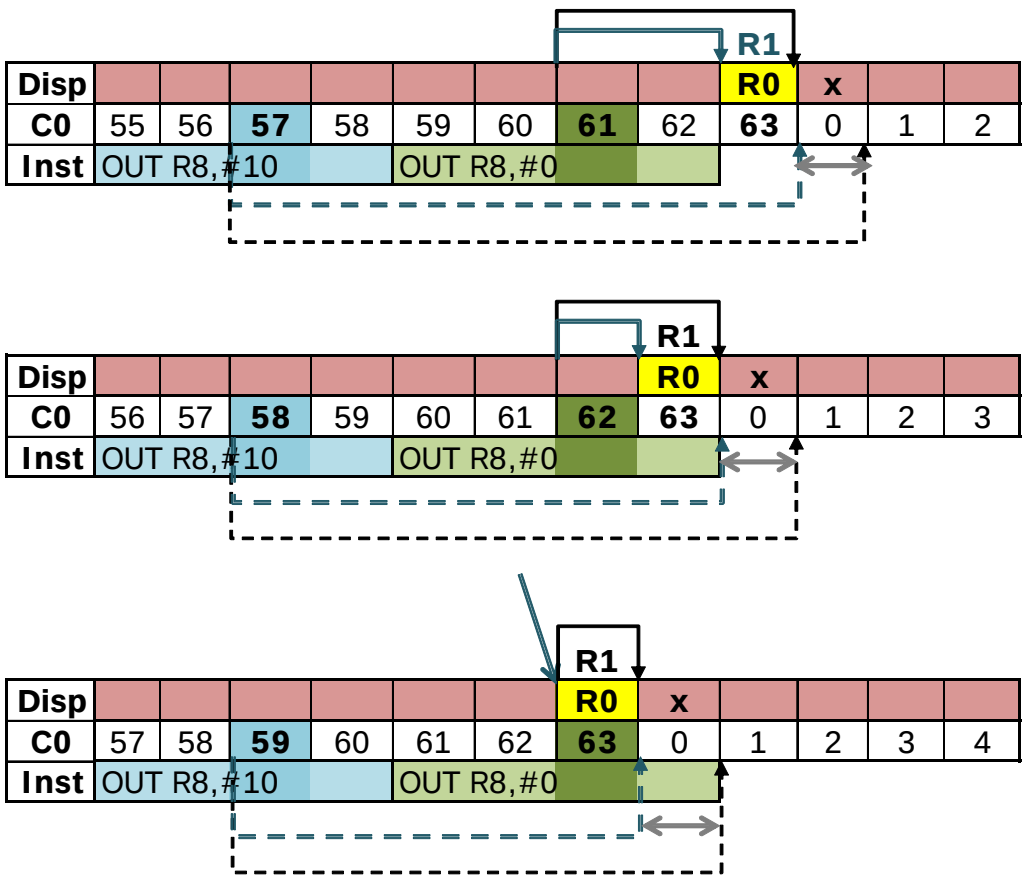
Remarque : Cette méthode est utilisable également lorsque R1>R0. La condition C0=R1 est juste remplacée par la condition C0=R0 dans ce cas.

Pour les différentes situations présentées ci-après, on considère que :

- R0=R1=63.
- La programmation de R8 est réalisée sur plusieurs lignes. Ainsi la dernière valeur programmée sur la ligne est celle présente lorsque le 1er OUT est réalisé de la ligne suivante.

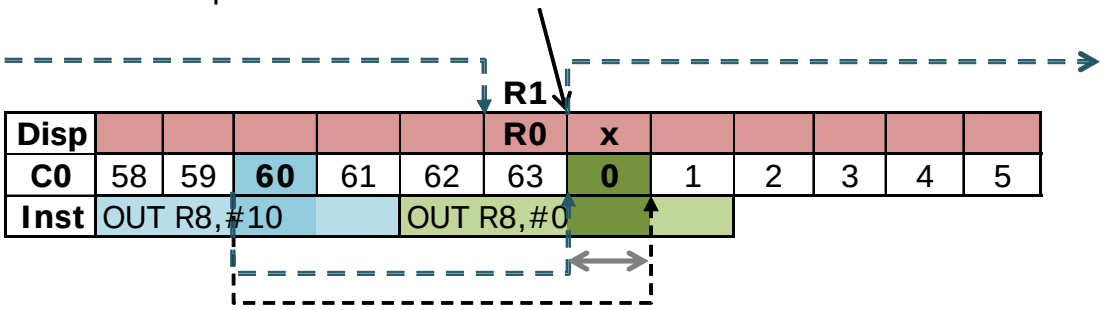
19.2.5.1 SKEW DISP+1 est annulé sur C0<=63.

La gestion du BORDER est traitée avec le 2ème OUT.
La programmation du 1er OUT (R8=#10) est ignorée.



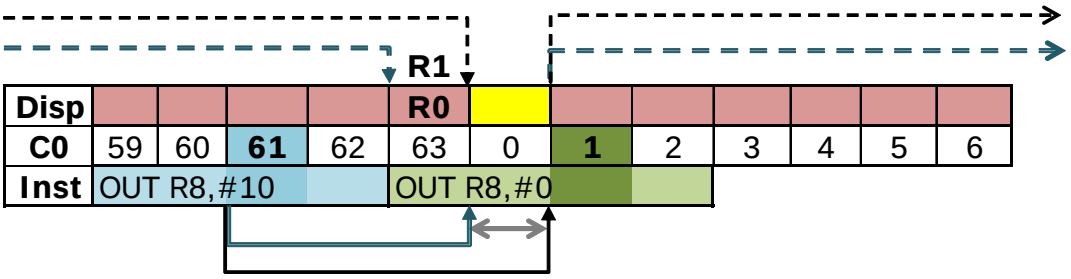
19.2.5.2 SKEW DISP+1 est annulé sur C0=0 : Y GN'EST GNOU ?

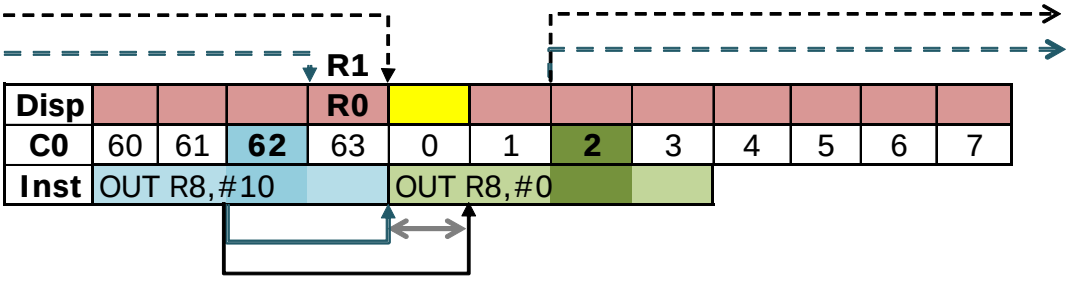
Le premier OUT (R8=#10) empêche le BORDER d'être activé sur C0=63 (Condition C0=R1 qui reporte le BORDER pour C0=0). Cependant, le 2ème OUT (R8=#00) sur C0=0 définit un BORDER OFF sur le caractère suivant C0=R0. Il est pris en compte immédiatement et annule la condition C0=R1 programmée sur le premier OUT.



19.2.5.3 SKEW DISP+1 est programmé sur C0<63.

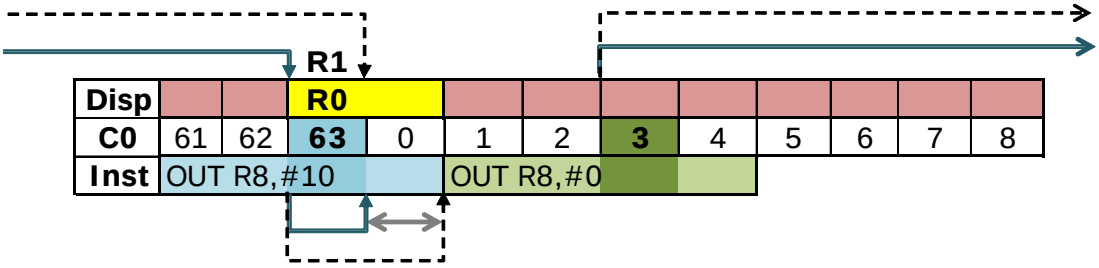
Le second OUT (R8=#0) survient trop tard pour empêcher le BORDER programmé avec le premier OUT d'être activé sur le caractère suivant C0=63 (Condition C0=R1 qui reporte le BORDER pour C0=0).





19.2.5.4 SKEW DISP+1 est programmé sur C0=63 : COMBO BORDER !

Avant d'être programmé avec #10, R8 valait 0, et le BORDER était programmé pour C0=R1 (soit C0=63). La programmation de R8 avec #10 n'a pas été suffisamment rapide pour empêcher le BORDER d'être activé. Cependant cette programmation est prise en compte pour le caractère suivant, ce qui se traduit par un octet de BORDER complémentaire.



19.3 FONCTIONS INTERLACE

19.3.1 GÉNÉRALITÉS

La première chose qu'on peut dire, c'est que la méthode d'approche du sujet par les différents fabricants de CRTC est assez... variée.

Ce premier chapitre concerne le principe de fonctionnement tel qu'il est décrit (très superficiellement) dans les guides techniques des composants. Les chapitres consacrés à cette fonction seront complétés dans des versions plus avancées de ce document (n'ayant pas encore terminé l'analyse de certains résultats pour des raisons de fond (ah ah)).

Il existe 2 modes interlacés programmables sur tous les CRTC.

L'objectif principal du mode interlacé est de créer un "frame" composé de 2 frames à 50Hz, « entrelacés », afin d'afficher une image avec une résolution verticale doublée. Une image "complète" est composée de 2 "frame" de 50 Hz afin d'afficher une image complète à 25 Hz en 625 lignes (30 Hz en 525 lignes).

À cette fin, le CRTC doit :

- Etre capable de créer une image composée de 625 lignes distinctes (sur un écran 50Hz)
- Afficher deux frames de 312 lignes, plus une ligne additionnelle.
  La longueur d'un frame étant alors de 19968 + 32 = 20000 µsec.

Afin d'augmenter la résolution verticale, le CRTC, lors de la VSYNC, retarde le signal de la moitié d'une ligne lorsque C4 passe à R7 pour le frame affiché avec les lignes paires. Lorsque C4 atteint R7, le signal VSYNC du CRTC est généré en prenant C0=R0/2 comme nouvelle référence

Si le CRTC communiquait directement avec l'écran CTM, **ce qui n'est pas le cas sur CPC**, cela aurait pour effet de faire remonter le canon à électrons une demi-ligne plus haut (-0.5). Mais une ligne additionnelle est ajoutée à la fin du premier frame, ce qui revient à déplacer le début du frame impair une ligne vers le bas (-0.5+1=0.5) permettant ainsi d'ordonner les lignes.

La capacité de gestion d'une image « Interlace » dépend fortement du moniteur :

- D'une part sur la capacité de persistance des phosphores, pour limiter le scintillement (que certains appelleraient "flashouille" dans certaines contrées froides et humides).
- D'autre part sur la stabilité des éléments haute tension du moniteur pour limiter les distorsions sur les bords de l'écran.

Sur CPC, le GATE ARRAY fusionne les signaux HSYNC et VSYNC en un signal composite, et la VSYNC est envoyée en même temps que la HSYNC au CTM par le GATE ARRAY. Or c'est la position de départ de la VSYNC durant la ligne qui permet de faire remonter le canon à électrons plus ou moins haut (plus la VSYNC démarre tard dans la ligne, plus le faisceau remonte haut).

La HSYNC étant (en principe) toujours située au même endroit, toutes les lignes affichées sur un moniteur CTM ont une définition de 1 pixel, ce qui contrarie la logique énoncée ci-dessus. Dans le cas de figure d'un écran CPC « Standard » avec R2=46 et R0=63, la VSYNC générée par le CRTC (sur C0=0 sur le premier frame, sur C0=31 sur le deuxième) est récupérée par le GATE ARRAY, qui va attendre la 1ère µseconde de la 2ème HSYNC pour envoyer le signal au moniteur. Le canon remonte alors exactement au même endroit que sur le frame précédent (-0.0).

La ligne additionnelle générée à la fin du premier frame positionne le frame impair 1 ligne complète vers le bas (-0.0+1=1). La première ligne du frame pair alterne alors avec la ligne additionnelle, ce qui est assez éloigné de l'effet attendu.

On a donc dans cette situation :

- 1 frame pair d'une durée de 19968 µsec, la **VSYNC** n'étant pas « décalée ».
- 1 frame impair, qui hérite de la ligne additionnelle du frame pair, et dure 20032 µsec.

Heureusement ceci est loin d'être irrémédiable (voir chapitre 15.3, page 137).

19.3.2 LES DEUX MODES INTERLACE

19.3.2.1 INTERLACE SYNC MODE : FONCTION 00xx0001

Ce mode est utilisé lorsqu'on souhaite que le CRTC affiche la même information pour le frame pair et le frame impair.

L'objectif de ce mode est d'améliorer la qualité des caractères affichés en remplissant les espaces situés entre les lignes en mode "non Interlace".

Dans ce mode, le CRTC se contente d'augmenter la résolution en déplaçant la position du signal VSYNC de 1/2 ligne, comme indiqué dans le paragraphe précédent.

L'image dans cette situation utilise la même mémoire vidéo, en affichant deux fois la même chose.
Il n'est en principe pas nécessaire dans ce mode de reprogrammer les registres du CRTC.
La ligne 0 du frame des lignes paires est affiché en premier, suivi de la ligne 0 du frame des lignes impaires, et ainsi de suite.

Screen:	Even	Odd	Beautiful drawing							
C9=	0									
C9=		0								
C9=	1									
C9=		1								
C9=	2									
C9=		2								
C9=	3									
C9=		3								
C9=	4									
C9=		4								
C9=	5									
C9=		5								
C9=	6									
C9=		6								
C9=	7									
C9=		7								

19.3.2.2 INTERLACE SYNC & VIDEO MODE : FONCTION 00xx0011

Ce mode **IVM** est qualifié de "**Video Mode**" car il peut être utilisé pour afficher des images pour lesquelles chacune des 625 lignes est différente.

Contrairement au mode "Interlace Sync Mode", aucune ligne du frame construit pour les 2 frames n'est répétée.

Sur le premier frame, le CRTC affiche les lignes pour lesquelles C9 est pair.
Sur le second frame, le CRTC affiche les lignes pour lesquelles C9 est impair.

À l'issue de l'affichage des 2 écrans (~0.04 seconde), les lignes se suivent dans l'ordre pair/impair.

Étant donné qu'il est nécessaire d'afficher 2 fois plus de lignes, il faut en conséquence programmer les registres du CRTC comme si on construisait un frame de 624 lignes.

Cette logique est mise à l'épreuve par les concepteurs des circuits.

L'ajout de la 625ème ligne est géré automatiquement par le CRTC, pour rappel.

Note: Dans les documents UM6845R dont je dispose, la figure (7) utilisée pour décrire ce mode est erronée (description des numéros de lignes affichées dans les écrans pair/impair).
Cette erreur n'existe pas dans les documents UM6845 de UMC (figure 13), qui est une copie "conforme" du HD6845S de HITACHI (et détecté comme un CRTC 0).

Screen:	Even	Odd	ArtWork from MOMA							
C9=	0									
C9=		1								
C9=	2									
C9=		3								
C9=	4									
C9=		5								
C9=	6									
C9=		7								
C9=	8									
C9=		9								
C9=	10									
C9=		11								
C9=	12									
C9=		13								
C9=	14									
C9=		15								

À cause du repositionnement de l'envoi de la **VSYNC** par le **GATE ARRAY**, le frame pair est affiché une ligne complète plus tôt que le frame impair

Dans ce contexte, la ligne 0 alterne avec une ligne de BORDER, alors que la ligne 1 alterne avec la ligne 2, la ligne 3 avec la ligne 4, et ainsi de suite...

L'image décrite précédemment est affichée comme suit. Les pixels "communs" apparaissent alors dans leur couleur d'origine, plutôt qu'en alternance avec l'encre blanche.

C9	Beautiful Drawing Even Screen							
0								
2								
4								
6								
8								
10								
12								
14								

C9	Beautiful Drawing Odd Sceen							
	BORDER or PREVIOUS LINE							
1								
3								
5								
7								
9								
11								
13								
15								

19.3.3 RESTRICTIONS

Il existe des différences de spécification sur les registres à mettre à jour (et leur valeur) en mode « Interlace » entre les différents circuits. Ces différences sont principalement liées au mode de comptage de C9 et C4 initialement défini par les concepteurs des circuits.

Bien entendu, ces restrictions sont faites pour ne pas être respectées, mais elles donnent cependant des indices intéressants sur le fonctionnement interne des compteurs.

C'est notamment le cas pour le registre R9 sur les CRTC 0, 3 et 4, qui doit être programmé avec le nombre de ligne d'un caractère vertical, moins 2 pour le mode IVM. Cela facilite la comparaison de fin des lignes paires, car il suffit d'ignorer le bit 0 pour effectuer la comparaison de fin de caractère et gérer ce bit 0 comme celui de la parité de frame.

L'affichage d'une ligne sur deux provoque l'incrémentation plus rapide de C4 (sur les CRTC 0, 1, 3, 4) et la VSYNC se produit donc plus tôt lorsque C4=R7. La construction d'un frame dans le mode IVM nécessite donc d'adapter R4 et R7 à la taille totale de l'écran désiré (sauf pour le CRTC 2, qui a adopté une autre méthodologie).

Lorsqu'un écran de grande taille est ainsi défini, il est possible d'utiliser les "**Overscan Bits**" ™ pour éviter d'avoir à reprogrammer R12/R13 entre frames pairs et impairs, mais je m'égare.

19.3.4 FONCTION MAL AIMÉE

Le mode « Interlace » est une fonction (injustement) mal aimée par les développeurs sur CPC.
Note : J'aurais du sortir la démo 4 :-)

Pourtant, cette **fonction est prise en compte en temps réel** et son intérêt n'est pas seulement lié à sa capacité à traiter « convenablement » une image « Interlace » en produisant, via le GATE ARRAY, un signal utile pour le moniteur.

Dès que la fonction est activée sur une ligne C9 donnée, cela a une incidence sur **le comptage C9 et/ou C4 pour les lignes suivantes**.

Selon la valeur de C9 et sa parité lorsque R8 est modifié, alors la ligne suivante est immédiatement calculée suivant une sauce propre à chaque CRTC.

Autrement dit, dans la perspective de création d'un « Interlace » « complet », il faut tenir compte de la nouvelle construction de l'écran à partir de la **VSYNC**, lorsque C4=R7.

Dans la perspective de n'utiliser que certaines fonctions du mode « Interlace » IVM, il est possible de d'activer et désactiver ce mode avant que certaines fonctions aient lieu.

Ainsi, le nombre de lignes d'un frame dépend du moment où le mode « Interlace » a été activé. Jusqu'à son activation le frame contient des lignes paires et impaires. Dès que le mode IVM est activé le frame contient des lignes paires et/ou impaires (selon des règles propres au CRTC) jusqu'à ce que le mode IVM soit désactivé, afin de revenir à un comptage « classique ».

L'activation du mode « Interlace », en plus de modifier le comptage C9 et C4, active deux autres gestions distinctes :

- La **gestion d'ajout d'une ligne additionnelle à la fin du premier frame**, avant que C9=C4=C0=0.

- La **gestion d'activation du signal VSYNC sur C0=R0/2 (donc sur C0=31 dans le cas d'un frame où R0=63)** lorsque C4=R7 (que je vais appeler **MID-VSYNC**).

19.4 PROGRAMMATION VERTICALE EN INTERLACE

19.4.1 CRTC 0

En **INTERLACE SYNC**, R9 doit être programmé avec la valeur N-1 si N représente le nombre de ligne(s) d'un caractère vertical C4. R4, R5, R6 et R7 doivent être programmés en considérant que les caractères C4 contiennent N lignes chacun.

En **INTERLACE VIDEOMODE** (IVM), R9 doit être programmé avec la valeur N-2 si N représente le nombre de ligne(s) d'un caractère vertical. Le CRTC 0 partage cette particularité avec les CRTC 3 et 4. R4, R6 et R7 doivent être programmés en considérant que les caractères contiennent 2 fois moins de lignes. Lorsque N est impair, une logique particulière d'équilibrage a lieu (voir chapitre 19.5.2). R5 contient toujours un nombre fini de lignes sans tenir compte du mode « Interlace ».

Si un caractère vertical C4 fait 8 lignes, il faut que R9 contienne 6.
Si R9 est programmé avec 0, cela signifie qu'il y a 2 lignes minimum affichées (1 par frame).

19.4.2 CRTC 1

Si N représente le nombre de ligne(s) d'un caractère vertical, alors R9 doit être programmé avec la valeur N-1 dans les 2 modes « Interlace ».

En **INTERLACE SYNC**, R4, R5, R6 et R7 doivent être programmés en considérant que chaque caractère C4 contient N lignes.

En **INTERLACE VIDEOMODE** (IVM), R4, R6 et R7 doivent être programmés en considérant que les caractères C4 contiennent 2 fois moins de lignes. Lorsque N est impair, une logique particulière d'équilibrage a lieu (voir chapitre 19.5.3). R5 contient toujours un nombre fini de lignes sans tenir compte du mode « Interlace ».

19.4.3 CRTC 2

Si N représente le nombre de ligne(s) d'un caractère vertical, alors R9 doit être programmé avec la valeur N-1 dans les 2 modes « Interlace ».

En **INTERLACE SYNC**, R4, R6 et R7 doivent être programmés en considérant que les caractères C4 contiennent N lignes chacun. R5 contient toujours un nombre fini de lignes sans tenir compte du mode « Interlace ».

En **INTERLACE VIDEOMODE** (IVM), R4, R6 et R7 doivent être programmés en considérant que les caractères C4 contiennent N lignes chacun. R5 contient toujours un nombre fini de lignes sans tenir compte du mode « Interlace ».

Pour chaque valeur de C4, deux caractères de N/2 lignes sont traités.
Un compteur C9 spécifique (**C9.IVM**) est géré pour l'affichage. Ce compteur est initialisé lorsque C9 passe à 0, mais également lorsqu'il atteint la valeur de R9 « hors parité », afin de mettre à jour le pointeur vidéo VMA' sans que C4 soit incrémenté. (voir chapitre 19.8.3).

Lorsque R9=7, on obtient donc des caractères C4 de 8 lignes pour chaque frame avec un changement d'adresse toutes les 4 lignes.

Cela simplifie grandement la programmation de R4 et R7 dans les situations les plus courantes, puisque le nombre de lignes total à afficher durant un frame est équivalent à un mode non « Interlace ».

Ainsi R6 concerne 1 « double » caractère de 8 lignes et non 1 caractère de 4 lignes.

Cette gestion « Interlace » **pose cependant quelques problèmes lorsque R9 est pair.**
Dans cette situation, un **nombre impair de lignes** est généré pour chaque caractère C4 sur chaque frame.

Une ligne de plus que prévue est donc affichée sur les frames impair.
Si R9 vaut 6, par exemple, la ligne C9=7 sera tout de même affichée sur les frames impairs.
De plus, la mise à jour du pointeur vidéo (lorsque C0==R1) n'a plus lieu lorsque C4 s'incrémente car elle a lieu uniquement lorsque C9.IVM=R9 (hors parité). (voir chapitre 19.8.3).

C4 continue à être géré normalement, mais l'affichage est incorrect.

Les autres CRTC permettent de gérer indistinctement des caractères avec un nombre de lignes pair ou impair, moyennant une gestion sportive de la parité du frame, des compteurs et des registres.

C'est donc une légende urbaine de considérer que le CRTC 2 gère le mode « Interlace » mieux que les autres CRTC. Il est certes plus simple de modifier uniquement R8 sans avoir à modifier R4, R5, R6 ou R7, mais il est impossible de définir des caractères C4 composés de lignes impaires sans que ce décompte soit aligné sur un nombre pair durant les frames impairs et que cela provoque un grave défaut d'affectation du pointeur vidéo à chaque nouveau C4.

19.4.4 CRTC 3 & 4

En **INTERLACE SYNC**, R9 doit être programmé avec la valeur N-1 si N représente le nombre de ligne(s) d'un caractère vertical C4. R4, R5, R6 et R7 doivent être programmés en considérant que les caractères C4 contiennent N lignes chacun

En **INTERLACE VIDEOMODE** (IVM), R9 doit être programmé avec la valeur N-2 si N représente le nombre de ligne(s) d'un caractère vertical. Les CRTC AMSTRAD partagent cette particularité avec le CRTC 0. R4, R6 et R7 doivent être programmés en considérant que les caractères contiennent 2 fois moins de lignes. Lorsque N est impair, une logique particulière d'équilibrage a lieu (voir chapitre 19.5.5). R5 contient toujours un nombre fini de lignes sans tenir compte du mode « Interlace ».

Si un caractère vertical C4 fait 8 lignes, il faut que R9 contienne 6.
Si R9 est programmé avec 0, cela signifie qu'il y a 2 lignes minimum affichées (1 par frame).

19.5 PARITÉ

19.5.1 GÉNÉRALITÉS

En « Interlace », la parité du frame joue un rôle important car elle participe à :

- L'ajout de la ligne additionnelle lorsque la construction du frame pair est terminée.
- La génération d'une **MID-VSYNC** lorsque C4=R7 et C0=R0/2 sur le frame pair.
- Au calcul des C9 pairs et impairs.

Le point commun entre tous les CRTC est de disposer d'un état interne pour le frame qui bascule de pair à impair et vice-versa selon différentes conditions. À noter que sur les CRTC 0, 2, 3 et 4, il existe non pas un, mais plusieurs états distincts de gestion parité (voir chapitre 19.5.5)

19.5.2 CRTC 0

Il existe plusieurs états de gestion de parité **:**

- **ParitéFrame=ParitéR6 lorsque C4=C9=C0=0**. Cet état définit la parité du premier C9 du frame.

- **ParitéC9** qui définit le bit 0 des C9 en mode « Interlace »

- **ParitéR6=ParitéFrame xor 1** lorsque **C4 atteint R6**.

L'état **ParitéR6** permet d'anticiper la parité du prochain frame. Si **ParitéFrame** est pair, alors **ParitéR6** sera impair et vice-versa. Si C4 ne peut pas atteindre R6 (car R6>R4) cet état n'est plus mis à jour et **ParitéFrame** reste figé avec la dernière valeur de **ParitéR6**.

La gestion de **ParitéR6** est indépendante de la valeur de R8.

L'état **ParitéR6** permet également de savoir si une ligne additionnelle sera générée à la fin du frame (voir chapitre 19.6.1)

Au début du frame, **ParitéFrame=ParitéR6**.
La parité de C9 est gérée uniquement lorsque R8=3 pour mettre à jour C9.

En IVM, R9 est programmé avec un nombre **pair** pour définir un nombre **pair** de lignes dans un caractère (Par exemple R9=6 pour obtenir 2x4 ligne/car=8 lignes). Ceci est lié à la méthode utilisée pour tester la fin d'un caractère composé uniquement de lignes paires ou de lignes impaires selon la parité courante. La parité C9 est alors identique pour toutes les valeurs de C4.

Lorsque R9 est **impair** en IVM (à contrario du CRTC 1), cela implique une différence entre le nombre de lignes paires et impaires pour un caractère entre 2 frames (le nombre total de lignes d'un caractère vertical est impair). Il y a plus de lignes paires que de lignes impaires. Dans cette circonstance, les concepteurs du circuit ont équilibré le nombre de lignes entre 2 frames afin d'éviter qu'il y ait un frame avec R4+1 lignes de plus sur un frame pair que sur un frame impair.

La solution adoptée consiste à alterner des caractères composés de lignes paires avec des caractères composés de lignes impaires sur un même frame **à chaque fois que C4 évolue**.

Ainsi, lorsque **R9 est impair**, **la parité des lignes dépend de celle de C4 et de la parité courante du frame**.

La parité de C9 est calculée ainsi lorsque R8 vaut 3:
ParitéC9=C4.0 xor ParitéFrame
(0=Pair/1=Impair)

Ainsi, sur un frame pair, la parité de C9 est égale à celle de C4.
Sur un frame impair, la parité de C9 est l'inverse de celle de C4.

Tous les C4 pairs, l'équilibre des lignes est préservé.

Exemple :
R9=7. Sur un frame pair sur C4=0, le CRTC génère 5 lignes paires (0.2.4.6.8) suivies de 4 lignes impaires (1.3.5.7). Sur un frame impair sur C4=0, le CRTC génère 4 lignes impaires suivies de 5 lignes paires. Tous les C4 pairs, l'équilibre des lignes est ainsi maintenu à 9 lignes.

Pour être totalement satisfaisante, cette méthode implique une gestion spécifique de la VSYNC en mode IVM. En effet, si R7 est programmé sur un C4 impair, alors la VSYNC est retardée de 1 ligne. Elle se produit lorsque C4==R7 et C9.VMA==2 sur les C4 impairs. Ceci permet d'éviter un déphasage avec la VSYNC générée sur ce même C4 sur un frame pair.

	PARITYFRAME=ODD				PARITYFRAME=EVEN		
	C4	C9			C4	C9	
	0	0	R8=3		0	0	
	0	3			0	2	
	0	5			0	4	
	0	7			0	6	
	1	0			0	8	
VSYNC	1	2	R7=1		1	1	VSYNC
	1	4			1	3	
	1	6			1	5	
	1	8			1	7	
VSYNC	2	1	R7=2		2	0	VSYNC
	2	3			2	2	
	2	5			2	4	
	2	7			2	6	
	3	0			2	8	
VSYNC	3	2	R7=3		3	1	VSYNC
	3	4			3	3	
	3	6			3	5	
	3	8			3	7	
VSYNC	4	1	R7=4		4	0	VSYNC
	4	3			4	2	

Remarque :
Si R8 passe à 3 sur un C4 impair, cela peut provoquer un déphasage de 1 ligne entre les VSYNC des frames pairs et impairs. En effet, cette technique de décalage de VSYNC utilisée par les ingénieurs qui ont conçu le circuit ne fonctionne correctement que pour gérer le déséquilibre de ligne entre un C4 pair et un C4 impair.

Par exemple, si R9=7 et que R8 passe à 3 sur C4=1 (C9=0), le nombre de lignes sur C4=0 sera identique quelque soit la parité du frame (à savoir 8 lignes), mais il sera différent sur C4=1 (soit 5 lignes paires sur un frame impair, soit 4 lignes impaires sur un frame pair).
Ce déphasage dans le décompte de lignes entre les 2 frames déphase la VSYNC :

	PARITYFRAME=ODD			PARITYFRAME=EVEN		
	C4	C9		C4	C9	
	0	0		0	0	
	0	1		0	1	
	0	2		0	2	
	0	3		0	3	
	0	4		0	4	
	0	5		0	5	
	0	6		0	6	
	0	7		0	7	
	1	0	R8=3	1	0	
	1	2		1	3	
	1	4		1	5	
	1	6		1	7	
	1	8	R7=2	2	0	VSYNC
VSYNC	2	1	R7=2	2	2	
	2	3		2	4	
	2	5		2	6	
	2	7		2	8	
	3	0	R7=3	3	1	VSYNC
VSYNC	3	2	R7=3	3	3	
	3	4		3	5	
	3	6		3	7	
	3	8	R7=4	4	0	VSYNC
VSYNC	4	1	R7=4	4	2	
	4	3		4	4	
	4	5		4	6	
	4	7		4	8	
	5	0	R7=5	3	1	VSYNC
VSYNC	5	2	R7=5	3	3	

Remarque : Sur un frame pair, la VSYNC se produit en milieu de ligne sur C0=R0/2.

Il existe plusieurs méthodes pour se positionner sur une parité précise.

Il est possible de connaitre la parité courante étant donné qu'il y a un bug de comptage lorsque le mode IVM est activé sur C9=R9 avec une parité impaire, ou lorsque le mode IVM est désactivé sur C9.VMA=R9+1 (voir chapitre 19.6.1).

Une fois que la parité est identifiée, il suffit de laisser C4 atteindre R6 une fois pour que la parité s'inverse.

19.5.3 CRTC 1

Il existe deux états de parité :

- **ParitéFrame** permute entre chaque frame **lorsque C4=C9=C0=0**
- **ParitéC9** permute entre chaque C4 si **R9 est pair**.

Lorsqu'un frame débute, l'état **ParitéFrame** permute de pair à impair et vice-versa, **quelque soit la valeur de R8**.

Si **ParitéFrame** est pair, alors une **ligne additionnelle** et une **MID-VSYNC** sont programmées.
Si **ParitéFrame** est impair, alors **pas de ligne additionnelle** et **pas de MID-VSYNC**.

Au début du Frame, **ParitéC9=ParitéFrame**.

Lorsque **R9 est pair**, le nombre de lignes d'un caractère vertical est **impair**. Dans cette circonstance, les concepteurs du circuit ont **équilibré** le nombre de lignes entre 2 frames afin d'éviter qu'il y ait un frame avec R4+1 lignes de plus sur un frame pair que sur un frame impair.

La solution adoptée consiste à alterner des caractères composés de lignes paires avec des caractères composés de lignes impaires dans un même frame. Ainsi, lorsque **R9 est pair**, **la parité des lignes dépend de celle de C4 et de la parité courante au début du frame.**

Cependant, contrairement à ce qui a été fait sur CRTC 0, 3 et 4, il n'y a aucune gestion spécifique de la **VSYNC** en mode IVM lorsque le nombre de ligne d'un caractère est impair. La **VSYNC** n'est pas retardée d'une ligne sur les C4 impairs lorsque R9 est pair. Il est impossible de positionner R7 sur un C4 impair sans créer un écart de 1 ligne sur la VSYNC entre 2 frames :

	EVEN FRAME				ODD FRAME	
	C4	C9			C4	C9
+32	0	0	VSYNC	VSYNC	0	1
	0	2			0	3
	0	4			0	5
	0	6			0	7
	0	8		VSYNC	1	0
+32	1	1	VSYNC		1	2
	1	3			1	4
	1	5			1	6
	1	7			1	8
+32	2	0	VSYNC	VSYNC	2	1
	2	2			2	3
	2	4			2	5
	2	6			2	7
	2	8		VSYNC	3	0
+32	3	1	VSYNC		3	2
	3	3			3	4
	3	5			3	6
	3	7			3	8
+32	4	0	VSYNC	VSYNC	4	1
	4	2			4	3
	4	4			4	5
	4	6			4	7
	4	8		VSYNC	5	0
+32	5	1	VSYNC		5	2
	5	3			5	4

Lorsque R8 est modifié (R8 passe de 0 à 3 ou inversement), la parité et/ou le bit 0 de C9 sont mis à jour selon des règles qui font intervenir la parité courante, la parité de C9 avant le passage en IVM et la parité de C4 si R9 est pair.

Ces mises à jour s'effectuent sur les 3ème et la 4ème µsecondes de l'instruction OUT(C),C (sur R8).

Sur la 3ème µseconde lorsque R8 passe de 3 à 0 ou de 0 à 3:

- Si R9 est pair, les parités de C4 et C9 sont utilisées pour définir la parité du nouveau C9.
- Si R9 est impair, seule la parité de C9 est utilisée pour définir la parité du nouveau C9.

ParitéC9=C9.0
ParitéC9=ParitéC9 xor (C4.0 and not(R9.0))

Il faut noter que **ParitéFrame** n'est pas modifiée.

Sur la 4ème µseconde lorsque IVM devient actif (OUT R8,3) :

- Si **ParitéFrame** est pair et R9 est pair, alors **ParitéC9** se calque sur la parité de C4.
- Si **ParitéFrame** est pair et R9 est impair, alors **ParitéC9** est pair.
- **ParitéFrame** devient paire, sauf pour les cas où **ParitéFrame** et **ParitéC9** étaient impairs avant la demande de passage en IVM.

```
Si (ParitéFrame==PAIR)
Alors
      ParitéC9=C4.0 and (not R9.0)
Fin Si
ParitéFrame=ParitéFrame and (ParitéC9 xor (C4.0 and (not R9.0)))
```

Sur la 4ème µseconde lorsque IVM devient inactif (OUT R8,0) :

Lorsque le mode IVM est désactivé (OUT R8,0), alors **ParitéFrame** devient égale à **ParitéC9**.

ParitéFrame=ParitéC9

Ainsi, si le mode IVM est activé et désactivé sur une ligne C9 paire, quelle que soit la valeur de R9, la parité est fixée à PAIR. Il est ainsi possible de fixer la parité assez facilement sur ce CRTC.

ParitéFrame s'inverse à chaque nouveau frame.

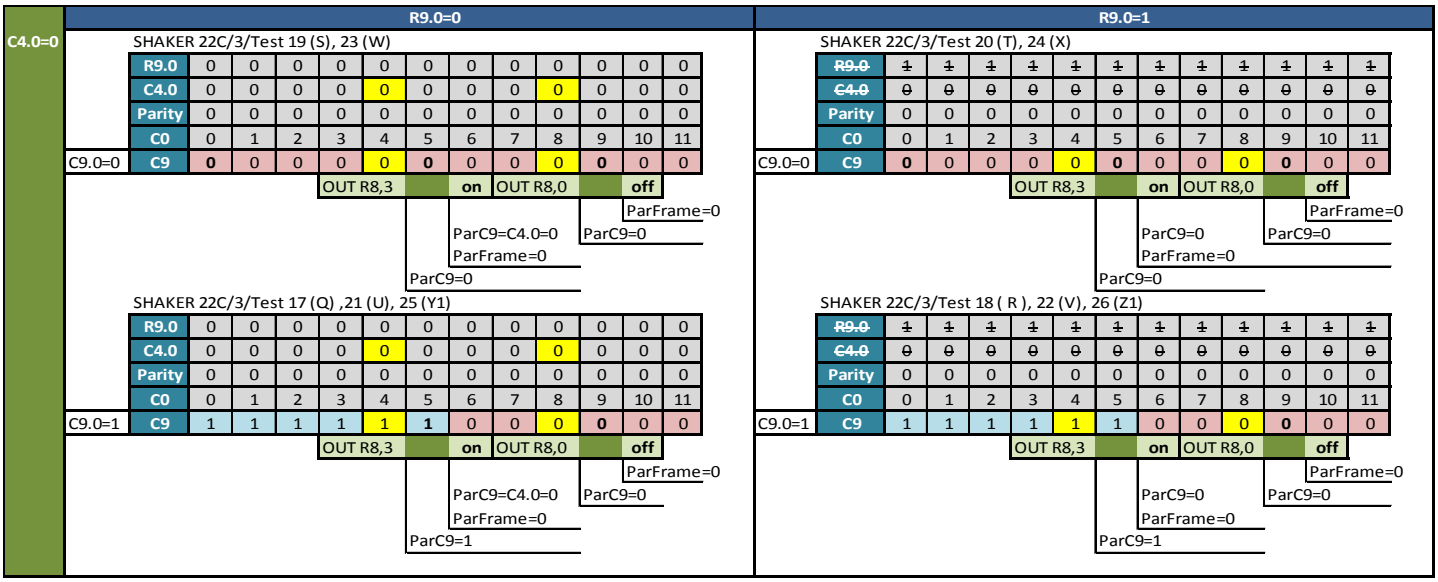
ParitéC9 s'inverse à chaque incrémentation de C4 lorsque R9 est pair.

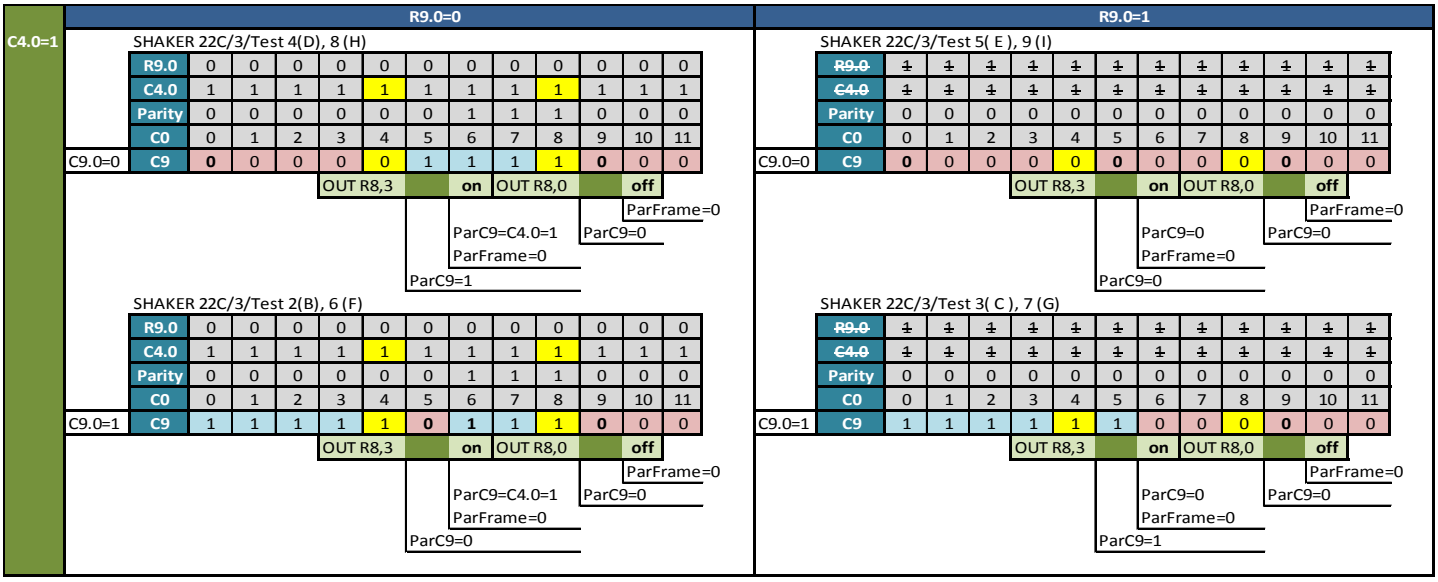
Il est nécessaire de tenir compte de ces permutations si le mode IVM est actif durant plusieurs frames. Le bit 0 de R9 est pris en compte immédiatement. Le modifier avant ou après l'activation du mode IVM a donc une incidence sur C9. Inversement, désactiver le mode IVM peut également modifier C9, et affecter la condition de fin de caractère C4.

Les schémas ci-après décrivent l'ensemble des cas de figure qu'il est possible de rencontrer lorsque le mode IVM est activé, puis désactivé.

Parité Initiale : Paire

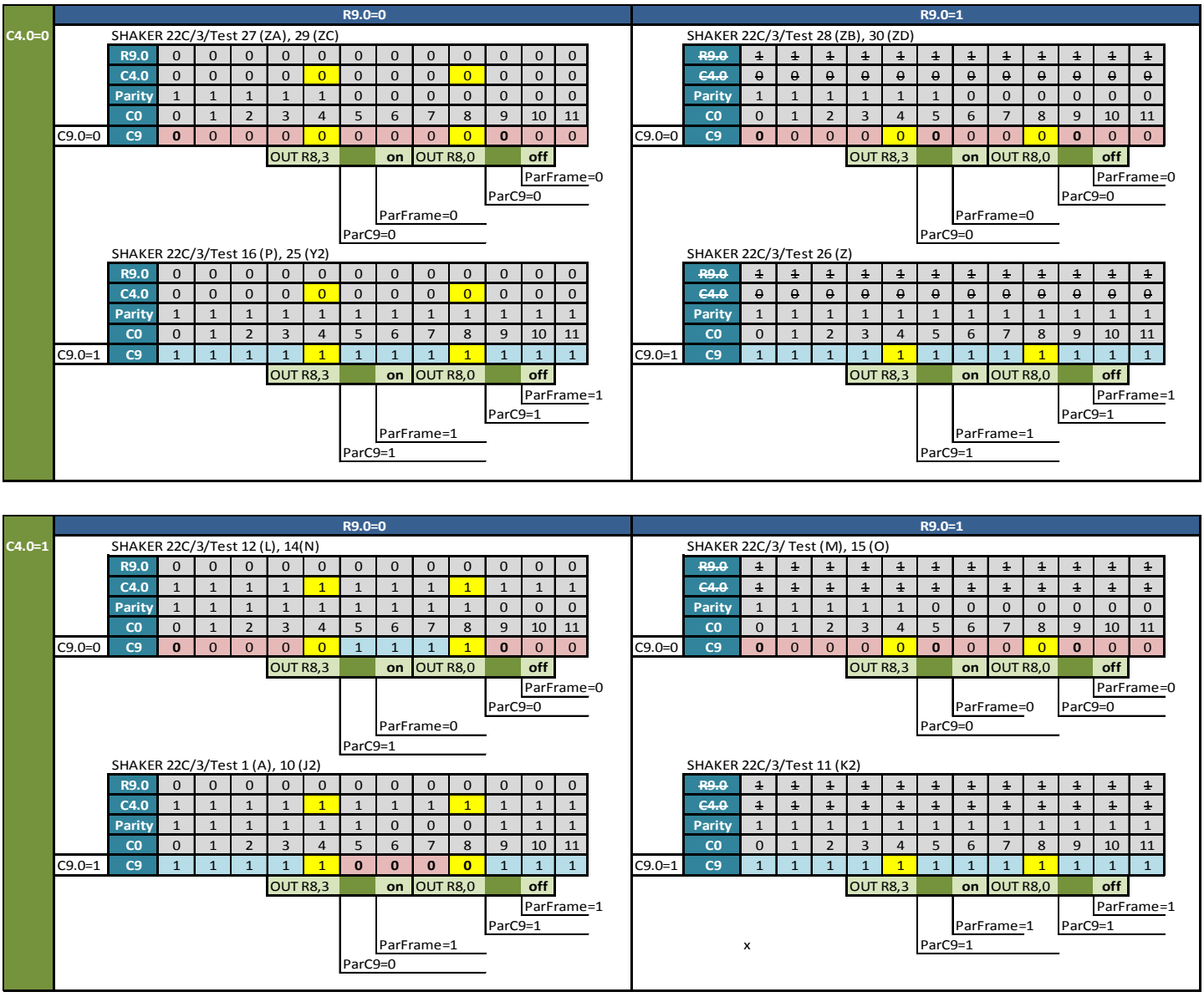
Les schémas ci-dessous décrivent l'état de la parité frame et de la parité de C9 pour une parité courante paire, selon la parité du C9 précédent et de celle de C4 (lorsque R9 est pair), au moment ou se produit un OUT R8, 3 suivi d'un OUT R8,0.





Parité initiale : Impaire

Les schémas ci-dessous décrivent l'état de la parité frame et de la parité de C9 pour une parité courante impaire, selon la parité du C9 précédent et de celle de C4 (lorsque R9 est pair), au moment ou se produit un OUT R8, 3 suivi d'un OUT R8,0.



19.5.4 CRTC 2

Il existe plusieurs états de gestion de parité **:**

- **ParitéFrame=ParitéR6 lorsque C4=C9=C0=0**. Cet état définit la parité de tous les C9 du frame**.**

- **ParitéR6=ParitéFrame xor 1** lorsque **C4 atteint R6**.

L'état **ParitéR6** permet d'anticiper la parité du prochain frame. Si **ParitéFrame** est pair, alors **ParitéR6** sera impair et vice-versa. Si C4 ne peut pas atteindre R6 (car R6>R4) cet état n'est plus mis à jour et **ParitéFrame** reste figé avec la dernière valeur de **ParitéR6**.

La gestion de **ParitéR6** est indépendante de la valeur de R8.

L'état **ParitéR6** permet également de savoir si une ligne additionnelle sera générée à la fin du frame (voir chapitre 19.6.3).

Au début du frame, **ParitéFrame=ParitéR6**.
La parité de C9 est gérée uniquement lorsque R8=3 pour mettre à jour C9.

Contrairement aux autres CRTC, la gestion de C9 a été réalisée de manière simple, non sans introduire cependant quelques contraintes (R9 impair par exemple).

La parité est respectée quelles que soient les valeurs de R9 et de C4.
Sur les autres CRTC, R9 définit un nombre total de lignes à partager entre 2 frames, ce qui pose des problèmes dès que ce nombre de lignes est impair, et nécessite quelques cabrioles afin d'équilibrer les lignes entre 2 frames et traiter convenablement la VSYNC.

La gestion de la parité pour le calcul de C9 est prise en compte **immédiatement** à partir du 3ème nop de l'instruction OUT(C),r sur R8 :

EVEN PARITY FRAME

C0	0	1	2	3	4	5	6	7	8	9	10	11
C9	3	3	3	3	3	3	3	3	3	3	3	3
C9.Vma	3	3	3	3	3	6	6	6	6	3	3	3
				OUT R8,3			on	OUT R8,0			off	

ODD PARITY FRAME

C0	0	1	2	3	4	5	6	7	8	9	10	11
C9	3	3	3	3	3	3	3	3	3	3	3	3
C9.Vma	3	3	3	3	3	7	7	7	7	3	3	3
				OUT R8,3			on	OUT R8,0			off	

Le CRTC 2 ne peut peut-être pas faire du **SPLITBORDER**, mais il peut faire du **SPLITC9** !

Il est possible de tester l'existence de la ligne additionnelle afin de déterminer la parité. Si le mode IVM est activé sur la première ligne d'un frame impair, alors cette ligne deviendra une ligne additionnelle, et une nouvelle ligne 0 suivra l'ancienne ligne 0, ce qui allongera la taille du frame de R0 µsec.

19.5.5 CRTC 3 & 4

Il existe deux états de parité :

- **ParitéFrame** permute entre chaque frame **lorsque C4=C9=C0=0**
- **ParitéC9** permute entre chaque C4 si R9 est impair.

Lorsqu'un frame débute, l'état **ParitéFrame** permute de pair à impair et vice-versa, **quelque soit la valeur de R8**. Cet état de parité n'est pas modifiable directement.

Si **ParitéFrame** est pair, alors une **ligne additionnelle** et une **MID-VSYNC** sont programmées.
Si **ParitéFrame** est impair, alors **pas de ligne additionnelle** et **pas de MID-VSYNC**.

Au début du Frame, **ParitéC9=ParitéFrame**.

Lorsque R8 passe à 1 ou 3, **ParitéC9=C9.0** (ParitéC9 est fixé avec la parité du C9 courant).
Cette parité de C9 est gérée uniquement lorsque R8=3 pour mettre à jour C9.

ParitéC9 permute à **chaque nouveau C4 lorsque R9 est IMPAIR** (comme sur CRTC 0) (nombre impair de lignes pour un caractère en mode IVM).

Cela signifie qu'il est possible d'avoir des C9 pairs sur un premier frame impair, ou des C9 impairs sur un premier frame pair. **La parité de C9 s'aligne ensuite sur la parité du frame**.

Autrement dit, la parité des C9 est définie **immédiatement** lorsque R8 passe à 3 en opposition à la parité du frame (qui agit alors uniquement au niveau de la **ligne additionnelle** et de la **MID-VSYNC**). Dans ce cas, la parité C9 sera alors identique sur le frame suivant. Contrairement au CRTC 1 et 2, et comme le CRTC 0, C9 ne change cependant pas en cours de ligne.

En mode IVM, R9 est programmé avec un nombre **pair** pour définir un nombre **pair** de lignes dans un caractère (Par exemple R9=6 pour obtenir 2 x 4 ligne/car=8 lignes). Ceci est lié à la méthode utilisée pour tester la fin d'un caractère, composé uniquement de lignes paires ou de lignes impaires selon l'état de **ParitéC9**. Dans ce cas, la parité est identique quelle que soit la valeur de C4.

Lorsque R9 est **impair** en IVM, cela implique une différence entre le nombre de lignes paires et impaires pour un caractère entre 2 frames (le nombre total de lignes d'un caractère vertical est impair). Dans cette circonstance, les concepteurs du circuit ont équilibré le nombre de lignes entre 2 frames afin d'éviter qu'il y ait un frame avec R4+1 lignes de plus sur un frame pair que sur un frame impair. La solution adoptée consiste à alterner des caractères composés de lignes paires avec des caractères composés de lignes impaires sur un même frame **à chaque fois que C4 évolue**. Ainsi, lorsque R9 est **impair**, **la parité des lignes dépend de celle de C4 et de la parité courante de C9**. (La parité de C9 étant rechargée à partir de la parité du frame à chaque nouveau frame).

Pour être totalement satisfaisante, cette méthode implique une gestion spécifique de la VSYNC en mode IVM. En effet, si R7 est programmé sur un C4 impair durant un frame impair, alors la VSYNC est retardée de 1 ligne. Ceci permet d'éviter un déphasage avec la VSYNC générée sur ce même C4 sur un frame pair. Le schéma suivant montre cet équilibrage après 1 frame.

	PARITYFRAME=ODD			PARITYFRAME=EVEN		
	C4	C9		C4	C9	
	0	1		0	0	
	0	3		0	2	
	0	5		0	4	
	0	7		0	6	
	1	0		0	8	
VSYNC	1	2	R7=1	1	1	VSYNC
	1	4		1	3	
	1	6		1	5	
	1	8		1	7	
VSYNC	2	1	R7=2	2	0	VSYNC
	2	3		2	2	
	2	5		2	4	
	2	7		2	6	
	3	0		2	8	
VSYNC	3	2	R7=3	3	1	VSYNC
	3	4		3	3	
	3	6		3	5	
	3	8		3	7	
VSYNC	4	1	R7=4	4	0	VSYNC
	4	3		4	2	

Cependant, selon la parité de C9 sur lequel R8 passe à 3, cela peut entrainer une contradiction entre la parité du frame et la parité de C9 sur le premier frame. Il peut également y avoir un déséquilibre de comptage si R8 passe à 3 sur un C9 impair.

Exemple 1 : R9=7. Si R8 passe à 3 sur C9=1 (impair) sur un frame impair, la ligne C9=0 a déjà été affichée et il y a donc 1 ligne de plus sur C4=0. Le frame étant impair, la VSYNC est retardée de 1 ligne sur les C4 impairs. Sur le frame suivant (pair), la VSYNC ne sera pas retardée à chaque C4 impair. Ceci entraine un déséquilibre de VSYNC **entre les 2 premiers frames** sur toutes les valeurs de R7:

	PARITYFRAME=ODD			PARITYFRAME=EVEN		
	C4	C9		C4	C9	
	0	0		0	0	
R8=3	0	1		0	2	
	0	3		0	4	
	0	5		0	6	
	0	7		0	8	
	1	0	R7=1	1	1	VSYNC
VSYNC	1	2	R7=1	1	3	
	1	4		1	5	
	1	6		1	7	
	1	8	R7=2	2	0	VSYNC
VSYNC	2	1	R7=2	2	2	
	2	3		2	4	
	2	5		2	6	
	2	7		2	8	
	3	0	R7=3	3	1	VSYNC
VSYNC	3	2	R7=3	3	3	
	3	4		3	5	
	3	6		3	7	
	3	8	R7=4	4	0	VSYNC
VSYNC	4	1	R7=4	4	2	
	4	3		4	4	

Remarque : Sur un frame pair, la VSYNC se produit en milieu de ligne sur C0=R0/2.

Exemple 2 : R9=7. Si R8 passe à 3 sur C9=1 (impair) sur un frame pair, la ligne C9=0 a déjà été affichée et il y a donc 1 ligne de plus sur C4=0. Le frame étant pair, la VSYNC n'est pas retardée de 1 ligne sur les C4 impairs. Sur le frame suivant (impair), la VSYNC sera retardée à chaque C4 impair. Ceci entraine un déséquilibre de VSYNC entre **les 2 premiers frames** sur les R7 pairs :

	PARITYFRAME=EVEN			PARITYFRAME=ODD		
	C4	C9		C4	C9	
	0	0		0	1	
R8=3	0	1		0	3	
	0	3		0	5	
	0	5		0	7	
	0	7		1	0	
VSYNC	1	0	R7=1	1	2	VSYNC
	1	2		1	4	
	1	4		1	6	
	1	6		1	8	
	1	8	R7=2	2	1	VSYNC
VSYNC	2	1	R7=2	2	3	
	2	3		2	5	
	2	5		2	7	
	2	7		3	0	
VSYNC	3	0	R7=3	3	2	VSYNC
	3	2		3	4	
	3	4		3	6	
	3	6		3	8	
	3	8	R7=4	4	1	VSYNC
VSYNC	4	1	R7=4	4	3	
	4	3		4	5	

Pour que la transition ne pose aucun problème sur les 2 premiers frames, il faut que R8 soit programmé avec 3 sur la première ligne d'un frame pair. Le problème étant que pour connaitre la parité d'un frame, il faut passer en IVM (R8=3). Il y a donc une chance sur deux que la VSYNC soit déphasée entre les deux premiers frames.

19.6 LIGNE ADDITIONNELLE INTERLACE

19.6.1 CRTC 0

La ligne additionnelle est ajoutée en fin de frame (après les lignes R5 le cas échéant) si un des deux modes « Interlace » est activé (R8=3 ou 1) et l'état **ParitéR6** est **impair**.

Pour rappel, l'état **ParitéR6** devient impair lorsque C4 atteint R6 sur un frame pair, et pair lorsque C4 atteint R6 sur un frame impair, car il sert à anticiper la parité du frame suivant.

Si R6>R4, l'état **ParitéR6** n'est plus mis à jour et cela a pour effet de figer la parité du frame et l'ajout de la ligne additionnelle.

Ce principe de gestion implique que si on est sur un frame pair, et que C4 atteint R6, alors **ParityR6** devient impair. Une ligne additionnelle sera alors générée à la fin de ce frame pair. Le nouveau frame devient impair (égal à **ParityR6**). Mais si, durant ce nouveau frame (impair) C4 ne peut plus atteindre R6 (car R6 a été modifié pour être supérieur à R4) alors une ligne additionnelle sera générée pour chaque frame tant que R6>R4 (et que R8=3 ou 1) et ce quelque soit la parité des C9. L'inverse est vrai : **ParityR6** devient pair, et il n'y a donc pas de ligne additionnelle à la fin du frame. Le frame suivant devient pair, mais si R6 devient supérieur à R4 durant ce frame, alors tous les frames resteront pairs et sans ligne additionnelle.

C4 est incrémenté une seule fois pour toutes les lignes additionnelles (R5 et « Interlace ») et est égal à C4=R4+1.

19.6.2 CRTC 1

La ligne additionnelle est ajoutée en fin de frame (après les lignes R5 le cas échéant) si un des deux modes « Interlace » est activé (R8=3 ou 1) et que **ParitéFrame** est pair.

C4 est incrémenté à chaque fois que C9=R9. Lorsque le mode « Interlace » est actif, si R9+1 est un multiple de R5 alors C4 est incrémenté une fois de plus sur tous les frames pairs. Dis autrement, le CRTC complète les lignes C4/C9 selon le mode de comptage en vigueur.

19.6.3 CRTC 2

La ligne additionnelle est ajoutée en fin de frame (après les lignes R5 le cas échéant) si un des deux modes « Interlace » est activé (R8=3 ou 1) et l'état **ParitéR6** est **impair**.

Pour rappel, l'état **ParitéR6** devient impair lorsque C4 atteint R6 sur un frame pair, et pair lorsque C4 atteint R6 sur un frame impair, car il sert à anticiper la parité du frame suivant.

Si R6>R4, l'état **ParitéR6** n'est plus mis à jour et cela a pour effet de figer la parité du frame et l'ajout de la ligne additionnelle.

Ce principe de gestion implique que si on est sur un frame pair, et que C4 atteint R6, alors **ParityR6** devient impair. Une ligne additionnelle sera alors générée à la fin de ce frame pair. Le nouveau frame devient impair (égal à **ParityR6**). Mais si, durant ce nouveau frame (impair) C4 ne peut plus atteindre R6 (car R6 a été modifié pour être supérieur à R4) alors une ligne additionnelle sera générée pour chaque frame tant que R6>R4 (et que R8=3 ou 1) et ce quelque soit la parité des C9. L'inverse est vrai : **ParityR6** devient pair, et il n'y a donc pas de ligne

additionnelle à la fin du frame. Le frame suivant devient pair, mais si R6 devient supérieur à R4 durant ce frame, alors tous les frames resteront pairs et sans ligne additionnelle.

C4 est incrémenté à chaque fois que C9=R9. Lorsque le mode « Interlace » est actif, si R9+1 est un multiple de R5, alors C4 est incrémenté une fois de plus sur tous les frames pairs. Si R7 est programmé avec cette valeur de C4, il n'y a plus de VSYNC un frame sur 2.

Il existe un **bug notable** sur la gestion de la ligne additionnelle.

Si le mode IVM est activé **sur la première ligne d'un frame impair, alors cette ligne deviendra une ligne additionnelle**, et une nouvelle ligne 0 suivra l'ancienne ligne 0, ce qui allongera la taille du frame de R0 µsec. Ceci est vrai quelque soit la valeur de C0 (0 à R0) sur lequel le mode IVM est activé.

À contrario, si le mode IVM est désactivé durant la ligne additionnelle (C4 étant alors supérieur à R4), alors C4 ne sera pas automatiquement remis à 0 sur la ligne suivante. C9 va compter jusqu'à atteindre R9, et C4 s'incrémentera tant qu'il n'a pas atteint R4.

Exemple : Si R4=38 (R5=0) et qu'une ligne additionnelle est générée car IVM est actif à la fin du frame pair (R8=3 lorsque C4=R4, C9=R9, C0=R0) alors une ligne additionnelle IVM est ajoutée sur C4=39, C9=0. Si IVM est désactivé sur cette ligne (R8=0), alors C9 va compter jusqu'à R9, puis C4 sera de nouveau incrémenté pour passer à 40 (car R4=38).

Si la parité était impaire lorsque le mode IVM est activé sur la ligne 0, cette ligne, désormais considérée comme une ligne additionnelle, devient immédiatement impaire. Ainsi les C9 affichés dès que R8=3 sur cette ligne seront impairs (soit C9=1). La ligne additionnelle ne peut pas être déclenchée avec une parité paire figée, car **ParitéR6** est faux sur un frame impair lorsque C4=R6. Cette dernière condition est nécessaire pour obtenir une parité paire.

19.6.4 CRTC 3 & 4

La ligne additionnelle est ajoutée en fin de frame (après les lignes R5 le cas échéant) si un des deux modes « Interlace » est activé (R8=3 ou 1) et que **ParitéFrame** est vrai.

L'ajout de la ligne additionnelle ne dépend pas de l'équivalence C4=R6, contrairement aux CRTC 0 et 2.

Lorsque la ligne additionnelle est ajoutée, C4 n'est pas incrémenté (contrairement à tous les autres CRTC). La dernière valeur de C4 porte toutes les lignes programmées dans R9, plus celles de R5 plus la ligne additionnelle « Interlace » générée durant les frames pairs.

Remarque : La ligne additionnelle générée ne tient pas compte des états de parité comme sur les autres CRTC. C9 vaudra toujours 0, même si les autres lignes sont impaires sur C4=R4.

Remarque : La mise à jour du pointeur vidéo est prise en compte sur C9=R9 de C4=R4 sans que C4 soit incrémenté. La première ligne additionnelle débute avec le pointeur que valait VMA au moment ou C9=R9 et C0=R1.

19.7 MID-VSYNC

19.7.1 GÉNÉRALITÉS

On parle de **MID-VSYNC** lorsque la **VSYNC** se produit au milieu d'une ligne.

En général, la **VSYNC** se produit lorsque C4 est égal à R7 sur n'importe quelle position de C0 (sauf sur les CRTC 3 et 4, qui imposent que C4=C9=C0=0).

Il existe également une exception sur les CRTC 0, 3 et 4 lorsque le nombre de lignes d'un caractère C4 est impair sur un frame impair et un C4 impair. La **VSYNC** peut alors être retardée d'une ligne. La **MID-VSYNC** n'est pas cumulative avec cette ligne car elle ne peut pas se produire sur un frame impair avec un C4 impair.

Lorsque l'état de **MID-VSYNC** est actif, alors la **VSYNC** se déclenche uniquement lorsque C0 atteint R0/2.

C'est ce qui permet en principe de créer des demi-lignes, en conjonction avec la ligne additionnelle qui permet au moniteur de compenser l'écart entre 2 frame en « Interlace » (chaque frame occupant alors 32 µs de plus (avec R0=63)).

19.7.2 CRTC 0, 1, 2

La **MID-VSYNC** est générée lorsque C4=R7 si **ParitéFrame** est pair (et que R8 vaut 3 ou 1).

Dans le cas où R7=0, la gestion de la **VSYNC** coïncide avec **le début du frame**.
La gestion de **ParitéFrame est prioritaire sur la gestion de la VSYNC**.

Ainsi, si R7=0, et que **ParitéFrame** permute et devient pair lorsque C4=C9=C0=0, la comparaison C4/R7 sera traitée ensuite pour déclencher une **VSYNC**.

Si R8=3 ou R8=1, et que **ParitéFrame** est devenu pair, alors la **VSYNC** se produira lorsque C0 atteindra R0/2.

La **MID-VSYNC** a donc toujours lieu lorsque **ParitéFrame** est pair, y compris lorsque R7=0.

19.7.3 CRTC 3, 4

La **MID-VSYNC** est générée lorsque C4=R7 si **ParitéFrame** est pair (et que R8 vaut 3 ou 1).

Dans le cas particulier ou R7=0, la gestion de la **VSYNC** coïncide avec **le début du frame**.
Dans ce cas, **la gestion de la VSYNC est prioritaire sur l'affectation de ParitéFrame**

Ainsi, si R7=0 (et R9=3 ou 1), la comparaison C4/R7 est traitée avant que **ParitéFrame** permute.

Si **ParitéFrame** était impair, alors il n'y aura pas de **MID-VSYNC**, et la **VSYNC** débutera sur C4=C9=C0=0, bien que **ParitéFrame** soit devenu Pair.
Inversement, si **ParitéFrame** était pair, alors il y aura une **MID-VSYNC,** et la **VSYNC** débutera lorsque C0 atteindra R0/2 et bien que **ParitéFrame** soit devenu impair.

La **MID-VSYNC** a donc toujours lieu lorsque **ParitéFrame** est pair, sauf lorsque R7=0.
Et elle n'a jamais lieu lorsque **ParitéFrame** est impair, sauf lorsque R7=0.

19.8 COMPTAGES EN INTERLACE VIDEOMODE

19.8.1 CRTC 0

Lorsque le mode **IVM** est activé, le compteur C9 continue à s'incrémenter normalement.

Cependant, lorsque C9 est utilisé dans la constitution de l'adresse VMA, C9 est considéré comme décalé à gauche de 1 bit, et le bit 0 représente la parité. Chaque incrémentation « correspond » donc à un ajout de 2 au compteur C9-VMA tenant compte de **ParitéC9**.

La gestion d'incrémentation de C9 continue d'être traitée sur la valeur normale de C9.

À partir de la **ligne C9 qui suit celle ou R8 passe à 3**, la valeur de C9 est multipliée par 2 pour être testée avec R9 afin de permettre à C9 de repasser à 0. La capacité de R9 étant de 5 bits, le bit de poids fort est « perdu » dans l'opération de multiplication.

On peut formuler ça ainsi sur les lignes situées après celle ou R8 est passé à 3 :

```
Si ((C9 x 2) or ParitéFrame)==(R9+ParitéFrame)
Alors
        Si C4==R4 (ou fin de frame)
        Alors
                C4=0
                Si ParitéR6==vrai (si C4==R6 vrai sur un frame pair)
                        ParitéFrame=ParitéFrame xor 1
                Fin Si
        Sinon
                C4++
        Fin Si
        Si R9.0==1 (gestion de la parité C9 en fonction de C4 si R9 est impair)
        Alors
                ParitéC9=C4.0 xor ParitéFrame
        Fin Si
        C9=0 ;
        C9.VMA=(C9 x 2) or ParitéC9
Sinon
        C9=C9+1
        C9.VMA=(C9 x 2) or ParitéC9
Fin Si
```

Lorsque R8 passe à 3, un état indique que le calcul de la valeur comparée à R9 or ParitéFrame sera réalisé sur le prochain C0=0, après le test C9/R9 de la ligne. Cela est très certainement fait ainsi pour éviter que le C9 utilisé pour l'affichage bascule en cours de ligne, comme c'est le cas sur les CRTC 1 et 2. La valeur de test pour R9 est donc l'ancien C9, mais la parité est cependant **prise en compte immédiatement pour R9**.

Ainsi, sur la ligne ou R8 passe à 3, c'est C9 (et non C9.VMA) qui est comparé à R9 or ParitéFrame. **Si C9=R9 et que la parité est impaire, alors le test C9=R9 or 1 est faux : C9 n'est pas remis à 0 et déborde**. Autrement dit C9=R9+1.

La valeur de C9 lorsque R8 passe à 3 peut conduire à un débordement de C9.

Par exemple, si R9=6 et C9=4 lorsque R8 passe à 3, C9 est différent de 'R9 or ParitéFrame' (6 ou 7) et le C9 suivant sera donc C9+1=5. Le C9.VMA affiché (et testé jusqu'à la fin de la ligne) sera '(C9x2)+ParitéFrame', soit 10 ou 11 selon la parité courante. C9 va alors continuer à s'incrémenter jusqu'à ce que 'C9x2+ParitéFrame' soit égal à 'R9 or paritéC9'.

Lorsque R8 revient à 0, la même mécanique s'applique. L'état positionné avec R8 est pris en compte lorsque C0 repasse à 0. Le test a lieu avec C9.VMA (qui intègre la parité) et R9 : La parité n'est plus prise en compte pour R9. Le test de limite est donc réalisé avec 'C9x2+ParitéFrame' (C9.VMA) et R9 (sans parité).

Remarque : Le test C9/R9 ou C9.VMA/'R9 or ParitéC9' a lieu aussi lorsque C0=R1 (qui détermine l'activation du BORDER) et l'affectation de VMA' avec VMA lorsque C9=R9 (ou C9.VMA=R9 or ParitéC9). La prise en compte de la parité dans R9 dès que R8=3 (et sa non prise en compte dès que R8=0) dans le test avec C9/C9.VMA peut donc aussi affecter la mise à jour de VMA'.

Si C9 valait 3 et R9=6 sur un frame impair, alors C9.VMA=7 en début de ligne. Lorsque R8 passe à 0, C9.VMA est comparé à R9 (la prise en compte de parité est annulée). C9.VMA=7 <> R9=6, ce qui conduit C9 à être incrémenté (et donc passer à 4). Si on avait souhaité que C9 repasse à 0, il faudrait par exemple programmer R9 avec C9.VMA avant la fin de la ligne (soit 7) afin que le comparaison entre C9.VMA et R9 sans parité permette à C9 de revenir à 0.

Les schémas pages suivantes décrivent différentes situations de comptage, lors du passage en mode IVM ou lors de la sortie du mode IVM. Ils indiquent la valeur de C9 considérée et celle disponible pour la constitution du pointeur vidéo.

La période IVM pour les tests suivants couvre seulement quelques lignes du frame.
Valeur des registres pour les résultats indiqués :
R9=6 (pair) et R8=0 avant de passer à 3 (ou repasser à 0).

Passage en mode IVM sur CRTC 0 :

EVEN FRAME

C4	C9	C9-VMA	R8 UPDATE
0	0	0	R8=3
0	1	2	
0	2	4	
0	3	**6**	
1	0	0	
1	1	2	
1	2	4	
1	3	**6**	

C4	C9	C9-VMA	R8 UPDATE
0	0	0	
0	1	1	R8=3
0	2	4	
0	3	**6**	
1	0	0	
1	1	2	
1	2	4	
1	3	**6**	

C4	C9	C9-VMA	R8 UPDATE
0	0	0	
0	1	1	
0	2	2	R8=3
0	3	**6**	
1	0	0	
1	1	2	
1	2	4	
1	3	**6**	

C4	C9	C9-VMA	R8 UPDATE
0	0	0	
0	1	1	
0	2	2	
0	3	3	R8=3
0	4	8	
0	5	10	
0	6	12	
0	7	14	
0	8	16	
0	9	18	
0	10	20	
0	11	22	
0	12	24	
0	13	26	
0	14	28	
0	15	30	
0	16	0	
0	17	2	
0	18	4	
0	19	**6**	
1	0	0	
1	1	2	
1	2	4	
1	3	**6**	

ODD FRAME

C4	C9	C9-VMA	R8 UPDATE
0	0	0	R8=3
0	1	3	
0	2	5	
0	3	**7**	
1	0	1	
1	1	3	
1	2	5	
1	3	**7**	

C4	C9	C9-VMA	R8 UPDATE
0	0	0	
0	1	1	R8=3
0	2	5	
0	3	**7**	
1	0	1	
1	1	3	
1	2	5	
1	3	**7**	

C4	C9	C9-VMA	R8 UPDATE
0	0	0	
0	1	1	
0	2	2	R8=3
0	3	**7**	
1	0	1	
1	1	3	
1	2	5	
1	3	**7**	

C4	C9	C9-VMA	R8 UPDATE
0	0	0	
0	1	1	
0	2	2	
0	3	3	R8=3
0	4	9	
0	5	11	
0	6	13	
0	7	15	
0	8	17	
0	9	19	
0	10	21	
0	11	23	
0	12	25	
0	13	27	
0	14	29	
0	15	31	
0	16	1	
0	17	3	
0	18	5	
0	19	**7**	
1	0	1	
1	1	3	
1	2	5	
1	3	**7**	

EVEN FRAME

C4	C9	C9-VMA	R8 UPDATE
0	0	0	
0	1	1	
0	2	2	
0	3	3	
0	4	4	R8=3
0	5	10	
0	6	12	
0	7	14	
0	8	16	
0	9	18	
0	10	20	
0	11	22	
0	12	24	
0	13	26	
0	14	28	
0	15	30	
0	16	0	
0	17	2	
0	18	4	
0	19	**6**	
1	0	0	
1	2	2	
1	4	4	
1	6	**6**	

ODD FRAME

C4	C9	C9-VMA	R8 UPDATE
0	0	0	
0	1	1	
0	2	2	
0	3	3	
0	4	4	R8=3
0	5	11	
0	6	13	
0	7	15	
0	8	17	
0	9	19	
0	10	21	
0	11	23	
0	12	25	
0	13	27	
0	14	29	
0	15	31	
0	16	1	
0	17	3	
0	18	5	
0	19	**7**	
1	0	1	
1	2	3	
1	4	5	
1	6	**7**	

C4	C9	C9-VMA	R8 UPDATE
0	0	0	
0	1	1	
0	2	2	
0	3	3	
0	4	4	
0	5	5	R8=3
0	6	12	
0	7	14	
0	8	16	
0	9	18	
0	10	20	
0	11	22	
0	12	24	
0	13	26	
0	14	28	
0	15	30	
0	16	0	
0	17	2	
0	18	4	
0	19	**6**	
1	0	0	
1	2	2	
1	4	4	
1	6	**6**	

C4	C9	C9-VMA	R8 UPDATE
0	0	0	
0	1	1	
0	2	2	
0	3	3	
0	4	4	
0	5	5	R8=3
0	6	13	
0	7	15	
0	8	17	
0	9	19	
0	10	21	
0	11	23	
0	12	25	
0	13	27	
0	14	29	
0	15	31	
0	16	1	
0	17	3	
0	18	5	
0	19	**7**	
1	0	1	
1	2	3	
1	4	5	
1	6	**7**	

EVEN FRAME

C4	C9	C9-VMA	R8 UPDATE
0	0	0	
0	1	1	
0	2	2	
0	3	3	
0	4	4	
0	5	5	
0	6	**6**	**R8=3**
1	0	0	
1	2	2	
1	4	4	
1	6	**6**	
2	0	0	
2	2	2	
2	4	4	
2	6	**6**	
3	0	0	
3	2	2	
3	4	4	
3	6	**6**	
4	0	0	
4	2	2	
4	4	4	
4	6	**6**	
5	0	0	

ODD FRAME

C4	C9	C9-VMA	R8 UPDATE
0	0	0	
0	1	1	
0	2	2	
0	3	3	
0	4	4	
0	5	5	
0	6	6	**R8=3**
0	7	15	
0	8	17	
0	9	19	
0	10	21	
0	11	23	
0	12	25	
0	13	27	
0	14	29	
0	15	31	
0	16	1	
0	17	3	
0	18	5	
0	19	**7**	
1	0	1	
1	1	3	
1	2	5	
1	3	**7**	

Sortie du mode IVM sur CRTC 0 :

EXIT IVM MODE ON EVEN FRAME

C4	C9	C9-VMA	R8 UPDATE
0	0	0	**R8=3**
0	1	2	
0	2	4	
0	3	6	
1	0	0	**R8=0**
1	1	1	
1	2	2	
1	3	3	
1	4	4	
1	5	5	
1	6	6	
1	7	7	

C4	C9	C9-VMA	R8 UPDATE
0	0	0	**R8=3**
0	1	2	
0	2	4	
0	3	6	
1	0	0	
1	1	2	**R8=0**
1	2	2	
1	3	3	
1	4	4	
1	5	5	
1	6	6	
1	7	7	

C4	C9	C9-VMA	R8 UPDATE
0	0	0	**R8=3**
0	1	2	
0	2	4	
0	3	6	
1	0	0	
1	1	2	
1	2	4	**R8=0**
1	3	3	
1	4	4	
1	5	5	
1	6	6	
2	7	7	

C4	C9	C9-VMA	R8 UPDATE
0	0	0	**R8=3**
0	1	2	
0	2	4	
0	3	6	
1	0	0	
1	1	2	
1	2	4	
1	3	6	**R8=0**
2	0	0	
2	1	1	
2	2	2	
2	3	3	

EXIT IVM MODE ON ODD FRAME

C4	C9	C9-VMA	R8 UPDATE
0	0	1	**R8=3**
0	1	3	
0	2	5	
0	3	7	
1	0	1	**R8=0**
1	1	1	
1	2	2	
1	3	3	
1	4	4	
1	5	5	
1	6	6	
1	7	7	

C4	C9	C9-VMA	R8 UPDATE
0	0	1	**R8=3**
0	1	3	
0	2	5	
0	3	7	
1	0	1	
1	1	3	**R8=0**
1	2	2	
1	3	3	
1	4	4	
1	5	5	
1	6	6	
1	7	7	

C4	C9	C9-VMA	R8 UPDATE
0	0	1	**R8=3**
0	1	3	
0	2	5	
0	3	7	
1	0	1	
1	1	3	
1	2	5	**R8=0**
1	3	3	
1	4	4	
1	5	5	
1	6	6	
1	7	7	

C4	C9	C9-VMA	R8 UPDATE
0	0	1	**R8=3**
0	1	3	
0	2	5	
0	3	7	
1	0	1	
1	1	3	
1	2	5	
1	3	7	**R8=0**
1	4	4	
1	5	5	
1	6	6	
1	7	7	

19.8.2 CRTC 1

Le mode de calcul de C9 est orienté sur deux principes. D'une part une équivalence en nombre de ligne entre les frames pairs et impairs, et d'autre part une conservation de la parité des lignes.
La parité est fixée lorsque R8=3 selon la parité de R9 et de C9. Si R9 est pair (nombre de lignes impair d'un caractère), alors la parité de C4 est également prise en compte.

Lorsque le mode IVM est sélectionné, le comptage s'effectue en 2 temps et dépend de la parité de R9 pour permettre de traiter tous les cas de figure.

Lorsque C0 passe à 0 :

- **C9=C9+not (R9.0) (C9 est incrémenté si R9 est pair)**
- **Si (C9 and %11110) == (R9 and %11110) (test C9/R9 Hors Parité)**
  Alors
  Gestion de C4 (C4++ ou C4=0 si C4==R4)
  ParitéC9 = ParitéC9 xor (not R9.0) (inversion ParitéC9 si R9 est pair)
  C9=ParitéC9
  Sinon
  C9=C9+1+(R9.0)
  Fin Si

Dès que R8 repasse à 0, la logique de comptage reprend normalement.

```
Si C9==R9
Alors
    C9=0
    Gestion de C4 (C4++ ou C4=0 si C4==R4)
Sinon
    C9=C9+1
Fin Si
```

19.8.3 CRTC 2

Il n'est pas nécessaire de reprogrammer R4 lorsque R8 passe à 3. C'est également le cas lorsque le mode IVM est programmé sur une partie du frame. C4 n'est pas faussé lorsque R8 est mis à jour, même temporairement. R5, R6 et R7 n'ont également pas besoin d'être reprogrammé pour que le frame soit synchronisé avec l'écran.

En mode « Interlace », C9 est comparé avec R9 de manière classique pour traiter C4.
Un autre compteur, **C9.IVM**, est utilisé pour l'affichage et gérer la mise à jour du pointeur vidéo.

Lorsque R9 est impair, l'adresse vidéo est modifiée 2 fois pour chaque valeur de C4.

On peut formuler ça ainsi sur les lignes situées après celle ou R8 est passé à 3 :

```
Si (C9==R9)
Alors
        C9.IVM=ParitéFrame
        C9=0 ; Gestion C4 (C4++ ou C4=0)
Sinon
        C9.IVM=C9.IVM+2
        C9=C9+1
Fin Si

Si (C9.IVM and &1e==R9 and &1e)
Alors
        Si (C0==R1)
        Alors
                VMA'=VMA
        Fin si
        C9.IVM=ParitéFrame
Fin Si
```

La gestion du comptage **C9.IVM** a lieu tout le temps, quelque soit la valeur de R8.
L'activation du mode IVM en cours de ligne utilise immédiatement **C9.IVM pour l'affichage.**

La gestion spécifique d'affectation de VMA' avec VMA lorsque C0==R1 n'est traitée que lorsque R8 vaut 3. Lorsque R8 vaut 0 ou 2, cette affectation a lieu uniquement lorsque C9==R9. Autrement dit, C4 peut s'incrémenter sans que **VMA'** soit mis à jour en mode IVM.

C9.IVM est réinitialisé (0 ou 1 selon la parité frame) deux fois durant un caractère C4, car il compte 2 fois plus vite que C9. Une fois lorsque **C9.IVM** atteint R9 (hors parité) et une fois lorsque C9 atteint R9. Si R9 est pair, C9 atteint R9 avant que **C9.IVM** atteigne R9 (hors parité) et le pointeur vidéo VMA' n'est pas transféré dans VMA.

Si, par exemple, C4=1 et R9 vaut 6 lorsque R8=3, C9 va compter de 0 à 6 quelque soit la parité du frame. Sur les frames pairs, **C9.IVM** vaudra 0, 2, 4, 6, 0, 2, 4 et sur les frames impairs, **C9.IVM** vaudra 1, 3, 5, 7, 1, 3, 5. Cependant, "**C9.IVM and &1E**" ne sera égal à "**R9 and &1E**" (6) qu'une seule fois (lorsque **C9.IVM**=6 ou 7) et C4 va passer à 2 sans que le pointeur vidéo soit mis à jour.

Il faut également rappeler que si le mode IVM est activé sur la première ligne d'un frame (Lorsque C4=C9=0) alors que la parité était impaire, alors le **C9** et **C9.IVM** sont réinitialisés sur la 2ème ligne (**C9.IVM=ParitéFrame** et **C9=0**).

La valeur limite de **C9.IVM** est toujours traitée « hors parité », en excluant le bit 0 de R9 et de **C9.IVM** pour le test de comparaison.

Si R8 passe de 0 à 3 lorsque C9=5 et R9=11, alors **C9.IVM**=10 sur un frame pair et **C9.IVM**=11 sur un frame impair.

Si R8 passe de 3 à 0 lorsque **C9.IVM**=8 et R9=11, alors C9 passe à 4 si on était sur la 5ème ligne ou 10 si on était sur la 11ème ligne.

Cette translation d'usage de C9 ou C9.IVM pour l'affichage est immédiatement prise en compte dans la constitution de l'adresse affichée, et ce y compris durant la ligne, à partir de la position C0 ou R8 est modifié.

Le BORDER est activé lorsque C4=R6. Si on considère qu'une valeur de C4 permet de définir 2 caractères affichés, alors le BORDER fonctionne par paire.

Les schémas pages suivantes décrivent différentes situations de comptage, lors du passage en mode IVM ou lors de la sortie du mode IVM.

R9=7 et R8=0 avant de passer à 3 (ou repasser à 0).

Passage en mode IVM sur CRTC 2 :

EVEN FRAME

C4	C9	C9-IVM	R8 UPDATE
0	0	0	R8=3
0	1	2	
0	2	4	
0	3	6	
0	4	0	
0	5	2	
0	6	4	
0	7	6	
1	0	0	
1	1	2	
1	2	4	
1	3	6	
1	4	0	
1	5	2	
1	6	4	
1	7	6	

ODD FRAME

C4	C9	C9-IVM	R8 UPDATE
0	0	0	R8=3
0	1	3	
0	2	5	
0	3	7	
0	4	1	
0	5	3	
0	6	5	
0	7	7	
1	0	1	
1	1	3	
1	2	5	
1	3	7	
1	4	1	
1	5	3	
1	6	5	
1	7	7	

C4	C9	C9-IVM	R8 UPDATE
0	0	0	
0	1	1	R8=3
0	2	4	
0	3	6	
0	4	0	
0	5	2	
0	6	4	
0	7	6	
1	0	0	
1	1	2	
1	2	4	
1	3	6	
1	4	0	
1	5	2	
1	6	4	
1	7	6	

C4	C9	C9-IVM	R8 UPDATE
0	0	0	
0	1	1	R8=3
0	2	5	
0	3	7	
0	4	1	
0	5	3	
0	6	5	
0	7	7	
1	0	1	
1	1	3	
1	2	5	
1	3	7	
1	4	1	
1	5	3	
1	6	5	
1	7	7	

C4	C9	C9-IVM	R8 UPDATE
0	0	0	
0	1	1	
0	2	2	R8=3
0	3	6	
0	4	0	
0	5	2	
0	6	4	
0	7	6	
1	0	0	
1	1	2	
1	2	4	
1	3	6	
1	4	0	
1	5	2	
1	6	4	
1	7	6	

C4	C9	C9-VMA	R8 UPDATE
0	0	0	
0	1	1	
0	2	2	R8=3
0	3	7	
0	4	1	
0	5	3	
0	6	5	
0	7	7	
1	0	1	
1	1	3	
1	2	5	
1	3	7	
1	4	1	
1	5	3	
1	6	5	
1	7	7	

EVEN FRAME

C4	C9	C9-IVM	R8 UPDATE
0	0	0	
0	1	1	
0	2	2	
0	3	3	R8=3
0	4	0	
0	5	2	
0	6	4	
0	7	6	
1	0	0	
1	1	2	
1	2	4	
1	3	6	
1	4	0	
1	5	2	
1	6	4	
1	7	6	

ODD FRAME

C4	C9	C9-VMA	R8 UPDATE
0	0	0	
0	1	1	
0	2	2	
0	3	3	R8=3
0	4	1	
0	5	3	
0	6	5	
0	7	7	
1	0	1	
1	1	3	
1	2	5	
1	3	7	
1	4	1	
1	5	3	
1	6	5	
1	7	7	

C4	C9	C9-IVM	R8 UPDATE
0	0	0	
0	1	1	
0	2	2	
0	3	3	
0	4	4	R8=3
0	5	2	
0	6	4	
0	7	6	
1	0	0	
1	1	2	
1	2	4	
1	3	6	
1	4	0	
1	5	2	
1	6	4	
1	7	6	

C4	C9	C9-VMA	R8 UPDATE
0	0	0	
0	1	1	
0	2	2	
0	3	3	
0	4	4	R8=3
0	5	3	
0	6	5	
0	7	7	
1	0	1	
1	1	3	
1	2	5	
1	3	7	
1	4	1	
1	5	3	
1	6	5	
1	7	7	

C4	C9	C9-IVM	R8 UPDATE
0	0	0	
0	1	1	
0	2	2	
0	3	3	
0	4	4	
0	5	5	R8=3
0	6	4	
0	7	6	
1	0	0	
1	1	2	
1	2	4	
1	3	6	
1	4	0	
1	5	2	
1	6	4	
1	7	6	

C4	C9	C9-VMA	R8 UPDATE
0	0	0	
0	1	1	
0	2	2	
0	3	3	
0	4	4	
0	5	5	R8=3
0	6	5	
0	7	7	
1	0	1	
1	1	3	
1	2	5	
1	3	7	
1	4	1	
1	5	3	
1	6	5	
1	7	7	

EVEN FRAME

C4	C9	C9-IVM	R8 UPDATE
0	0	0	
0	1	1	
0	2	2	
0	3	3	
0	4	4	
0	5	5	
0	6	6	R8=3
0	7	6	
1	0	0	
1	1	2	
1	2	4	
1	3	6	
1	4	0	
1	5	2	
1	6	4	
1	7	6	

ODD FRAME

C4	C9	C9-VMA	R8 UPDATE
0	0	0	
0	1	1	
0	2	2	
0	3	3	
0	4	4	
0	5	5	
0	6	6	R8=3
0	7	7	
1	0	1	
1	1	3	
1	2	5	
1	3	7	
1	4	1	
1	5	3	
1	6	5	
1	7	7	

C4	C9	C9-IVM	R8 UPDATE
0	0	0	
0	1	1	
0	2	2	
0	3	3	
0	4	4	
0	5	5	
0	6	6	
0	7	7	R8=3
1	0	0	
1	1	2	
1	2	4	
1	3	6	
1	4	0	
1	5	2	
1	6	4	
1	7	6	

C4	C9	C9-VMA	R8 UPDATE
0	0	0	
0	1	1	
0	2	2	
0	3	3	
0	4	4	
0	5	5	
0	6	6	
0	7	7	R8=3
1	0	1	
1	1	3	
1	2	5	
1	3	7	
1	4	1	
1	5	3	
1	6	5	
1	7	7	

Sortie du mode IVM sur CRTC 2 :

EXIT IVM MODE

EVEN FRAME

C4	C9	C9-VMA	R8 UPDATE
0	0	0	R8=3
0	1	2	
0	2	4	
0	3	6	
0	4	0	
0	5	2	
0	6	4	
0	7	6	
1	0	0	R8=0
1	1	1	
1	2	2	
1	3	3	
1	4	4	
1	5	5	
1	6	6	
1	7	7	

ODD FRAME

C4	C9	C9-VMA	R8 UPDATE
0	0	0	R8=3
0	1	3	
0	2	5	
0	3	7	
0	4	1	
0	5	3	
0	6	5	
0	7	7	
1	0	1	R8=0
1	1	1	
1	2	2	
1	3	3	
1	4	4	
1	5	5	
1	6	6	
1	7	7	

C4	C9	C9-VMA	R8 UPDATE
0	0	0	R8=3
0	1	2	
0	2	4	
0	3	6	
0	4	8	
0	5	0	
0	6	2	
0	7	4	
1	0	0	
1	1	2	R8=0
1	2	2	
1	3	3	
1	4	4	
1	5	5	
1	6	6	
1	7	7	

C4	C9	C9-VMA	R8 UPDATE
0	0	0	R8=3
0	1	3	
0	2	5	
0	3	7	
0	4	1	
0	5	3	
0	6	5	
0	7	7	
1	0	1	
1	1	3	R8=0
1	2	2	
1	3	3	
1	4	4	
1	5	5	
1	6	6	
1	7	7	

C4	C9	C9-VMA	R8 UPDATE
0	0	0	R8=3
0	1	2	
0	2	4	
0	3	6	
0	4	0	
0	5	2	
0	6	4	
0	7	6	
1	0	0	
1	1	2	
1	2	4	R8=0
1	3	3	
1	4	4	
1	5	5	
1	6	6	
1	7	7	

C4	C9	C9-VMA	R8 UPDATE
0	0	0	R8=3
0	1	3	
0	2	5	
0	3	7	
0	4	1	
0	5	3	
0	6	5	
0	7	7	
1	0	1	
1	1	3	
1	2	5	R8=0
1	3	3	
1	4	4	
1	5	5	
1	6	6	
1	7	7	

EXIT IVM MODE

EVEN FRAME

C4	C9	C9-VMA	R8 UPDATE
0	0	0	R8=3
0	1	2	
0	2	4	
0	3	6	
0	4	0	
0	5	2	
0	6	4	
0	7	6	
1	0	0	
1	1	2	
1	2	4	
1	3	6	R8=0
1	4	4	
1	5	5	
1	6	6	
1	7	7	

ODD FRAME

C4	C9	C9-VMA	R8 UPDATE
0	0	0	R8=3
0	1	3	
0	2	5	
0	3	7	
0	4	1	
0	5	3	
0	6	5	
0	7	7	
1	0	1	
1	1	3	
1	2	5	
1	3	7	R8=0
1	4	4	
1	5	5	
1	6	6	
1	7	7	

C4	C9	C9-VMA	R8 UPDATE
0	0	0	R8=3
0	1	2	
0	2	4	
0	3	6	
0	4	0	
0	5	2	
0	6	4	
0	7	6	
1	0	0	
1	1	2	
1	2	4	
1	3	6	
1	4	0	R8=0
1	5	5	
1	6	6	
1	7	7	

C4	C9	C9-VMA	R8 UPDATE
0	0	0	R8=3
0	1	3	
0	2	5	
0	3	7	
0	4	1	
0	5	3	
0	6	5	
0	7	7	
1	0	1	
1	1	3	
1	2	5	
1	3	7	
1	4	1	R8=0
1	5	5	
1	6	6	
1	7	7	

C4	C9	C9-VMA	R8 UPDATE
0	0	0	R8=3
0	1	2	
0	2	4	
0	3	6	
0	4	0	
0	5	2	
0	6	4	
0	7	6	
1	0	0	
1	1	2	
1	2	4	
1	3	6	
1	4	0	
1	5	2	R8=0
1	6	6	
1	7	7	

C4	C9	C9-VMA	R8 UPDATE
0	0	0	R8=3
0	1	3	
0	2	5	
0	3	7	
0	4	1	
0	5	3	
0	6	5	
0	7	7	
1	0	1	
1	1	3	
1	2	5	
1	3	7	
1	4	1	
1	5	3	R8=0
1	6	6	
1	7	7	

EXIT IVM MODE

EVEN FRAME

C4	C9	C9-VMA	R8 UPDATE
0	0	0	R8=3
0	1	2	
0	2	4	
0	3	6	
0	4	0	
0	5	2	
0	6	4	
0	7	6	
1	0	0	
1	1	2	
1	2	4	
1	3	6	
1	4	0	
1	5	2	
1	6	4	R8=0
1	7	7	

ODD FRAME

C4	C9	C9-VMA	R8 UPDATE
0	0	0	R8=3
0	1	3	
0	2	5	
0	3	7	
0	4	1	
0	5	3	
0	6	5	
0	7	7	
1	0	1	
1	1	3	
1	2	5	
1	3	7	
1	4	1	
1	5	3	
1	6	5	R8=0
1	7	7	

C4	C9	C9-VMA	R8 UPDATE
0	0	0	R8=3
0	1	2	
0	2	4	
0	3	6	
0	4	0	
0	5	2	
0	6	4	
0	7	6	
1	0	0	
1	1	2	
1	2	4	
1	3	6	
1	4	0	
1	5	2	
1	6	4	
1	7	6	R8=0
2	0	0	
2	1	1	

C4	C9	C9-VMA	R8 UPDATE
0	0	0	R8=3
0	1	3	
0	2	5	
0	3	7	
0	4	1	
0	5	3	
0	6	5	
0	7	7	
1	0	1	
1	1	3	
1	2	5	
1	3	7	
1	4	1	
1	5	3	
1	6	5	
1	7	7	R8=0
2	0	0	
2	1	1	

19.8.4 CRTC 3, 4

Il n'existe aucune documentation sur l'implémentation du mode « Interlace » sur ces CRTC émulés par AMSTRAD, mais la logique de comptage est cependant assez simple.

Rappelons que lorsque R9 est mis à jour avec une valeur inférieure à C9, alors C9 passe à 0. Comme sur les CRTC 0 et 1, l'activation du mode IVM modifie le comptage de C4, et nécessite donc d'adapter les valeurs de R4, R6 et R7 selon les évolutions de C4.

Pour que le mode IVM suive une logique de lignes paires/impaires, il est nécessaire de programmer R9 **de la même façon que sur un CRTC 0**. Il faut donc que R9 contienne le nombre de lignes d'un caractère moins 2. Soit la valeur 6 si un caractère est composé de 8 lignes. Si R9 est impair, le parité de C9 change à chaque fois que C4 évolue.

L'algorithme suivant décrit la gestion de C9 et C4 :

```
Si C9 >= R9
Alors
    Si C4==R4
    Alors
        C4=0
        ParitéFrame=ParitéFrame xor 1
        ParitéC9=ParitéFrame
    Sinon
        C4++
        Si R9.0==1 (R9 est impair)
        Alors
            ParitéC9=ParitéC9 xor 1
        Fin Si
    Fin Si
    C9=ParitéC9
Sinon
    Si R8<3
    Alors
        C9=C9+1
    Sinon
        C9=C9+2
        C9=C9 or ParitéC9
    Fin Si
Fin Si
```

Exemple 1 :

Sur la ligne N, R9=7, C9=6, C4=0, R4=10, ParitéC9=paire(0), ParitéFrame=paire(0)

Ligne N+1 C9<R9 (6<7)
C9=C9+2 (C9=8)

Ligne N+2 C9>=R9 (8>7)
C4=C4+1 (C4=1)
ParitéC9=1 car R9 est impair (R9=7)
C9=ParitéC9 (C9=1)

Dans cet exemple, la parité frame et la parité C9 sont paires (car ParitéC9=ParitéFrame au début du frame). Les lignes C9 affichées en début de frame sont alignées sur la parité du Frame et pour C4=0, on obtient les lignes C9=0, 2, 4, 6, 8. Lorsque C4 est incrémenté (pour passer à 1), la parité de C9 s'inverse (car R9 est impair, ceci afin d'équilibrer les lignes entre 2 frame, et obtenir des C4 composés de 5 lignes paires/4 lignes impaires et des C4 de 4 lignes paires/5 lignes impaires). Sur le caractère C4=1, on aura donc les C9=1, 3, 5, 7.

Exemple 2 :

Sur la ligne N, R9=7, C9=7, C4=9, R4=9, ParitéC9=impaire(1), ParitéFrame=paire(0)

Ligne N+1 C9>=R9 (7>=7)
C4=0 car C4==R4
ParitéFrame=1
ParitéC9=ParitéFrame=1
C9=PariteC9=1

Ligne N+2 C9<R9 (1<7)
C9=C9+2 (C9=3)

Dans cet exemple, on est sur le dernier C4 du frame. La parité du frame est paire et R9 est impair. Durant ce frame, les C9 étaient pairs sur les C4 pairs et les C9 étaient impairs sur les C4 impairs. R4 étant impair (nombre de C4 pairs, C4=0 à 9) on a des C9 pairs sur C4=0 et des C9 impairs sur C4=9. Mais sur le nouveau frame, la **ParitéC9** s'aligne sur la **ParitéFrame**. Sur le nouveau frame, on a alors des C9 impairs avec un C4 pair, et des C9 pairs pour un C4 impair.

Lorsque R8 passe de 0 à 3 (mode IVM ON), **ParitéC9** est immédiatement affecté avec la parité du C9 courant : **ParitéC9=C9.0**. La parité des C9 peut donc être en contradiction avec la parité du frame en cours jusqu'au prochain frame, ou la parité C9 s'aligne alors sur la parité du frame.

Les schémas pages suivantes décrivent différentes situations de comptage, lors du passage en mode IVM ou lors de la sortie du mode IVM.

Remarque : La période IVM de test couvre seulement quelques lignes du frame, et ce après la VSYNC, avant cette dernière et avant que C4 atteigne R6. R8 vaut 0 pendant la **VSYNC**.

R9=6 ou R9=7, et R8=0 avant de passer à 3 (ou repasser à 0).

Passage en mode IVM sur CRTC 3 & 4 :

R9=7, ODD OR EVEN FRAME

C4	C9	R8 UPDATE
0	0	R8=3
0	2	
0	4	
0	6	
0	8	
1	1	
1	3	
1	5	
1	7	
2	0	
2	2	
2	4	
2	6	
2	8	

R9=6, ODD OR EVEN FRAME

C4	C9	R8 UPDATE
0	0	R8=3
0	2	
0	4	
0	6	
1	0	
1	2	
1	4	
1	6	
2	0	
2	2	
2	4	
2	6	
3	0	
3	2	

C4	C9	R8 UPDATE
0	0	
0	1	R8=3
0	3	
0	5	
0	7	
1	0	
1	2	
1	4	
1	6	
1	8	
2	1	
2	3	
2	5	
2	7	

C4	C9	R8 UPDATE
0	0	
0	1	R8=3
0	3	
0	5	
0	7	
1	1	
1	3	
1	5	
1	7	
2	1	
2	3	
2	5	
2	7	
3	1	

C4	C9	R8 UPDATE
0	0	
0	1	
0	2	R8=3
0	4	
0	6	
0	8	
1	1	
1	3	
1	5	
1	7	
2	0	
2	2	
2	4	
2	6	

C4	C9	R8 UPDATE
0	0	
0	1	
0	2	R8=3
0	4	
0	6	
1	0	
1	2	
1	4	
1	6	
2	0	
2	2	
2	4	
2	6	
3	0	

R9=7, ODD OR EVEN FRAME

C4	C9	R8 UPDATE
0	0	
0	1	
0	2	
0	3	R8=3
0	5	
0	7	
1	0	
1	2	
1	4	
1	6	
1	8	
2	1	
2	3	
2	5	

R9=6, ODD OR EVEN FRAME

C4	C9	R8 UPDATE
0	0	
0	1	
0	2	
0	3	R8=3
0	5	
1	7	
1	1	
1	3	
1	5	
1	7	
2	1	
2	3	
2	5	
2	7	

C4	C9	R8 UPDATE
0	0	
0	1	
0	2	
0	3	
0	4	R8=3
0	6	
0	8	
1	1	
1	3	
1	5	
1	7	
2	0	
2	2	
2	4	

C4	C9	R8 UPDATE
0	0	
0	1	
0	2	
0	3	
0	4	R8=3
0	6	
1	0	
1	2	
1	4	
1	6	
2	0	
2	2	
2	4	
2	6	

Sortie du mode IVM sur CRTC 3 & 4 :

EVEN FRAME
EXIT IVM MODE, R9=7

C4	C9	R8 UPDATE
0	0	R8=3
0	2	R8=3
0	4	R8=3
0	6	R8=3
0	8	R8=3
1	1	**R8=0**
1	2	
1	3	
1	4	
1	5	
1	6	
1	7	
2	0	

ODD FRAME
EXIT IVM MODE, R9=6

C4	C9	R8 UPDATE
0	1	R8=3
0	3	R8=3
0	5	R8=3
0	7	R8=3
1	1	**R8=0**
1	2	
1	3	
1	4	
1	5	
1	6	
2	0	
2	1	
2	2	

C4	C9	R8 UPDATE
0	8	R8=3
1	1	R8=3
1	3	R8=3
1	5	R8=3
1	7	R8=3
2	0	**R8=0**
2	1	
2	2	
2	3	
2	4	
2	5	
2	6	
2	7	

C4	C9	R8 UPDATE
0	1	R8=3
0	3	R8=3
0	5	R8=3
0	7	R8=3
1	1	R8=3
1	3	**R8=0**
1	4	
1	5	
1	6	
2	0	
2	1	
2	2	
2	3	

C4	C9	R8 UPDATE
0	0	R8=3
0	2	R8=3
0	4	R8=3
0	6	R8=3
0	8	**R8=0**
1	0	
1	1	
1	2	
1	3	
1	4	
1	5	
1	6	
1	7	

C4	C9	R8 UPDATE
0	1	R8=3
0	3	R8=3
0	5	R8=3
0	7	R8=3
1	1	R8=3
1	3	R8=3
1	5	**R8=0**
1	6	
2	0	
2	1	
2	2	
2	3	
2	4	

EVEN FRAME
EXIT IVM MODE, R9=7

C4	C9	R8 UPDATE
0	0	R8=3
0	2	R8=3
0	4	R8=3
0	6	**R8=0**
0	7	
1	0	
1	1	
1	2	
1	3	
1	4	
1	5	
1	6	
1	7	

EVEN FRAME
EXIT IVM MODE, R9=6

C4	C9	R8 UPDATE
0	4	R8=3
0	6	R8=3
1	0	R8=3
1	2	R8=3
1	4	R8=3
1	6	**R8=0**
2	0	
2	1	
2	2	
2	3	
2	4	
2	5	
2	6	

C4	C9	R8 UPDATE
0	2	R8=3
0	4	R8=3
0	6	R8=3
0	8	R8=3
1	1	R8=3
1	3	**R8=0**
1	4	
1	5	
1	6	
1	7	
2	0	
2	1	
2	2	

C4	C9	R8 UPDATE
0	0	R8=3
0	2	R8=3
0	4	R8=3
0	6	R8=3
1	0	R8=3
1	2	**R8=0**
1	3	
1	4	
1	5	
1	6	
2	0	
2	1	
2	2	

C4	C9	R8 UPDATE
0	4	R8=3
0	6	R8=3
0	8	R8=3
1	1	R8=3
1	3	R8=3
1	5	**R8=0**
1	6	
1	7	
2	0	
2	1	
2	2	
2	3	
2	4	

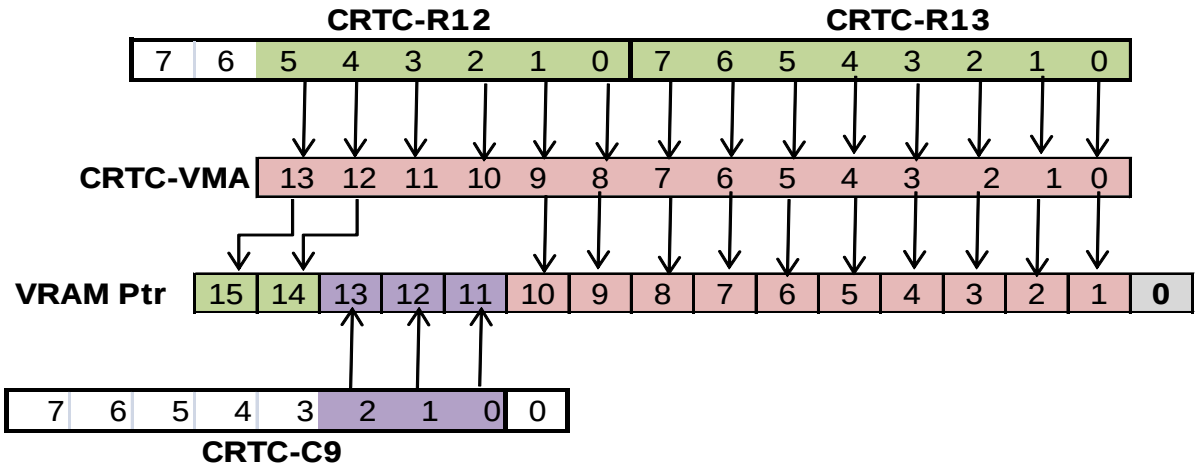
C4	C9	R8 UPDATE
0	2	R8=3
0	4	R8=3
0	6	R8=3
1	0	R8=3
1	2	R8=3
1	4	**R8=0**
1	5	
1	6	
2	0	
2	1	
2	2	
2	3	
2	4	

20 POINTEUR VIDEO:REGISTRES R12/R13

20.1 GÉNÉRALITÉS

Ces deux registres permettent de définir, en conjonction avec C9, l'adresse mémoire de base communiquée par le CRTC au GATE ARRAY pour qu'il affiche ses caractères.

20.2 CALCUL DU POINTEUR VIDÉO



Bit 0	**Always at 0 because the CRTC works in words**
Bits 1 to 10	**From bits 0 to 9 of CRTC-VMA**
Bits 11 to 13	**From bits 0 to 2 of C9**
Bits 14 and 15	**From bits 12 and 13 of CRTC-VMA**

Lorsque les conditions de mise à jour de VMA/VMA' sont remplies, alors VMA ou VMA'=R12/R13. Voir chapitre 14, page 120.

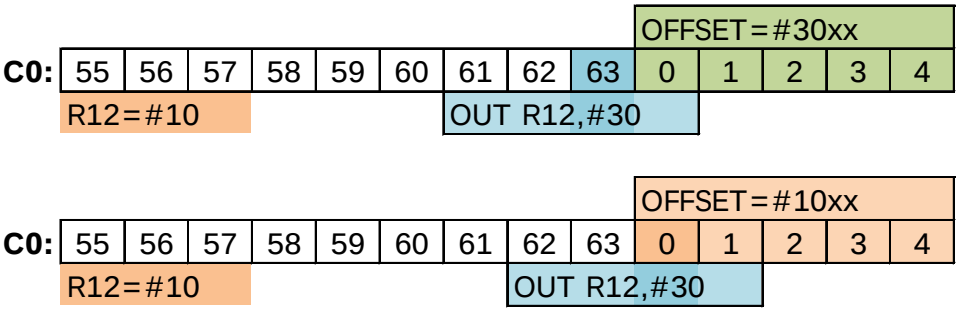
20.3 CONDITIONS DE MISE À JOUR

20.3.1 CRTC 0

Lorsque les compteurs C4, C9 et C0 passent à 0, les pointeurs vidéo (VMA' & VMA) sont initialisés avec R12/R13.

À noter que si les "frames" font 2 µsec (R4=R9=0 et R0=1) alors C4 passe par les valeurs 0 et 1 alternativement, permettant une prise en compte de R12 et/ou R13 toutes les 4 µs.
(Il s'agit du déclenchement de la gestion "R5" de ligne additionnelle, qui génère une ligne malgré R5=0, et incrémente donc C4 malgré R4=0).

La mise à jour de R12 et R13 est prise en compte immédiatement.



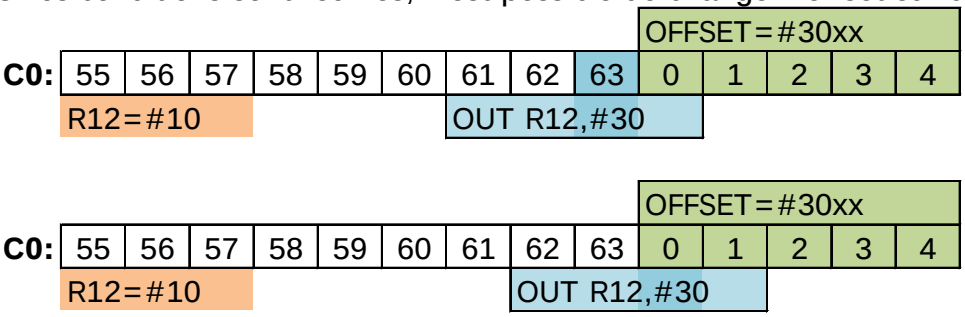
Le CRTC 0 charge **VMA' & VMA avec R12/R13 si C4=0 et C0=0.**

20.3.2 CRTC 1

Lorsque C0 et C9 passent à 0 et que C4=0, le pointeur vidéo VMA est initialisé avec R12/R13.

À noter que la mise à jour de R12 et R13 est immédiate.

Si les conditions sont réunies, il est possible de changer l'offset sur des frames de 1 µs (R0=0).



Le CRTC 1 charge **VMA avec R12/R13 tant que C4=0.**

Cette particularité offre une certaine souplesse car il est possible de modifier l'adresse sur une période bien plus grande que tous les autres CRTC. Pour permettre une compatibilité avec le CRTC 1, il est donc conseillé de ne pas modifier l'adresse pour le prochain frame trop tôt, lorsque C4=0. On peut par exemple constater ce bug au niveau de l'affichage de la végétation dans le jeu « **007 The Living Daylights** » (Domark, 1987). L'adresse de la zone de score est programmée trop tôt et remplace l'adresse de la zone de décor contenant la végétation.

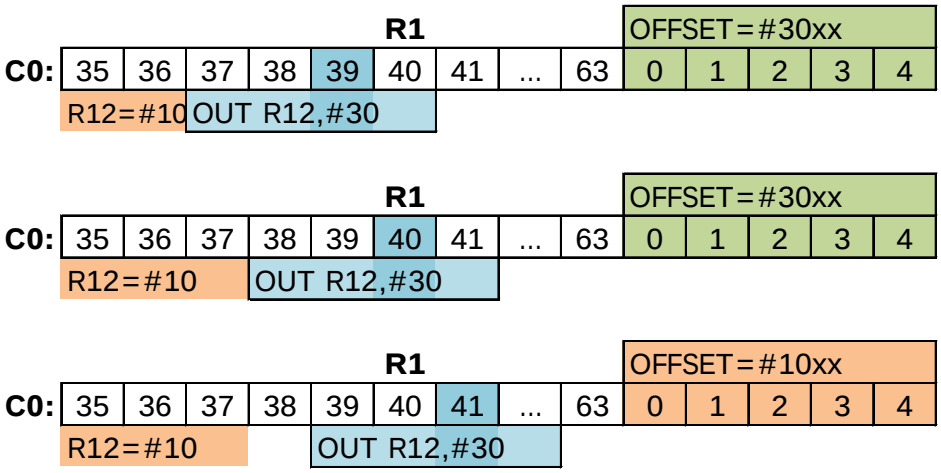
20.3.3 CRTC 2

Lorsque les compteurs C4, C9 et C0 passent à 0, le pointeur VMA' est initialisé avec R12/R13.

Contrairement aux autres CRTC, c'est VMA' qui est mis à jour avec R12/R13.

Pour rappel VMA' est le pointeur transitoire mis à jour lorsque C0 atteint R1, et qui vient mettre à jour VMA lorsque C9=R9.

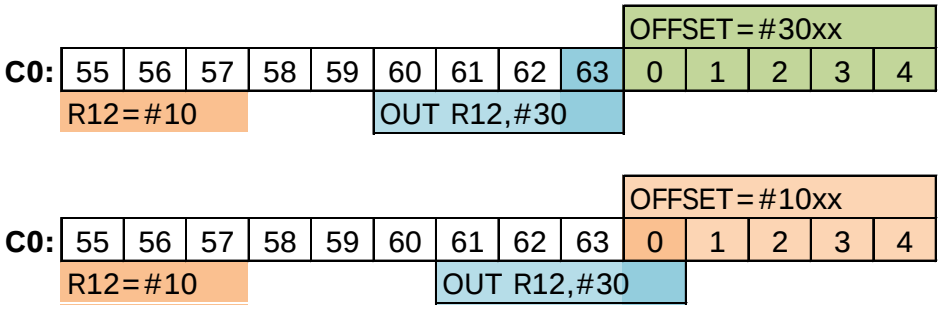
Si la mise à jour des registres R12/R13 est immédiate, la prise en compte pour la nouvelle ligne ne peut plus avoir lieu lorsque C0 dépasse R1.



Le CRTC 2 accepte de charger VMA' avec R12/R13 si C4=0 et C0=0.

20.3.4 CRTC 3 & 4

Lorsque les compteurs C4, C9 et C0 passent à 0, les pointeurs vidéo (VMA' & VMA) sont initialisés avec R12/R13.



Les CRTC 3 et 4 acceptent de charger VMA' & VMA avec R12/R13 si C4=0 et C0=0.

20.4 DELAIS DE PRISE EN COMPTE

Voir le chapitre sur le registre R0 pour le détail précis de la mise à jour du pointeur vidéo.

20.5 OVERSCAN-BITS

La construction de l'adresse mémoire est particulière dans la mesure ou :

- Les bits de C9 sont "inamovibles" dans le pointeur final.
- Le compteur interne d'adresse compte sur 14 bits avec les bits 10.11.12 qui ne "servent pas" (du moins en apparence).

R12

7	6	5	4	3	2	1	0
		p	p	o	o	a	a

R13

7	6	5	4	3	2	1	0
a	a	a	a	a	a	a	a

p	p	Start address
0	0	&0000
0	1	&4000
1	0	&8000
1	1	&C000

o	o	Managed size
0	0	16 k
0	1	16 k
1	0	16 k
1	1	32 k

	9	8	7	6	5	4	3	2	1	0
Start Addr +	a	a	a	a	a	a	a	a	a	a

Lorsque R12 et R13 sont transférés dans VMA/VMA', c'est VMA qui sert à afficher les caractères.

Bien que les bits 11, 12 et 13 du pointeur définitif proviennent de C9, le compteur VMA compte sur 14 bits. Aussi, lorsque le comptage arrive au bout de 10 bits d'adresse, les bits 2 et 3 venus de R12 participent au comptage, même si il ne servent pas directement pour le pointeur vidéo.

Si ces bits sont tous les deux à 1, cela provoque un report sur les bits 4 et 5 de R12.
Ces 2 bits représentent les bits 14 et 15 du pointeur vidéo.

La conséquence est donc un changement de page lorsqu'on arrive au bout des 10 bits.

Il est donc possible d'afficher une image de plus de 16k sans avoir recours à une rupture avec cette méthode.

21 REGISTRES EN LECTURE.

21.1 GÉNÉRALITÉS

Il existe deux types de lecture possibles sur les CRTC.

Le premier type a pour objet de récupérer la valeur contenue dans certains registres du CRTC. Selon les CRTC, cette lecture ne concerne pas les mêmes registres et les mêmes ports.

Le second type a pour objet de récupérer le statut de certaines opérations internes du CRTC afin de communiquer sur la réalisation de différentes conditions. La lecture de statuts n'existe pas sur tous les CRTC et n'utilise pas le même type de lecture.

21.2 LECTURE DU CONTENU DES REGISTRES

Afin de pouvoir lire le contenu d'un registre, il faut avant toute chose le sélectionner.
Pour rappel, la sélection d'un port I/O est réalisée avec l'adresse I/O &BC00.

21.2.1 CRTC 0

Seuls les 5 bits de poids faible du numéro de registre sont pris en compte. Les 3 autres bits de poids fort sont ignorés. Ce CRTC est capable de lire le contenu des registres suivants sur le port situé en &BF00:

Register	Definition	Unit	r/w	7	6	5	4	3	2	1	0
R12	Display start address (High)	Pointer	r/w	0	0						
R13	Display start address (Low)	Pointer	r/w								
R14	Cursor address (High)	Pointer	r/w	0	0						
R15	Cursor address (Low)	Pointer	r/w								
R16	Light Pen (High)	Pointer	r	0	0						
R17	Light Pen (Low)	Pointer	r								

Remarque : Le curseur n'est pas géré sur CPC. Cependant, il est parfaitement possible de stocker des valeurs dans R14 et R15 et les relire ensuite.
Une tentative de lecture d'un autre registre (0 à 255) renvoie la valeur 0.

21.2.2 CRTC 1, 2

Seuls les 5 bits de poids faible du numéro de registre sont pris en compte. Les 3 autres bits de poids fort sont ignorés. Ces CRTC sont capables de lire le contenu des registres suivants sur le port situé en &BF00 :

Register	Definition	Unit	r/w	7	6	5	4	3	2	1	0
R14	Cursor address (High)	Pointer	r/w	0	0						
R15	Cursor address (Low)	Pointer	r/w								
R16	Light Pen (High)	Pointer	r	0	0						
R17	Light Pen (Low)	Pointer	r								

Remarque : Le curseur n'est pas géré sur CPC. Cependant, il est parfaitement possible de stocker des valeurs dans R14 et R15 et les relire ensuite.

Pour le CRTC 2, une tentative de lecture d'un autre registre (0 à 255) renvoie la valeur 0.
Pour le CRTC 1, une tentative de lecture d'un autre registre renvoie 0, à l'exception du registre 31 (et toutes les valeurs dont les bits 0 à 4 sont à 1) qui renvoie une valeur non nulle (j'ai obtenu 127 ou 255). Ce registre a probablement été défini par UMC sans être utilisé sur ce modèle.

21.2.3 CRTC 3, 4

Pour les CRTC 3 et 4, seuls les 3 bits de poids faible du numéro de registre sélectionné sont considérés pour lire un registre selon la table suivante.

Lire le registre 4 revient donc également à lire le registre 12 (8+4) ou le 20 (16+4)
La lecture d'un registre CRTC est possible sur deux adresses I/O indistinctement: &BE00 et &BF00

On notera que le port « Register Status » a la même fonction que le port « Register Read ». Ces CRTC gèrent néanmoins une collection impressionnante de statuts. Les concepteurs ont utilisé les registre R10 et R11 à cette fin (voir chapitre suivant sur les statuts).

Ces CRTC sont capables de lire le contenu des registres suivants :

Nb	Register	Definition	Unit	r/w	7	6	5	4	3	2	1	0
0	R16	Light Pen (High)	Pointer	r	0	0						
1	R17	Light Pen (Low)	Pointer	r								
2	R10	Asic CRTC Status 1	Function	r								
3	R11	Asic CRTC Status 2	Function	r								
4	R12	Display start address (High)	Pointer	r/w								
5	R13	Display start address (Low)	Pointer	r/w								
6	R14	Cursor address (High)	Pointer	r/w	0	0						
7	R15	Cursor address (Low)	Pointer	r/w								

Remarque : Le curseur n'est pas géré sur CPC. Cependant, il est parfaitement possible de stocker des valeurs dans ces registres et les relire ensuite, y compris sur les CRTC 3 et 4.

21.3 LECTURE DES STATUS

21.3.1 GÉNÉRALITÉS

Seul le CRTC 1 dispose d'un registre de status présent sur le port spécifique &BE00.
Ce port est un miroir du port de lecture pour les CRTC 3 et 4, qui gèrent les status autrement.

CRTC	7	6	5	4	3	2	1	0
0	x	x	x	x	x	x	x	x
1	x	L	V	x	x	x	x	x
2	x	x	x	x	x	x	x	x
3	d	d	d	d	d	d	d	d
4	d	d	d	d	d	d	d	d

High impedance

High impedance

L	
1	Light pen reading
0	R16 / R17 registers can be read
V	
1	BORDER R6 is true
0	BORDER R6 is false

d : Mirror of BF00 port (read of CRTC reg)

Other CRTC	7	6	5	4	3	2	1	0
UM6845E	U	L	V	x	x	x	x	x
R6545E	U	L	V	x	x	x	x	x

U	
1	Update event
0	Reg R31 has been read / written by the MPU

21.3.2 CRTC 0, 2

Ces deux CRTC **ne possèdent pas** de registre de status.
C'est pourquoi, il est déconseillé d'utilisé la valeur lue sur le port &BE00 sur les CRTC 0 et 2, notamment pour tester le type du CRTC (risque de Candy Crush).
Mon CPC CRTC 2 renvoie toujours 255 en lecture sur ce port.
Mon CPC CRTC 0 renvoie aléatoirement 255 ou 127 sur ce port.

21.3.3 CRTC 1

L'adresse d'entrée/sortie du registre de status sur ce CRTC est &BE00.

Sur le CRTC 1, le bit 5 du registre de Status est mis à jour lorsque C0=R0 selon les conditions BORDER R6 (Faux: C4=C9=C0=0 / Vrai: C4=R6 & C9=C0=0).

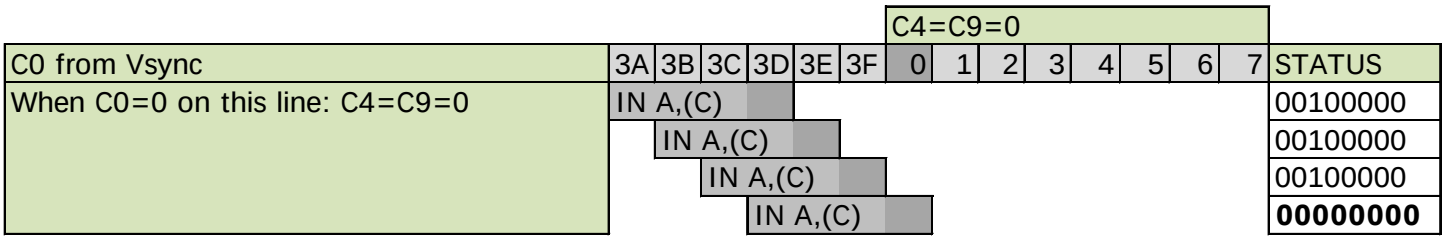
Le bit vaut 1 lorsque la condition BORDER R6 est vraie.
Le bit vaut 0 lorsque la condition BORDER R6 est fausse.

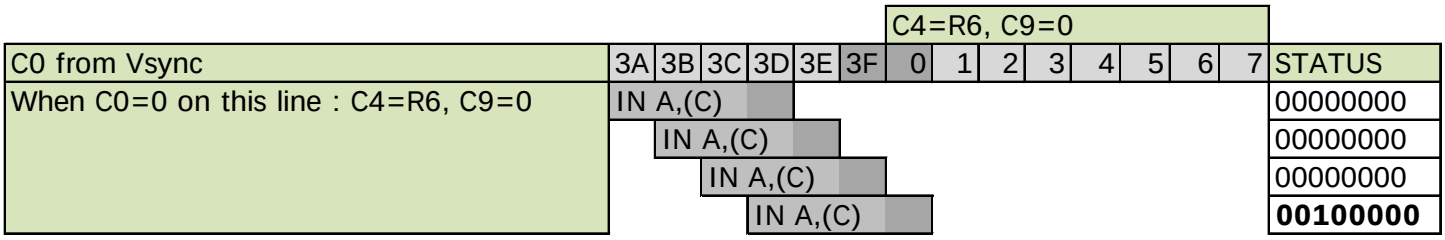
Cela ne signifie par forcément que du BORDER ou des CARACTERES sont affichés, car cette évaluation est faite lorsque C0=R0.

Or du BORDER peut déjà être affiché si la condition BORDER R1 est vraie.

À noter également que si R6 est positionné à 0 (alors que C4>0) pour générer du BORDER, cet état n'est pas détecté. La valeur 0, comme d'autres registres du CRTC 1, est gérée de manière particulière. Autrement dit, si R6=0 alors que C4>0 et que le statut valait 0 (Caractères affichés), le bit 5 du registre de status continuera de valoir 0.

Les schémas sur la page suivante décrivent le passage exact à 1 ou 0 de ce bit de status.





21.3.4 CRTC 3, 4

Les concepteurs de ces ASIC ont utilisé R10 et R11 comme registres de statuts afin de tracer un nombre important d'évènements. Il est ainsi possible d'attendre des évènements très précis et de calculer assez simplement la valeur de tous les registres du CRTC qui sont en lecture seule.

La première identification de ces statuts a été faite sur le CPC PLUS par Kevin THACKER (ArnoldEmu) qui y fait référence ici : http://cpctech.cpc-live.com/docs/cpcplus.html
Cependant, c'était une erreur d'affirmer que ces status n'existent pas sur le CRTC 4.

La lecture de ces status demande une grande précision car plusieurs bits changent d'état uniquement durant 1 µsec. En l'occurrence, si le registre de status est lu avec une instruction **INI** ou **IND**, l'I/O est lue et écrite en ram durant la 4ème µseconde de l'instruction (qui en prend 5). Lorsque le registre est lu avec une instruction **IN reg8,(C)**, l'I/O est lue durant la 4ème µseconde de l'instruction (qui en prend 4). Il est donc nécessaire de caler parfaitement les instructions de lecture afin que la 4ème µseconde soit située au moment exact ou le status est positionné par le CRTC.

Remarque : Étant donné que le bit 3 du numéro de registre est forcé à 1, lire les registres R2 et R3 revient à lire R10 et R11.

21.3.4.1 Définition du STATUS 1

STATUS 1 (CRTC-R10)		
Bit number	**Bit Value**	**Event**
0	1	C0=R0
1	0	C0=R0/2
2	0	C0=R1-1
3	0	C0=R2
4	0	C0=R2+R3
5	0	C0=0..R0 on last Vsync Line
6	1	Always 1
7	0	On some specific C0 values for some C4 values :

Line from Vsync, to...		C0	C4 start	C9start	C9end	Note
8	15	39	31	0	7	R7=30
15	22	63	32	7	6	
56	63	55	37	0	7	
64	71	15	38	0	7	R4=38
72	78	63	0	0	6	
112	119	55	5	0	7	
120	127	15	6	0	7	
168	175	31	12	0	7	
216	223	47	18	0	7	
224	231	7	19	0	7	
272	279	23	25	0	7	

Le bit 1 de ce statut est positionné à 0 lorsque C0=R0 >> 1.
Le CRTC considère juste que le bit 0 de la valeur comparée avec C0 démarre sur le bit 1 de R0.

21.3.4.2 Définition du STATUS 2

STATUS 2 (CRTC-R11)		
Bit number	**Bit Value**	**Event**
0	0	C4=R4 and C9=R9 and C0=R0 : Last char of screen
1	0	C4=R6-1 and C9=R9 and C0=R0 : Last char displayed
2	0	C4=R7-1 and C9=R9 and C0=R0 : Last char before Vsync
3	0	see below
4	1	Always 1
5	0	C9=R9 : Status=0 for C0=0 to R0
6	0	Always 0
7	1	(C9=R9 and C0=63) or (C9=0 and C0=0 to R0-1)

Sur CRTC 4, la valeur du bit 3 change à chaque µseconde du frame sur une trame spécifique par périodes de 6 µsecondes: **0, 0, 1, 0, 1, 1**

Sur CRTC 3, il existe 2 trames différentes.

Une première trame de 3 µsecondes qui part de C0vs au dernier caractère du frame (C4=R4, C9=R9 et C0=R0): **0, 0, 1**

Une seconde trame, identique à celle du CRTC 4, qui part du nouveau frame (C4=0, C9=0, C0=0) jusqu'au premier caractère de la **VSYNC** : **0, 0, 1, 0, 1, 1**

21.4 DUMMY REGISTER

Parmi quelques uns des mythes et légendes à propos des CRTC du CPC, figure l'existence du registre 31, appelé DUMMY REGISTER.

Si ce registre 31 existe bien sur le CRTC UM6845E de la société UMC, **il n'existe pas** sur les CRTC UM6845R (type 1), ni sur les CRTC UM6845 (type 0).

Sur le CRTC UM6845R, le bit 7 du registre de status qui y fait référence est simplement inutilisé.
Sur le CRTC UM6845, le registre de status lui-même n'existe pas (de là à trouver son bit 7...).

Cependant, il faut noter que la lecture de ce registre sur un CRTC UM6845R (type 1) renvoie une valeur non nulle, contrairement aux autres registres en lecture seule.

Sur le CRTC UM6845E (hors CPC), le registre 31 est en relation avec le mode transparent.

La gestion de ce mode utilise également les bits 6 et 7 du registre 8 en plus du bit 7 du registre de status. Je ne m'étendrais pas sur ce sujet, qui est aussi intéressant que la programmation du EF9345P...

22 FULLSCREEN & CENTRAGE

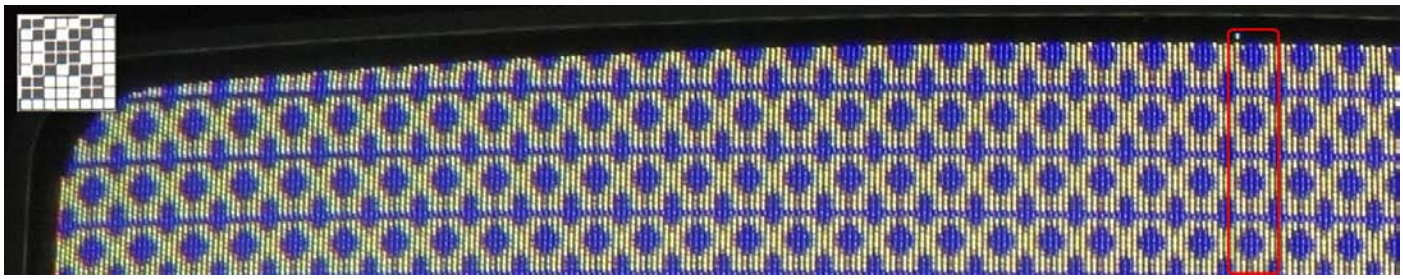
22.1 PRÉAMBULE

Les écrans cathodiques du CPC sont convexes avec une courbure d'écran assez visible.
À cause de cette forme et de l'angle de déviation du faisceau d'électrons, les pixels situés sur les bords sont plus grands que ceux situés au centre.

De plus, le boitier qui entoure le tube cathodique dispose de coins et bords arrondis qui épousent « à peu près » la forme du tube cathodique, mais masquent cependant au passage quelques pixels complémentaires. Les données indiquées dans ce chapitre correspondent au cas général. À cause de la nature analogique des écrans, il peut y avoir quelques petites variations de taille ou de positionnement entre 2 écrans. N'oublions que le CPC a plus de 30 ans.

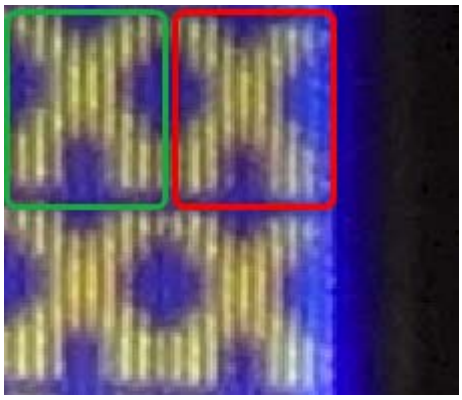
On peut constater des écarts de plusieurs pixels horizontaux et lignes verticales entre le centre de l'écran et les bords de ce dernier.

Verticalement, un écart de 5 lignes peut exister entre la première ligne visible dans un coin et la première ligne visible au milieu de l'axe horizontal de l'écran :

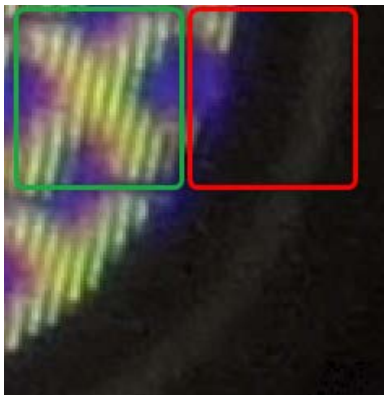


Horizontalement, un écart de presque 1 caractère CRTC peut être constaté entre une colonne située au niveau du coin latéral haut ou bas de l'écran et cette même colonne au milieu de l'axe vertical de l'écran (ligne 136) :

Colonnes au milieu à droite.



Mêmes colonnes dans le coin inférieur droit.



C'est en comptant le nombre de caractères horizontaux et de lignes verticales à partir du milieu des axes horizontaux et verticaux de l'écran qu'on peut déterminer les valeurs à programmer pour que l'intégralité des pixels visibles de l'écran soient « occupés ». Il faut toutefois garder à l'esprit que de nombreux pixels de ces lignes et colonnes ne seront pas visibles.

22.2 FULLSCREEN HORIZONTAL

Sur un moniteur CTM, on peut visualiser une ligne horizontale de **48 caractères CRTC**, ce qui représente 96 octets (192 pixels mode 0, 384 pixels mode 1, 768 pixels mode 2).

Le centrage d'une ligne horizontale dépend du signal HSYNC (voir chapitre 16, page 140).
Le premier caractère visible à gauche est le 15ème caractère à partir de la position C0=R2.

Pour obtenir un centrage exact sur une ligne de 64 µsec (R0=63), on doit positionner R2=50 (avec R3>=6) pour afficher C0=0 à l'extrême gauche de l'écran.

Selon le réglage H-HOLD du moniteur (accessible via un petit tournevis plat dans le petit trou à l'arrière gauche du moniteur), on peut voir apparaitre le BORDER à droite ou à gauche si la taille de la ligne est définie à 48 caractères.

Si R3 est inférieur à 6, alors le frame est décalé à droite sur l'écran (voir chapitre 14.4, page 122).

22.3 FULLSCREEN VERTICAL

Sur un moniteur CTM, on peut visualiser **272 lignes** verticales à 50 Hz (et bien plus avec un «Interlace maison » en abaissant la fréquence, mais c'est un autre sujet).

L'image commence à être visible à partir de la 34ème ligne, ce qui représente la deuxième ligne du 5ème caractère de 8 lignes à partir du début de la VSYNC. (voir chapitre 15, page 132).

Si R7 est positionné avec 35 alors que R4=38 (et R5=R8=0), alors 33 lignes « non visibles » seront générées à partir de la VSYNC, et la ligne C4=0/C9=1 sera visible en partie.

23 TRUCS ET ASTUCES

Dès qu'il est nécessaire de modifier les registres de plusieurs circuits, les 64 µsec d'une ligne sont souvent une contrainte importante à respecter. Plusieurs astuces permettent de gagner de précieuses µsec. Les principales optimisations résident dans l'économie de registres Z80A, la méthode d'accès aux entrées-sorties ou la gestion des itérations. Ces 3 approches sont combinables à loisir.

23.1 MISE À JOUR DE R12/R13

Lorsqu'il est question de modifier fréquemment les valeurs des registres de certains circuits, il n'est par exemple pas toujours nécessaire de modifier les 2 registres R12/R13 lorsque certaines ruses sont employées en temps critique, en général pour les démos.

Il est possible de n'affecter que l'un ou l'autre, ou même de manière dissociée selon l'architecture mémoire de ce qui doit être affiché.

Le mode d'accès au CRTC, qui implique la sélection préalable d'un registre, est très consommateur en CPU. (tout étant relatif...). Une option pour minimiser le temps d'accès à ces deux registres est d'utiliser l'instruction **OUT (nn),A** avec toutes les réserves qui entourent l'utilisation de cette instruction malicieuse sur un CPC.

Exemple 1 :

Le code suivant, par exemple, permet d'initialiser le couple R13/R12 rapidement (en tenant bien sûr compte du préchargement des registres). Le principe est de permettre la sélection d'un registre CRTC sans manipuler B plusieurs fois pour basculer sur la fonction d'écriture de données du circuit.

Prérequis : B=&BC, A=12, C=13, H=R12, L=R13
(facultativement, SP=&BABA)

```
OUT (C),C       ; Sélect R13
INC B           ; B=Crtc Data Address
OUT (C),L       ; R13=L
OUT (#FF),A     ; Select R12 (12="combo" CRTC Reg Select & Valeur Reg Select)
OUT (C),H       ; R12=H
```

Il est possible de sélectionner les registres CRTC 8, 4 et 0 avec cette méthode. Cependant les deux dernières valeurs envoient également la valeur du registre A sur le port A du PPI.

Exemple 2 : R0=0

```
LD B,#BD        ; B=Crtc Data Address
XOR A
OUT (#FF),A     ; Select R0
OUT (C),A       ; or OUT(C),0 (see chapter 23.4)
```

Dans le contexte où seuls les registres 12 et 13 du CRTC sont modifiés pluiseurs fois une astuce consiste à permuter l'ordre d'affectation des registres entre chaque ligne afin d'économiser la sélection d'un registre. Cela permet de sélectionner un seul registre par ligne et d'affecter ce registre 2 fois de suite. (R12, R13 / R13, R12 / R12, R13 /...).

C'est le cas par exemple lorsqu'il faut modifier les 2 registres sur chaque ligne dans la cadre d'une rupture ligne à ligne (RLAL) sur les CRTC 0, 1, 3 et 4 :

Contexte initial : Selection R12
Première ligne : Mise à jour R12 / Sélection R13--Mise à jour R13
Seconde ligne : Mise à jour R13 / Sélection R12--Mise à jour R12
Troisème ligne : Mise à jour R12 / Selection R13--Mise à jour R13
Et ainsi de suite...

Remarque : ce principe de gain de CPU pour la sélection d'un index lorsque plusieurs registres d'un même circuit sont mis à jour séquentiellement est applicable aux autres circuits qui gèrent un index de sélection de registre, comme le GATE ARRAY ou le AY-3-8912.

Ainsi, par exemple, si il est nécessaire de changer la couleur de plusieurs pen sur chaque ligne, il est judicieux d'ordonner l'ordre des pen « communs » entre chaque ligne :

Première ligne : Select Pen 1--Coul pen 1 / Select Pen 2--Coul Pen 2 / Select Pen 3--Coul Pen 3
Seconde ligne : Coul Pen 3 / Select Pen 1--Coul Pen 1 / Select Pen 2--Coul Pen 2
Troisième ligne : Coul Pen 2 / Select Pen 3--Coul Pen 3 / Select Pen 1--Coul Pen 1
Et ainsi de suite...

23.2 UTILISATION COMMUNE DE REGISTRE(S)

Dans le but d'économiser les précieux registres du Z80A, quelques astuces consistent à placer la valeur des ports d'entrée/sortie ou la valeur des registres de sélection dans des registres qui servent également de pointeurs sur des tables. Cela limite cependant l'accès à des tables dont le poids fort de l'adresse, par exemple, est une valeur valide pour le circuit.
Cela pose néanmoins une contrainte importante sur l'organisation de la mémoire.
Par exemple, il est possible de placer en mémoire aux adresses &BC00, &BD00, &BE00, &7F00 des tables contenant des valeurs utiles.

Exemple 1 :

```
LD B,#BD        ; BC est à la fois port d'écriture CRTC
LD A,(BC)       ; et pointeur sur la valeur à écrire (indexée par C ici)
OUT (C),A
```

Exemple 2:

```
LD B,#BE        ; BE est port d'écriture CRTC (via OUTI)
LD H,B          ; Mais également pointeur sur la valeur à écrire (indexée par L)
OUTI            ; HL étant également incrémenté dans l'opération
```

Ce sport peut également s'appliquer aux registres de sélection, comme 12 lorsqu'il est question de modifier R12 par exemple.

```
LD B,#BC
LD H,#0C        ; HL est un pointeur sur une table entre #0C00 et #0CFF
OUT (C),H       ; Mais H sert aussi de sélecteur de registre CRTC
INC B
INC B
OUTI            ; La valeur présente à l'adresse #0Cxx est envoyée dans R12
```

23.3 ATTENTE DE VSYNC

Il est parfois intéressant que B soit déjà préchargé à l'issue d'une attente « traditionnelle » de **VSYNC**. Il est possible d'utiliser l'instruction **IN A,(#FF)** pour éviter que B soit impliqué, mais cela implique de recharger A avec l'adresse I/O du Port B (PPI) à chaque itération :

```
WSYNC   LD A,#F5                    LD B,#F5
        IN A,(#FF)          WSYNC   IN A,(C)
        RRA                         RRA
        JR NC,WSYNC                 JR NC,WSYNC
```

En chargeant A à l'aide d'un autre registre 8 bits préchargé, la boucle occupe alors le même temps qu'une boucle « standard » avec B car IN A,(#FF) s'exécute en 3 µsecondes.

23.4 VALEUR 0

Il existe une instruction non documentée (#ED,#71) intéressante qui permet d'envoyer la valeur 0 sur l'adresse d'I/O présente dans le registre B : **OUT (C),0**.
À priori, la valeur envoyée par cette instruction dépend du type de logique MOS du Z80A. Sur un NMOS, qui équipe les CPC, c'est 0 qui est envoyé. Sur un Z80 CMOS, c'est la valeur 255.
Exemple :

```
LD BC,#BC09     ; Voir chapitre 12.4.2, page 92
OUT (C),C
INC B
OUT (C),C
OUT (C),0
```

23.5 OUTI/OUTD ET REGISTRE D'ETAT

La documentation officielle est fausse concernant les bits N et C de du registre F du Z80A pour les instruction **OUTI** et **OUTD**. (c'est sans doute le cas aussi pour OTIR et OTDR, mais d'un intérêt limité sur CPC). Lorsque la somme de la valeur envoyée (présente en HL) au circuit et le registre L (post traitement (incrémenté/décrémenté)) est supérieur à 255, alors N=C=1. Cela permet potentiellement de tester la fin d'une table sans compteur mais cela crée une contrainte sur l'adresse de la table.

23.6 CODE AUTOMODIFIÉ

Lorsqu'un code est exécuté en ram, il possède un avantage par rapport à un code exécuté en rom : Il peut modifier son propre code objet. Ceci présente des avantages en terme de longueur de code et de cpu. C'est le cas notamment lorsqu'il est question de stocker la valeur d'un compteur, qui nécessite une variable.
Exemple d'une fonction gérant un compteur :
Ram/Rom :

```
                LD A,(MyCounter)        ; Charge la valeur du compteur dans A
                DEC A                   ; Modifie le compteur
                LD (MyCounter),A        ; Sauvegarde la valeur du compteur
                RET
MyCounter       DB 100
```

Ram :

```
MyCounter       LD A,100                ; Charge la valeur du compteur dans A
                DEC A                   ; Modifie le compteur
                LD (MyCounter+1),A      ; Sauvegarde la valeur du compteur
                RET
```

L'adresse de la variable **MyCounter** est ici présente directement dans l'opérande de l'instruction **LD A,n** (cette instruction est codée **#3E, n**).
L'opérande **n** se trouve donc à 1 octet du début de l'instruction.

Selon les instructions, l'offset de l'opérande à modifier peut varier :
Pour un **LD H,valeur** l'opérande valeur est situé à l'offset +1 du début de l'instruction.
Pour un **LD IXH, valeur**, l'opérande est situé à l'offset +2 du début de l'instruction.

L'auto-modification implique donc de tenir compte de l'offset relatif des opérandes dans les instructions concernées.

Des méthodologies existent pour éviter d'utiliser des « MyCounter + offset » partout dans un source car cela peut poser des problèmes de maintenance parce que cela implique d'aller modifier le source partout ou MyCounter est utilisé. (par exemple un **LD H,valeur** est transformé en **LD IXH, valeur**).

La solution la plus utilisée consiste à utiliser une directive **EQU** qui définit la variable relativement au code. Dans l'exemple précédent, on aurait :

```
                LD A,100            ; Charge la valeur du compteur dans A
MyCounter       EQU $-1
                DEC A               ; Modifie le compteur
                LD (MyCounter),A    ; Sauvegarde la valeur du compteur
                RET
```

MyCounter pointe alors à l'adresse d'assemblage (signalée conventionnellement par le caractère $) moins 1. Des assembleurs peuvent sans doute aller plus loin en proposant une directive combinée à **EQU** capable de fournir l'offset relatif des opérandes de l'instruction selon leur type.
Un autre exemple de modification d'opérande sur 16 bits est indiqué dans le chapitre 23.8 avec la sauvegarde du registre SP.

L'auto-modification présente aussi l'intérêt de pouvoir modifier les instructions elles-mêmes.
Les possibilités sont nombreuses.

Exemple 1 : Inhiber ou activer un traitement

```
                ;
                ; Activer le traitement
                ;
                LD A,#37    ; Opcode de "SCF"
                LD (EtatBitC),A
                ;
                ; Désactiver le traitement
                ;
                LD A,#B7    ; Opcode de "OR A"
                ld (EtatBitC),A

                ...
                ...
EtatBitC        SCF
                JR C,Traitement1 / CALL C, Traitement1 / RET NC
```

Un assembleur moderne devrait en principe permettre de récupérer le (ou les) opcodes d'une instruction. Par exemple avec une directive « **Opcode(« instruction »)** ».
La gestion d'une permutation d'instruction pourrait alors s'écrire :

```
                LD A,(EtatBitC)                          ; 4 µs
                XOR (Opcode(« SCF ») xor Opcode(« OR A»)) ; 2 µs
                LD (EtatBitC),A                          ; 4 µs
                ...
                ...
EtatBitC        SCF/OR A                                 ; 1 µs
                JR C, Traitement                         ; 3/2 µs
```

Le **XOR** sert ici à permuter de #37 à #B7 et vice-versa.
Il est aussi parfaitement envisageable de modifier l'instruction de branchement elle-même.
Par exemple transformer un **JR C** (#38) en **JR NC** (#30).
On peut aussi modifier la valeur de branchement située dans l'opérande situé derrière l'opcode pour une instruction de branchement (JR, JP, CALL).

Si la permutation a pour objet un flip-flop de traitement, on peut aussi écrire :

```
                LD A,%01010101           ; 2 µs
FlipFlopSt      EQU $-1
                RRCA                     ; 1 µs
                LD (FlipFlopSt),A        ; 4 µs
                JR C,Traitement          ; 3/2 µs
```

Il est possible par exemple d'inverser le sens de comptage d'un compteur en transformant un INC en DEC, ou même en désactivant le comptage via un NOP :

```
                LD A,100                 ; Charge la valeur du compteur dans A
MyCounter       EQU $-1
CounterFunc     DEC A                    ; DEC A / INC A / NOP
                LD (MyCounter),A         ; Sauvegarde la valeur du compteur
                RET
```

L'automodification de code permet d'inverser facilement des instructions de comptage, afin d'éviter des tests ou l'écriture d'une 2ème fonction :

- Un XOR 1 sur les opcodes INC A ,INC B, INC C, INC D, INC E, INC H, INC L permet de les transformer respectivement en DEC A, DEC B, DEC C, DEC D, DEC E, DEC H, DEC L, et vice-versa.

- Un XOR 8 sur les opcodes INC BC, INC DE, INC HL, INC SP permet de les transformer respectivement en DEC BC, DEC DE, DEC HL, DEC SP et vice-versa.

Dans un contexte tendu avec des I/O, il est courant d'avoir des OUT (C),reg8 qui se suivent.
Il est intéressant de se souvenir ici que ces instructions occupent 2 octets en ram et commencent toutes par le préfixe #ED. Il est possible de modifier facilement le registre affecté en modifiant l'octet qui suit le préfixe #ED (#79=A/#41=B/#49=C/#51=D/#59=E/#61=H/#69=L).
La valeur #71 permet d'envoyer la valeur 0 (avec le Z80A installé dans les CPC).

Dans le cas particulier de l'instruction OUT(C),C l'automodification permet de transformer l'instruction. Il est par exemple possible de neutraliser cette instruction en modifiant le préfixe avec une instruction neutre (NOP ou LD reg8,reg8) mais cela présente l'inconvénient de modifier la durée de l'instruction, qui passe alors de 4 à 2 µsec.

Dans un contexte de temps fixe, on pourra remplacer l'instruction complète par #232B (INC HL/DEC HL). Il est déconseillé de modifier le registre B avec #FF pour désactiver l'envoi sur un port donné, car quelques extensions CPC ont eu la brillante idée d'utiliser uniquement C pour s'identifier.

Remplacer un OUT(C),C par un IN (C) (#ED70) est également déconseillé car un IN sur CPC peut provoquer un OUT avec la valeur qui se trouve sur le bus.

Enfin, pour clore ce chapitre des instructions modifiées, il est important de citer l'instruction ADD HL,BC (opcode #09) qu'il est possible de transformer en ADD HL,SP (opcode #39) et vice-versa. Ceci peut être fort utile pour une routine de sprite « déroulée » lorsque BC=#800 et SP=#C800+(R1x2). Cela permet d'éviter de créer 8 routines d'affichage de sprite différentes pour gérer tous les cas d'un sprite plus haut que 8 lignes et susceptible d'être affiché sur n'importe quelle ligne d'un frame.

23.7 ITÉRATIONS ET CODE DÉROULÉ

Lorsque plus aucune optimisation n'est possible, une méthode très employée est d'éliminer les tests et instructions de branchements de boucle. Cela peut cependant entrainer une hausse non négligeable de la ram nécessaire. Le « déroulement » du code peut avoir lieu au niveau même des instructions du Z80A, comme LDIR ou LDDR.

Exemple :

```
LD BC,5                 ; 3 µs
LD HL,Source_pointer    ; 3 µs
LD DE, Dest_pointer     ; 3 µs
LDIR                    ; 29 µs ((6x4)+5)
                        ------
                         38 µs
```

```
LD HL,Source_pointer    ; 3 µs
LD DE,Dest_pointer      ; 3 µs
LDI                     ; 5 µs
LDI                     ; 5 µs
LDI                     ; 5 µs
LDI                     ; 5 µs
LDI                     ; 5 µs
                        -----
                         31 µs
```

Chaque « LDI » utilise 1 µs de moins que si l'opération avait eu lieu au sein d'un « LDIR ».
Mais chaque µsec gagnée ainsi coûte alors 2 octets de code (un LDI est codé « #ED,#A0 »).
Cela peut donc être très consommateur en mémoire (rom/ram) disponible.

Remarque : Lorsque le code fonctionne en ram, il est possible de créer un « code qui génère du code » plutôt que d'utiliser des « macros » de l'assembleur qui font ce travail. Dans l'exemple ci-dessus, un code pourrait générer les LDI à exécuter en ram. Cela permet d'obtenir des codes objets moins volumineux, et de mettre en perspective la notion de partage de ressource mémoire. La zone dédiée au code généré pouvant être réutilisée par d'autres fonctions.

Une alternative au « tout ou rien » consiste à minimiser la quantité de ram nécessaire au code « déroulé » en acceptant de perdre un peu de CPU pour « boucler » un peu moins souvent. Il s'agit donc ici de faire un ratio entre la ram consommée et la CPU gagnée.

Exemple :

```
LD BC,2000              ; 3 µs                  ; 3 octets
LD HL,Source_pointer    ; 3 µs                  ; 3 octets
LD DE,Dest_pointer      ; 3 µs                  ; 3 octets
LDIR                    ; 11999 µs ((6x2000)–1) ; 2 octets
                        ; ----------            ; ---------
                        ; 12008 µs              ; 11 octets
```

Le code ci-dessus occupe 11 octets, et s'exécute en 12008 µs.

Si le LDIR avait été intégralement déroulé pour les 2000 occurences définies dans BC, le code correspondant aurait duré 10006 µs ((2000 x 5)+6).
Mais en contrepartie, ce code aurait occupé 4006 octets en ram ou en rom. (2000x2+6).
Exemple d'une méthode mixte où la « répétition » périodique de LDIR est simulée par une boucle:

```
        LD HL,Source_pointer  ; 3 µs          ; 3 octets
        LD DE,Dest_pointer    ; 3 µs          ; 3 octets
        LD A,200              ; 2 µs          ; 2 octets
LOOP
        LDI                   ; 5 x 200 µs    ; 2 octets
        LDI                   ; 5 x 200 µs    ; 2 octets
        LDI                   ; 5 x 200 µs    ; 2 octets
        LDI                   ; 5 x 200 µs    ; 2 octets
        LDI                   ; 5 x 200 µs    ; 2 octets
        LDI                   ; 5 x 200 µs    ; 2 octets
        LDI                   ; 5 x 200 µs    ; 2 octets
        LDI                   ; 5 x 200 µs    ; 2 octets
        LDI                   ; 5 x 200 µs    ; 2 octets
        LDI                   ; 5 x 200 µs    ; 2 octets
        DEC A                 ; 1 x 200 µs    ; 1 octet
        JR NZ,LOOP            ; (3 x 200)-1 µs ; 2 octets
                              ; ------------   ; ------------
                              ; 10799 µs       ; 31 octets
```

Dans l'exemple ci-dessus, le code occupe 31 octets et consomme 10799 µsec.

Si on avait ajouté 10 LDI (portant leur nombre à 20) en diminuant la boucle a 100, on aurait un code plus long (31+20=51 octets) mais plus rapide (100 boucles de 104 µs + 7µs=10407 µs).

Ce principe peut être appliqué à des portions de code fortement répétées.
Par exemple pour du code qui affiche des sprites.
Et notamment lors des opérations de masquage qui consistent à faire un « trou » dans le décor (par exemple avec une opération AND) pour y « coller » le sprite (opération OR) avant affichage.

23.8 BRANCHEMENT INCONDITIONNELS

Au lieu de traiter des branchements conditionnels avec des séries de tests, une méthode classique consiste à utiliser des listes de pointeurs d'exécution stockés dans la pile. Chaque instruction **RET** (3 µsec) amène le registre PC à basculer sur une nouvelle fonction (et SP à pointer sur l'adresse ou est stocké l'adresse de la fonction suivante).

Cette méthode nécessite en général de préparer la table en amont, notamment pour gérer les cas de sortie si cela n'est pas prévu par ailleurs. Cette méthode condamne néanmoins l'usage des interruptions (sauf à l'avoir prévu).

Exemple :

```
EXEC_LIST1      DW FUNCTION1        EXEC_LIST2      DW FUNCTION2
                DW FUNCTION2                        DW FUNCTION1
                DW FUNCTION3                        DW FUNCTION3
;--------------------------------
FUNCTION1
                ...
                RET
;--------------------------------
FUNCTION2
                ...
                RET
;--------------------------------
FUNCTION3
                ...
SP_BACKUP1      LD SP,0                 ; Code auto modifié
                RET
;--------------------------------
MAIN
        DI
        LD (SP_BACKUP1+1),SP            ; Automodification du code
        LD SP,EXEC_LIST1                ; ou EXEC_LIST2
        RET
```

Dans une situation de temps critique, les instructions **JP HL, JP IX** et **JP IY** sont précieuses.
La première de ces 3 instructions s'exécute en 1 µsec (les 2 autres en 2 µs).
L, LX ou LY peuvent servir d'index de branchement à condition que le point d'entrée de toutes les fonctions soit présent dans la page de 256 octets définie par H, IXH, IYH.

Remarque : La notation ZILOG JP (HL/IX/IY) est trompeuse, car le saut a lieu à l'adresse contenue dans le registre HL/IX/IY et non à l'adresse contenue à l'adresse pointée par HL/IX/IY. Les assembleurs récents acceptent ces nouvelles notations.

Enfin, et c'est très anecdotique, il est toujours possible d'utiliser les interruptions du CPC et s'arranger pour que R52 génère une interruption afin d'interrompre une boucle, en tenant compte qu'une instruction non répétitive n'est pas sécable. Il faut toutefois tenir compte que les interruptions à ce stade ne sont pas fiables entre les différents CPC lorsque des instructions longues doivent être interrompues. (voir chapitre 26.7.2)

23.9 TRAITEMENTS PAR PAGES

Le Z80A reste un processeur 8 bits, même si il est capable de manipuler des valeurs 16 bits.

Cependant, les instructions qui utilisent cette possibilité sont plus lentes que celles qui manipulent des valeurs 8 bits. Il est ainsi plus rapide de faire un « INC L » (1 µs) qu'un « INC HL » (2 µs).

Par ailleurs, les accès à la ram sont plus rapides en adressage indirect.
LD A,(BC) ou LD (HL),E vont durer 2 µsec par exemple.

Un adressage absolu va être coûteux ('LD (adresse),A' 4µs et 'LD (adresse),DE' 6µs) et n'a pas grand intérêt lorsqu'il est question de traiter une table ou une structure.

Les adressages indexés sont assez indigestes parce que la valeur d'index « n » est stockée en dur dans l'instruction (LD A,(IX+n) 5µs). Ces instructions sont de surcroit très lentes car le pointeur est calculé sur 16 bits.

Dans un contexte de performance, l'architecture du Z80A favorise l'organisation des données par pages de 256 octets. La plupart des assembleurs intègrent des directives qui permettent d'aligner des tables sur des frontières de page (lorsque le poids faible de l'adresse vaut 0).

Il est ainsi possible d'accéder facilement au contenu d'une table en manipulant le poids faible de l'adresse contenue dans BC, DE ou HL. On considére alors que C, E ou L sont les index d'une table de 256 octets pointée respectivement par B, D ou H.

Ainsi pour définir une structure de données contenant plusieurs valeurs, l'architecture par page offre de très bonnes performances d'accès. Imaginons (par exemple) qu'on souhaite définir une structure qui contient 2 types de données : Mxx de type word (16 bits) et Nxx de type byte (8 bits).

Et ça tombe bien, nous avons besoin de disposer moins de 256 fois de cette structure.

On peut définir les choses ainsi

```
MNStruct    DW M00          ; Index 0
            DB  N00
            ;
            DW M01          ; Index 1
            DB N01
            ;
            DW M02          ; Index 2
            DB N02
            ...
            ...
            DW M85          ; Index 85
            DB N85
```

L'accès à une des structures de **MNStruct** consiste à multiplier l'index de la structure à atteindre par la taille de la structure (ici 3 octets) et additionner cet offset ainsi obtenu à **MNStruct**. Il suffit ensuite de lire (ou écrire) les valeurs stockées à l'adresse calculée. Le calcul et l'accès à ces structures impliquent de travailler sur 16 bits.

Si **MNStruct** est aligné en début de page (par exemple la table débute en #4000), il faut calculer l'adresse sur 16 bits si la taille totale des structures dépasse 256 octets.
Dans notre exemple, c'est le cas car l'index dépasse 84.
Il faut donc en tenir compte lorsqu'on fait évoluer le pointeur sur la structure calculée dans HL, BC ou DE.

Dans notre exemple la structure d'index 85 est située en #4000+(85x3)=#40FF
Le poids faible de M85 est situé sur la page #40, et les 2 autres valeurs de la structure sur la page #41. Si DE contenait #40FF, il faut faire un INC DE et non un INC E pour accéder à la valeur suivante avec LD A,(DE) ou LD (DE),A. À noter toutefois que si la table fait moins de 2 pages de 256 octets, le dépassement de page ne se produira que pour une seule des valeurs de la structure. Dans l'exemple, le INC DE concernera alors uniquement le calcul du pointeur sur le poids fort de M85, l'accès à N85 nécessitant uniquement un INC E.

Si 256 est un multiple de la taille de la structure, alors il n'est plus nécessaire de gérer l'incrémentation sur la structure courante avec des instructions 16 bits, car aucune structure ne se trouve alors entre 2 pages. Par exemple, si la structure faisait 4 octets, il y aurait exactement 64 structures par page de 256 octets. Il peut donc être pertinent d'ajouter 1 octet à la structure pour éviter d'avoir à utiliser une incrémentation 16 bits.

Une autre manière d'organiser les données est de définir 3 tables débutant chacune en frontière de page pour chaque valeur 8 bits déclarée dans la structure :

```
TabMLow     DB M00Low
            DB M01Low
            ...
            DB M85Low

TabMHigh    DB M00High
            DB M01High
            ...
            DB M85High

TabN        DB N00
            DB N01
            ...
            DB N85
```

Cette organisation facilite grandement les choses et est très performante.

Si par exemple, **TabMLow** est en #4000, **TabMHigh** est en #4100 et **TabN** est en #4200, alors le poids faible du pointeur désigne l'index et il suffit de jouer avec le poids fort pour changer de donnée.

Avec H=#40 et L=Index, on peut écrire :

```
LD E,(HL)   ; 2 µs ; Récupère TabMLow[Index] dans E
INC H       ; 1 µs ; Change de table pour se positionner sur TabMHigh
LD D,(HL)   ; 2 µs ; Récupère TabMHigh[Index] dans D
INC H       ; 1 µs ; Change de table pour se positionner sur TabN
LD C,(HL)   ; 2 µs ; Récupère TabN[Index] dans C
```

Ce type d'organisation est idéal lorsque 256 structures sont définies, car il n'y a pas de « gaspillage » de mémoire. Mais si ce n'est pas le cas et que les structures sont courtes, rien n'empêche d'en loger plusieurs au sein d'une même page. Dans notre exemple, les 86 premiers octets de chaque page sont utilisés, mais rien n'empêche de gérer une autre structure à partir de l'offset 86 de chaque page.

On peut aussi envisager qu'une valeur contenue dans une structure indexée soit l'index d'une autre table de structures. C'est une chose assez courante. Toujours avec l'exemple précédent, supposons que N soit un index sur une autre table située en #4300 (Tab2).
Si B contient #43, on peut écrire dans la continuité de l'exemple précédent :

```
LD C,(HL)   ; 2 µs ; Récupère TabN[Index] dans C
LD A,(BC)   ; 2 µs ; Récupère Tab2[TabN[Index]] dans A
```

Si, par un heureux hasard, il est possible de faire coexister au sein d'une page un autre index sur cette même page, on peut alors écrire :

```
;
LD L,(HL)   ; Lire dans L la valeur de R0, qui est un index pour trouver R9
LD H,(HL)   ; Lire dans H la valeur de R9
```

A plus petite échelle qu'une table alignée sur une page de 256 octets, on peut considérer qu'une structure de données peut débuter sur une **adresse paire ou impaire**. Des directives assembleur permettent de respecter cette contrainte. Ceci permet d'optimiser l'accès des données pointées par HL, par exemple, en utilisant successivement des **INC L / INC HL** si l'adresse de début des données est **paire,** et **INC HL / INC L** si l'adresse de début des données est **impaire**. En effet, on est certain que le dépassement de frontière de page ne se produit que sur une adresse impaire (lorsque L vaut 255).

On peut étendre cette notion d'alignement à des quantités multiples de 256 octets par page, afin de minimiser le nombre d'incrémentations sur 16 bits. Ainsi si une table est alignée sur une frontière de **16 octets dans une page de 256 octets**, alors on pourra utiliser **15 fois INC L et 1 fois INC HL** (pour un pointeur défini dans HL), car il y aura seulement 16 fois la probabilité que la zone ainsi définie soit en frontière de page de 256 octets. Ces techniques permettent de minimiser l'usage des incrémentations 16 bits, 2 fois plus coûteuses en CPU que des incrémentations 8 bits.

23.10 TRUCS EN VRAC

23.10.1 BOUCLES

Lors de l'écriture d'une boucle qui teste si un pointeur 16 bits a atteint une certaine valeur, une solution consiste à comparer respectivement le poids fort et le poids faible avec les valeurs à atteindre. Si la comparaison est réalisée avec le poids fort en premier, le poids faible du pointeur sera comparé dans un 2ème temps plus long, et ce jusqu'à ce que la condition d'égalité avec le poids faible soit vraie. Par contre, si la comparaison est réalisée avec le poids faible en premier, c'est le poids fort du pointeur qui sera comparé dans la 2ème partie du test autant de fois que la valeur du poids faible sera atteinte. Selon la valeur du pointeur 16 bits à atteindre, il est donc pertinent d'inverser l'ordre de test du poids fort et du poids faible :

```
LOOP                                    LOOP
        INC HL                                  INC HL
        LD A,H                                  LD A,L
        CP HIGH(ADRESSE)                        CP LOW(ADRESSE)
        JR NZ,LOOP                              JR NZ,LOOP
        CP LOW(ADRESSE); 2eme partie            CP HIGH(ADRESSE) ; 2eme partie
        JR NZ,LOOP                              JR NZ,LOOP
```

Si on souhaite tester qu'un registre 16 bits a atteint 0, on peut le tester ainsi :

```
        LD BC, Nb_Occurences
LOOP
        ...
        DEC BC          ; 2 µs      ; 1 octet
        LD A,B          ; 1 µs      ; 1 octet
        OR C            ; 1 µs      ; 1 octet
        JR NZ,LOOP      ; 3/2µs     ; 2 octets
```

Mais on peut également faire ceci, qui est aussi rapide mais qui occupe moins de place, et ne modifie pas A :

```
        LD BC, Nb_Occurences+255
LOOP
        ...
        DEC BC          ; 2 µs      ; 1 octet
        INC B           ; 1 µs      ; 1 octet
        DJNZ LOOP       ; 4/3 µs    ; 2 octets
```

Dans l'hypothèse ou HL a besoin d'être incrémenté, il est aussi possible d'écrire :

```
        LD BC, Nb_Occurences
        LD HL, Mon_Pointeur
LOOP
        ...
        CPI             ; 4 µs      ; 2 octets
        JP PE,LOOP      ; 3 µs      ; 3 octets
```

23.10.2 CALCUL DU POINTEUR VIDEO

Un grand classique sur CPC consiste à calculer l'adresse de la ligne suivante dans un frame formaté de manière « standard », avec des lignes de 64 µs et R9=7.

Les 3 bits de poids faible de C9 correspondent aux bits 11.12.13 du pointeur vidéo.
Dis autrement, l'offset du pointeur vidéo est C9 x #800.

Pour passer sur la ligne suivante sur un frame qui n'est pas contraint par une rupture démoniaque, il suffit donc d'ajouter #800 au pointeur courant. Lorsque le pointeur est sur la 8ème ligne d'un caractère vertical (C9=7), il faut effectuer un calcul spécifique pour que le pointeur revienne sur la première ligne de la ligne-caractère suivante (C9=0) : (#C000+(R1x2)).

La routine la plus classique pour une page vidéo débutant en #C000 est la suivante, à quelques variantes près :

```
                Variante 1          Variante 2
NEXTLINE        LD HL,PtrVideo      ...
                LD A,H              ...             ; 1
                ADD A,8             ...             ; 2
                LD H,A              ...             ; 1
                RET NC              ...             ; Test Dépassement
                LD BC,#C000+(R1x2)  LD A,L          ; Sans modifier BC
                ADD HL,BC           ADD A,R1x2
                RET                 LD L,A
                                    LD A,H
                                    ADC A,#C0
                                    LD H,A
                                    RET
```

Le test de dépassement de page, qui permet de savoir si la dernière ligne a été atteinte est valable pour une page située en #C000, mais pas sur une autre page. En effet, additionner 8 au poids fort d'une page située en #0000, #4000 ou #8000 ne fait pas déborder le pointeur et ne positionne par le flag C à 1. La table ci-après décrit les tests possibles selon les différentes pages.

#0000	#4000	#8000	#C000

LD A,H
ADD A,8
LD H,A
ADD A,A
RET P
	LD A,H
	ADD A,8
	LD H,A
	RET P
		LD A,H
		ADD A,8
		LD H,A
		ADD A,A
		RET P
			LD A,H
			ADD A,8
			LD H,A
			RET NC

Il est aussi possible de réaliser le calcul de l'offset sur une page donnée et appliquer ensuite une correction de l'offet en fonction de la page de 16k souhaitée.

Une solution consiste également à précalculer les pointeurs de début de ligne dans une table.
Un index correspond alors au numéro de ligne Y permettant d'accéder à ces pointeurs.
Mais cela implique de devoir ajouter la position X au pointeur ainsi calculé.

Une autre méthode consiste à modifier H grâce aux instruction **RES (2 µs)** et **SET (2 µs)**. Cependant, on ne peut pas accéder aux lignes linéairement avec cette méthode en utilisant à chaque fois 1 seule instruction. C'est cependant possible en ne respectant pas l'ordre des lignes. Mais cela nécessite alors d'organiser les données graphiques dans l'ordre de leur écriture.

Exemple :

```
LD HL,#C000          ; HL=#C000 (C9=0)
LD (HL),%00000001    ;
SET 3,H              ; HL=#C800 (C9=1)
LD (HL),%00000011    ;
SET 4,H              ; HL=#D800 (C9=3)
LD (HL),%00001111    ;
RES 3,H              ; HL=#D000 (C9=2)
LD (HL),%00000111
SET 5,H              ; HL=#F000 (C9=6)
LD (HL),%01111111
SET 3,H              ; HL=#F800 (C9=7)
LD (HL),%11111111
RES 4,H              ; HL=#E800 (C9=5)
LD (HL),%00111111
RES 3,H              ; HL=#E000 (C9=4)
LD (HL),%00011111
```

Chaque changement de ligne prend 2 µs/2 octets contre 3 µs/3 octets en considérant une solution qui sacrifie un registre et A pour réaliser la somme. Avec B=8 offert en sacrifice sur l'autel du CRTC, on aurait entre chaque ligne :

```
LD A,H      ; 1 µs
ADD A,B     ; 1 µs (avec B=8)
LD H,A      ; 1 µs
```

Une des particularités des scrolls hardware est que la mémoire vidéo cycle tant que les bits de pages ne sont pas affectés. (voir chapitre 20.5). Si par exemple la mémoire cycle sur la page de 16k située en #C000 (car les « Overscan Bits ™ » ne sont pas à 1), cela signifie que le CRTC va afficher l'octet situé #C000 après celui affiché en #C7FF. Et lorsqu'il faut recalculer le pointeur qui avait atteint la dernière ligne d'un caractère (C9=7 dans l'exemple) il est nécessaire d'en tenir compte lorsque R1x2 est ajouté à ce pointeur. En l'occurence si le pointeur vidéo avait dépassé #10000-(R1x2), il est nécessaire de corriger le pointeur calculé en anihiliant son bit 3 :

```
NEXTLINE    LD HL,PtrVideo
            LD A,H
            ADD A,8
            LD H,A              ; PtrVideo+#800
            RET NC              ; Test débordement sur page #C000
            LD A,L              ; Retour sur ligne 0
            ADD A,R1x2          ; En additionnant #C000+(R1x2)
            LD L,A
            LD A,H
            ADC A,#C0
            RES 3,A             ; Correction du pointeur vidéo
            LD H,A
            RET
```

Les choses peuvent singulièrement se compliquer pour parcourir la ram vidéo lorsque le pointeur est en frontière de zone de cyclage. La mémoire n'étant pas linéaire, il n'est pas toujours possible de corriger simplement le pointeur, et notamment sur les C9 impairs, comme on peut le voir dans le tableau qui suit :

C9	Pointeur courant	Ptr+1	Pointeur calculé	Correction	Pointeur corrigé
0	C7FF	INC HL	C800	RES 3,H	C000
1	CFFF	INC HL	D000	SET 3,H RES 4,H	C800
2	D7FF	INC HL	D800	RES 3,H	D000
3	DFFF	INC HL	E000	SET 3,H SET 4,H RES 5,H	D800
4	E7FF	INC HL	E800	RES 3,H	E000
5	EFFF	INC HL	F000	SET 3,H RES 4,H	E800
6	F7FF	INC HL	F800	RES 3,H	F000

7	FFFF	INC HL	0000	LD A,H
				OR #F8
				LD H,A
					F800

Effectuer une correction à chaque incrémentation d'un pointeur impair pour tenir compte d'une situation qui se produit une fois sur 2048 est extrêmement pénalisant lorsqu'il est question d'afficher des sprites sur un scrolling hardware, par exemple. De plus, les méthodes d'affichage de sprites les plus performantes nécessitent de pouvoir parcourir la ram vidéo dans les deux sens, ce qui représente une nouvelle complexité à gérer.

Une solution consiste à créer des routines spécifiques d'affichage lorsque le pointeur sur les données à parcourir dans la ram vidéo est sur la frontière de cyclage. Il est alors nécessaire d'identifier que la ligne-caractère traitée « contient » cette frontière, et appeler un code spécifique pour traiter la transition.

Par exemple, si il est question d'afficher un sprite de 4 octets de large (8 pixels mode 0, symbolisés par les zones en couleur) sur une ligne C9=0, alors il faut prévoir que ce sprite pourra occuper les emplacements suivants en mémoire vidéo :

C7FC	C7FD	C7FE	C7FF	C000	C001	C002	C003
C7FC	C7FD	C7FE	C7FF	C000	C001	C002	C003
C7FC	C7FD	C7FE	C7FF	C000	C001	C002	C003
C7FC	C7FD	C7FE	C7FF	C000	C001	C002	C003
C7FC	C7FD	C7FE	C7FF	C000	C001	C002	C003

Pour les cas particuliers indiqués en orange, il est par exemple possible de gérer deux pointeurs issus d'une table indexée par C9, afin de satisfaire les exigences d'un code spécifique. Dans l'exemple, on aurait par exemple HL=C7FF et HL'=C000 pour C9=0, permettant au code spéficique de permuter entre les deux adresses courantes avec l'instruction EXX (1 µs). La particularité de frontière serait ainsi circonscrite, permettant de ne faire aucune correction de pointeur dans le cas général.

23.10.3 POSITIONNEMENT DES FLAGS

Plus un rappel qu'un réel « truc »
La table suivante décrit les instructions disponibles selon quelques états du registre F

Asm	Définition	Flag	JR	JP	CALL	RET
Z	Nul	Z=1				
NZ	Non nul	Z=0				
C	Report	C=1				
NC	Pas de report	C=0				
PE	Parité paire	P=1				
PO	Parité impaire	P=0				
P	Positif	S=0				
M	Négatif	S=1				

Les états de F le plus souvent utilisés sont C et Z.
Ci-après quelques instructions pour les positionner :

Etat Z	Etat C	Instruction(s)	Mise à jour
1	0	CP A	
0	0	OR #F6	A=#F6
--	1	SCF	
--	0	OR A	
1	1	CP A + SCF	
0	1	SCF + SBC A,A	A=#FF

23.10.4 PLUTÔT QUE...

NEG (2 octets/2 µs) est équivalent à **CPL :INC A** (2 octets / 2 µs)

Plutôt que '**NEG :ADD A,d'** (4 octets / 4 µs)
Préférez '**CPL :ADD A,d+1**' (3 octets / 3 µs)

Plutôt que **'BIT 6,A :JP NZ,Kloug**' (5 octets / 5 µs)
Préférez '**ADD A,A : JP M,Kloug**' (4 octets / 4 µs)

Plutôt que **'LD A,reg8 : NEG'** (3 octets / 3 µs)
Préférez **'XOR A :SUB reg 8'** (2 octets / 2 µs)

Plutôt que **'LD A,0**' (2 octets / 2 µs)
Préférez **'XOR A**' (1 octet / 1 µs) mais uniquement si vous voulez que Z soit mis à 1 en plus.

Plutôt que **'CP 0**' (2 octets / 2 µs)
Préférez **'OR A**' (1 octet / 1 µs)

Plutôt que **'XOR #FF**' (2 octets / 2 µs)
Préférez **'CPL**' (1 octet / 1 µs)

24 UNE BRÈVE HISTOIRE DE TEMPS FIXE

24.1 INTRODUCTION

Sur CPC, la durée de chaque instruction est stable et linéarisée par le cadencement imposé par le GATE ARRAY au Z80A lorsqu'il a besoin d'accéder à la mémoire. Contrairement à d'autres processeurs, la vitesse des instructions ne dépend pas de l'ordre des instructions ou du type de ram à partir de laquelle elles sont lues par le processeur.

Cette particularité permet de calculer assez facilement le temps pris par une fonction, puisqu'il suffit de connaitre le nombre de µsecondes pris par chaque instruction.
Les instructions les plus courtes font 1 µs. C'est notamment le cas de l'instruction NOP, qui est souvent utilisée pour qualifier la durée d'un ensemble d'instructions.

La durée de certaines instructions peut toutefois être différente selon la réalisation d'une condition traitée par l'instruction. C'est notamment le cas des branchements conditionnels relatifs (JR cond, offset ou DJNZ offset), des appels de fonction (CALL cond,adresse) ou des retours de fonction conditionnels (RET cond). C'est aussi le cas pour les instructions répétitives (LDIR, LDDR, OTIR, OTDR) lors de la dernière itération.

Vous pouvez consulter le chapitre 25, page 255 pour obtenir sur une seule page la liste des temps en µseconde de chaque instruction du Z80A.

La durée du code d'une fonction peut donc varier selon les différentes conditions satisfaites durant son exécution. Que ce soit pour réaliser de jolies démos ou certains jeux exigeants, il est nécessaire de pouvoir créer du code dont on connait la durée, et s'assurer que cette durée ne varie pas.

En effet, torturer certains circuits impose de les solliciter périodiquement de manière précise, souvent à la µseconde près. C'est le cas pour le CRTC lorsqu'il est question de certaines techniques barbares évoquées dans ce document qui nécessitent une précision de l'ordre de la µseconde. C'est également vrai pour le GATE ARRAY lorsqu'il est sollicité pour changer sa palette de couleurs ou son mode graphique. Ou bien encore pour le AY-3-8912 (le générateur sonore du CPC) lorsqu'on cherche à créer de nouvelles sonorités inédites.

24.2 METHODES

L'écriture d'un code en temps fixe nécessite de tenir compte des éléments susceptibles de faire varier la durée de ce code. En général, ce sont les branchements conditionnels, ou les boucles avec un nombre d'itérations variables qui sont un obstacle à la réalisation de cet objectif.

Sans entrer dans une logique de temps partagé, si on a une fonction en Z80A dont le temps est variable, son passage en « temps fixe » nécessitera que ce soit le délai maximum de cette fonction qui soit la référence de durée fixe.

Autrement dit, il est nécessaire ici d'introduire la notion de **compensation**.

Prenons par exemple une fonction A qui dure, selon les situations, 40, 60 ou 95 µs.
Cette même fonction en temps fixe devra toujours durer 95 µs.
Lorsque la fonction durera 40 µs elle devra compenser avec 55µs, et elle devra compenser 35 µs lorsqu'elle durera 60 µs.

L'écriture d'un code de compensation n'est pas toujours aisé.
Quelques astuces méthodologiques et outils peuvent faciliter la tâche.

Exemple 1 :

```
FONCTION_01             ; Fonction temps variable
          OR A          ;  1          ; Teste si A==0
          JR Z,CESTZERO ;  2  / 3
          LD HL,1234    ;  3  / 0
          LD A,2        ;  2  / 0
          RET           ;  3  / 0
CESTZERO
          LD A,1        ;  0  / 2
          RET           ;  0  / 3
                        ; 11  /  8  (Bilan)
```

A droite, deux colonnes symbolisent la durée des 2 branches d'exécution de **FONCTION_01**.

Lorsque la condition A>0 est remplie **JR Z,CESTZERO** ne saute pas en **CESTZERO**.
L'instruction dure néanmoins 2 µs. La durée de chaque instruction est ensuite indiquée à droite dans la première colonne, et ce jusqu'au **RET**.
Dans cette situation la fonction a duré **11 µs**.

Lorsque A=0, le **JR Z** saute en **CESTZERO** en 3 µs.
Dans cette situation la fonction a duré **8 µs**.

La solution ici est facile à comprendre.
La compensation doit être de 11-8=3 µs.
Il est donc possible d'ajouter 3 **NOP** avant le **RET** après le **LD A,1**, et le tour est joué.
La fonction durera alors toujours **11 µs**.

L'exemple précédent est assez simple car la fonction utilise 2 branches de code totalement distinctes. Mais les choses se compliquent un peu si les instructions de saut reviennent dans des branches communes.

Exemple 2 :

```
FONCTION_02             ; Fonction temps variable
          LD A,(COUNTER) ;  4  / 4 ; Lecture du compteur
          DEC A          ;  1  / 1 ; Décrémentation compteur
          JR NZ,NOZERO   ;  2  / 3 ; A atteint 0 ?
          LD A,10        ;  2  / 0 ; Oui rechargé avec 10
NOZERO
          LD (COUNTER),A ;  4  / 4
          RET            ;  3  / 3
                         ; 16 / 15
```

Dans cet exemple, le code à partir de **NOZERO** est commun.
La compensation est ici plus délicate à réaliser car on ne peut pas le faire à partir de **NOZERO**.

C'est le code non exécuté lorsque la condition est fausse (A=0) qui allonge la durée.

Une technique consiste à utiliser une seule instruction de 1 µs lorsque la condition n'est pas réalisée. Il s'agit en l'occurence d'instructions telle que **EXX / EX DE,HL / LD r8,r8 / INC r8**

```
FONCTION_02             ; Fonction temps fixe
          LD A,(COUNTER) ;  4  / 4 ; Lecture du compteur
          DEC A          ;  1  / 1 ; Décrémentation compteur
          LD B,10        ;  2  / 2
          JR NZ,NOZERO   ;  2  / 3 ; A atteint 0 ?
          LD A,B         ;  1  / 0 ; Oui rechargé avec 10 (contexte)
NOZERO
          LD (COUNTER),A ;  4  / 4
          RET            ;  3  / 3
                         ; 17 / 17
```

Le registre B a été sacrifié sur l'autel du temps fixe.
Mais **FONCTION_02** durera toujours 17 µs. Mission accomplie !

On voit ici qu'une solution consiste à préparer un **contexte** dont la durée est supérieure à 1 µs (ici **LD B,10**) et qui sera exploité par l'instruction de 1 µs qui suit le **JR condition** (**LD A,B**).

Ainsi, **EXX** et **EX DE,HL** sont des instructions de contexte particulièrement efficaces.

```
FONCTION_03             ; Fonction Temps variable
          OR A           ;  01 / 01
          LD HL,ADRESSE2 ;  03 / 03
          LD DE,#4321    ;  03 / 03
          JR Z,CESTZERO  ;  02 / 03
          LD HL,ADRESSE1 ;  03 / 00
          LD DE,#1234    ;  03 / 00
CESTZERO
          LD (HL) ,E     ;  02 / 02
          INC HL         ;  02 / 02
          LD (HL),D      ;  02 / 02
          RET            ;  03 / 03
                         ;  24 / 19
```

Ici, la solution est également assez simple à gérer avec **EXX**.

```
FONCTION_03             ; Fonction temps fixe
            OR A        ; 01 / 01
            LD HL,ADRESSE1 ; 03 / 03
            LD DE,#1234 ; 03 / 03
            EXX         ; 01 / 01
            LD HL,ADRESSE2 ; 03 / 03
            LD DE,#4321 ; 03 / 03
            JR Z,CESTZERO ; 02 / 03
            EXX         ; 01 / 00   Changement de contexte
CESTZERO
            LD (HL) ,E  ; 02 / 02
            INC HL      ; 02 / 02
            LD (HL),D   ; 02 / 02
            RET         ; 03 / 03
                        ; 26 / 26
```

Dans le cas ou un seul contexte n'est pas suffisant, il est possible de les enchainer tant qu'aucune instruction ne vient modifier l'état de condition.
Si on reprend l'exemple précédent, avant le **RET**, on pourrait avoir :

```
            ...
CESTZER0
            LD (HL) ,E  ; 02 / 02
            INC HL      ; 02 / 02
            LD (HL),D   ; 02 / 02
            LD A,10     ; 02 / 02
            LD B,20     ; 02 / 02
            JR Z,ZEROAGAIN ; 02 / 03
            LD A,B      ; 01 / 00   Second contexte
ZEROAGAIN
            RET         ; 03 / 03
                        ; 36 / 36
```

Il existe parfois des situations ou le contenu d'un registre va déterminer un traitement :

```
            CP 0
            JR Z,TRAITEMENT0
            CP 1
            JR Z,TRAITEMENT1
            CP 2
            JR Z,TRAITEMENT2
            ...
```

Si les valeurs contenues dans A sont proches, une méthode consiste à créer une table d'adresses de traitement indexée par A.

```
TRAITEMENT_TAB
            DW TRAITEMENT0
            DW TRAITEMENT1
            DW TRAITEMENT2
            ...
```

Le code de branchement est assez simple et son exécution est en temps fixe.

```
LD L,A          ; Index x 2
LD H,0
ADD HL,HL
LD BC,TRAITEMENT_TAB
ADD HL,BC       ; Pointeur sur TRAITEMENT_TAB [Index]
LD A,(HL)       ; HL=TRAITEMENT_TAB[Index].PtrExec
INC HL
LD H,(HL)
LD L,A
JP (HL)
```

Si les valeurs testées ne sont pas linéaires et ne peuvent pas servir d'index, il est possible de créer une table des valeurs associées aux adresses, et parcourir l'intégralité de la table, **même lorsque la valeur est trouvée** avant d'avoir atteint la fin de la table. Pour la valeur trouvée, il suffit alors de retenir l'adresse trouvée.

Créer une fonction de recherche d'index générique en temps fixe est un allié précieux.

24.3 TEMPS FIXE ET INTERRUPTIONS

De manière générale, il est plutôt préférable de désactiver les interruptions si vous n'avez pas prévu de les comptabiliser. Mais rien n'empêche de les prendre en compte.

Cela n'a rien de complexe, dans la mesure ou l'appel en #38 (mode IM 1) dure 5 µs, quel que soit le CPC. Il faut cependant compter 7 µs en IM 2. Voir chapitre 26.4 pour plus d'informations.

De nombreux programmes se synchronisent grâce aux interruptions, et notamment grâce à l'instruction HALT. Cette méthode est souvent utilisée pour permettre de définir les périodes de code en « temps variable » avant que l'interruption se produise. Le code exécuté après l'instruction HALT est alors réservé au « temps fixe ».

Une difficulté est que beaucoup de méthodes d'optimisation de CPU impliquent que la pile serve de pointeur pour lire des données. Or, la pile est notamment utilisée par les instructions d'appel de fonctions (CALL, RST). Et une interruption réalise un RST. Aussi le contenu de la pile est modifié avec l'adresse de l'instruction où l'interruption a été acceptée par le Z80A.

Le code de l'interruption peut certes prévoir de « corriger » la ram « pervertie » pour revenir au code interrompu, mais cela nécessite de savoir exactement ce qui va être détruit et où :

```
MON_INTER
                LD (BACKUP_HL+1),HL; Sauvegarde HL
                POP HL              ; Récupère l'adresse du code interrompu
                LD (RET_INT_ADR+1),HL ; Prévoir la fin de l'interruption
                LD HL,xxxx          ; Valeur corrective ram pointée par SP
                LD (yyyy),DE        ; Correction de la ram
BACKUP_HL       LD HL,#2121         ; Restitution de HL
                EI                  ; Acquit. interruption
RET_INT_ADR     JP #C3C3            ; Retour au code interrompu
```

Il est plus simple de désactiver les interruptions avec un simple DI.

Cependant, cela entraine un biais, car si les interruptions doivent de nouveau être autorisées, cela ne peut pas être fait n'importe où si elles avaient été désactivées pendant plus de 52 lignes. Dans ce cas, la réactivation des interruptions durant les 20 dernières lignes d'un des prochain groupes de 52 lignes suivants entraine un décalage des interruptions, et il faut donc en tenir compte.

Enfin, pour terminer ce chapitre, il est utile de préciser qu'une interruption se produit à la fin d'une HSYNC. Sur les CRTC 3 et 4, la HSYNC se produit 1 µs plus tard que sur les CRTC 0, 1 et 2.

A programmation de R2 et R3 équivalentes les interruptions surviennent donc 1 µs plus tard sur les CRTC 3 et 4. Une méthode couramment employée consiste à réduire R3 de 1 pour permettre une certaine compatibilité. De plus, il faut ajouter à ce décalage 1 µs de plus car les I/O CRTC des instructions OUT(C),C sur les CRTC 3 et 4 se produisent 1 µs plus tard (OUTI n'étant toutefois pas concerné par cette différence).

24.4 OUTILS COMPENSATOIRES

Les exemples du chapitre précédent montrent qu'une programmation en temps fixe nécessite de faire des calculs. Cela peut considérablement alourdir la maintenance d'un code dès qu'on ajoute une seule instruction.

Des assembleurs peuvent intègrer des directives permettant de calculer la cpu des instructions situées entre 2 labels et utiliser la valeur ainsi trouvée pour « **compenser** ».
Imaginons par exemple les directives **CPU_START et CPU_END:<Variable>**

Exemple de **compensation** d'une fonction dont la **période** est de 64 µs.

```
            LD HL,TAB_JOLIES_COULEUR      ;
            LD B,#7F
            OUT (C),0                     ; Pen 0
            LD A,20                       ; 20 lignes de 64 µs
LOOP_START
CPU_START:
            LD A,(HL)                     ; 2 µs
            OUT (C),A                     ; 4 µs
            INC HL                        ; 2 µs
            DEC A                         ; 1 µs
CPU_END: DureeCPU
            DEFS 64-DureeCPU-3,[NOP]
            JP NZ,LOOP_LINE               ; 3 µs
LOOP_END
```

Dans cet exemple, l'objectif est que le code présent entre **LOOP_START** et **LOOP_END** dure toujours 64 µs, quel que soient les mises à jour susceptibles d'être réalisées ensuite entre les directives **CPU_START** et **CPU_END** par le programmeur.
L'assembleur va calculer la CPU dans **DureeCPU**.
Dans notre exemple, **DureeCPU vaut 9 µs**.
Il est nécessaire de tenir compte des 3 µs de la boucle (le **JP NZ**), pour définir le nombre de NOP à créer pour la compensation (via **DEFS**) afin que la durée soit toujours égale à 64 µs.

La programmation en temps fixe nécessite de pouvoir « **perdre** » du temps.

On peut attendre « longtemps », comme c'est le cas avec la fonction **wait_usec** détaillée à la fin du chapitre 7.2. Cette fonction est prévue pour permettre d'attendre une quantité de µsec définie sur 16 bits, mais ne permet pas de définir une valeur d'attente inférieure à 49 µs.

Il est possible d'écrire une fonction plus précise :

```
Wait_64     nop
...         ...
...         ...
Wait_13     nop
Wait_12     nop
Wait_11     nop
Wait_10     nop
Wait_09     nop
Wait_08     ret
```

Ainsi un **CALL Wait_08** prendra très exactement 8 µs (Le CALL suivi du RET).
Si **Wait_64** est considéré comme le délai de référence de cette fonction, alors le pointeur d'exécution pour un délai d'attente compris entre 8 et 64µs est **Wait_64+64-Delai**.

Pour des délais plus courts que 8 µs, il est possible d'utiliser des insctructions neutres.
Plusieurs de ces instructions sont décrites dans le chapitre 24.7.

Le calcul par un assembleur de la CPU entre 2 labels reste cependant très **limitatif pour gérer la compensation**. C'est utile, mais cela ne permet pas de tenir compte de la réalité d'exécution du code. Par ailleurs, même en adoptant des réflexes de programmation en temps fixe, les mises à jour d'un source peuvent se révéler fastidieuses, voir ingérables.

Il existe un outil qui permet de calculer la CPU située entre deux adresses afin d'**automatiser la compensation**. Il a été créé pour permettre la démocratisation de ce type de programmation. Sa mise en oeuvre permet de concevoir du code en temps fixe sur l'intégralité d'un frame (19968 µs) sans avoir à faire des calculs fastidieux. De plus un code en temps fixe sur une période de 19968 µs rend obsolète l'attente de la VSYNC.

Il existe une grosse différence entre la CPU calculée par une directive d'assembleur (afin de créer le code) et un outil qui calcule la CPU exacte d'un code déjà assemblé.
En effet, la CPU dépend de nombreux facteurs qu'un assembleur ne peut pas gérer (code auto-modifié, code généré, boucles, appels conditionnels, ...).

Il est possible de télécharger le source Z80A de cet outil ici :
http://logonsystem.fr/down/CalcCpuv4.asm

Pour l'utiliser il faut passer en paramètre une adresse de départ (dans HL) et une adresse de fin (dans DE) pour le code concerné. La fonction retourne dans BC le nombre de µs calculé. Il est ainsi simple de pouvoir calculer la compensation nécessaire sur une période de temps fixe donnée.

Afin de généraliser l'application de cet outil à plusieurs « portions » de code, on peut définir une structure pointée par IX et contenant les paramètres d'appel et de compensation:

```
CCPU_UNITE                  ; Compensation de CPU Unitaire
            ld l,(ix+0)     ; HL=adresse de début du code
            ld h,(ix+1)
            ld e,(ix+2)     ; DE=adresse de fin du code
            ld d,(ix+3)
            ;
            call CalcCPU    ; Calcul de la durée du code
            ;
            ld l,(ix+4)     ; HL=durée attendue du code
            ld h,(ix+5)
            or a
            sbc hl,bc       ; Dont on soustrait la durée calculée
            ex de,hl        ; DE=durée de compensation (en µs)
            ld l,(ix+6)     ; HL=adresse du code de compensation
            ld h,(ix+7)
            ld (hl),e       ; Dans lequel on stocke la durée de compensation
            inc hl
            ld (hl),d
            ret
```

Exemple de code utilisant cette fonction :

```
                ld ix,SCCPU_FRAME     ; Définition du code à compenser
                call CCPU_UNIT        ; Mise à jour du « ld de,0 » à FRAME_COMP
                ;
                call sync_vbl         ; Attente de la Vsync
FRAME_START                           ; 1ère µsec après la vsync (C0vs)
                ...
                votre code ici
                ...
FRAME_COMP      ld de,0               ; xxxx µs (durée compensatoire)
FRAME_END
                call wait_usec        ; 5 µs
                jp FRAME_START        ; 3 µs

SCCPU_FRAME     dw FRAME_START, FRAME_END
                dw 19968-8, FRAME_COMP+1
```

Dans cet exemple, on souhaite que **FRAME_START** soit appelé toutes les 19968 µs, quel que soit le code en temps fixe qui est saisi entre les labels **FRAME_START** et **FRAME_END**.

La fonction **CCPU_UNIT** va faire le calcul, puis soustraire le résultat trouvé de 19968-8 (les 8 µs correspondent à la durée du **call wait_usec** et du **jp FRAME_START**). On ne peut en effet pas « intégrer » l'attente de compensation dans le calcul puisque l'objectif du calcul est justement d'en définir la durée.

A l'issue de la fonction, la valeur de compensation a modifié l'opérande de « **ld de,0** ».

Dans cet exemple, l'utilisation de cette fonction est simplifié. Notamment il ne gère pas le cas ou le code dure plus longtemps que la durée maximale prévue afin de prévenir le programmeur que son code est trop long.

Par ailleurs, en mode de développement, il est possible de gérer la compensation au « **run** », mais prévoir un **gestionnaire de compensation** pour le code « livré » qui calcule toutes les compensations du projet. C'est ainsi que SHAKER est développé.

Il est possible de gérer **plusieurs zones de code** au sein d'un même frame, ayant **chacune leurs zones de compensation**. Il est possible d'encapsuler cette fonction dans un gestionnaire de compensation bien plus performant, capable notamment d'utiliser des compensations en cascade, en commençant par les fonctions de plus bas niveau. Des sous fonctions peuvent ainsi être compensées, alors qu'elle seront elle-même intégrées dans la compensation de fonctions de plus haut niveau.

Il ne faut cependant jamais oublier que le calculateur « simule » le fonctionnement du code, et met donc à jour la ram (données, code) lors de son calcul. Il faut donc penser à réinitialiser les données susceptibles d'avoir été modifiées au cours de la simulation. Dans le cadre d'un gestionnaire de compensation, cela nécessite de prévoir une fonction d'initialisation dans le cas où du code a été automodifié, à appeler entre chaque compensation des fonctions de sous niveaux.

24.5 LIBÉREZ LE TEMPS FIXE !

Dans un contexte de code « cyclique » ou « court », comme peuvent l'être des démos, il est possible de s'affranchir des règles de programmation en temps fixe à l'aide du calculateur CPU évoqué au chapitre précédent.

Pour une démo dont la durée serait de 4 mn avec des parties créées en « temps libre », il est envisageable de calculer pour chaque frame sa valeur de compensation.

Chaque minute nécessite 3000 valeurs de compensation. (50 frame/sec x 60 sec).

Afin d'éviter de définir 3000 valeurs 16 bits (soit 6000 octets), il est possible de redéfinir l'unité de temps. Si la durée maximale à compenser est de 19968µs, alors on peut définir 256 unités de CPU de 78 µs (19968/256). Enfin, ces valeurs peuvent également être compressées.

Le calcul dynamique de cette table n'est pas envisageable en l'état, car le calculateur simule le code et est donc plus lent qu'une exécution directe.

Il est possible de contourner ce problème en segmentant le code avec les différentes zones en temps fixe, afin de permettre d'assembler, tel un légo, le calcul des valeurs de la table des compensations avec les différentes briques déjà calculées.

24.6 DU TEMPS LIBRE AU TEMPS FIXE...

Transformer un programme écrit « librement » en temps fixe peut se révéler complexe.

Voici une méthode que j'ai utilisé pour transformer un player de musique AYC en temps fixe.
Le source est disponible ici : http://logonsystem.fr/down/TFixAYC.rar

L'idée générale est de calculer en temps réel la cpu consommée par le player de musique, et de compenser cette durée à la fin de l'exécution.

Ceci implique d'une part de définir des unités de temps mesurables dans le code, quitte à le transformer un peu pour adapter certaines sections aux unités de CPU disponibles.

Plus la quantité de registres Z80A disponibles alloués à cette tâche est importante, plus cela permet d'accroitre la précision de calcul.

Dans l'exemple, 5 registres sont utilisés pour mesurer la CPU :
IXL=UT 24µs / C=UT 16µs / IYH=UT 10 µs / IXH=UT 8µs/ IYL=UT 4µs

L'unité de temps minimale est de 4 µs, ce qui est idéal pour la compensation.

Ainsi pour une portion de code qui utilise 10µs, on va trouver un INC IYH, par exemple.

Cela implique de calculer la CPU **la plus longue** réalisée par le player selon la musique utilisée, afin de pouvoir compenser la CPU à partir de cette valeur. C'est pourquoi le player intègre une option qui permet de calculer préalablement la valeur de compensation à implémenter dans le source définitif (dans le source il s'agit de l'équivalence **CalcCpuMax**).

Une fois la durée maximale connue, il suffit de compenser cette valeur par unité de 4µs :

```
ccpu_wait04
                inc a                   ; 1 µs
                jr nz,ccpu_wait04       ; 3 µs
```

Le calcul temps réel de la CPU dégrade un peu la performance globale du player, mais cette dégradation est néanmoins atténuée par l'**obligation de compensation**.

Ainsi, pour les informations techniques, ce player est capable de gérer 14 registres sonore à partir d'un format standard non transformé. Le coût de la décompression avec des buffers 8 bits, ainsi que l'envoi d'un registre au générateur sonore est de 61.21 µsec.

24.7 PERDRE DU TEMPS…

Lorsqu'il est question d'attendre de manière précise, il est possible d'utiliser des séries de NOP de 1 µsec mais cela peut considérablement allonger le code présent en ram. Si l'objectif est de minimiser la taille d'une boucle en ram, il existe quelques instructions plus ou moins « neutres » pour les registres, qui occupent peu d'espace et permettent d'attendre plus longtemps que l'instruction NOP.

Attention aux interruptions et au contenu de la pile (ou registres) lorsque la neutralité des données est gérée avec des instructions associées à la pile.

Instruction (s)	Durée (en µsec)	Espace occupé (en octets)	Attention !
NOP	1	1	
CP (HL)	2	1	Modifie F
JR $+2	3	2	
INC HL + DEC HL	4	2	
INC (HL) + DEC (HL)	6	2	Modifie F
PUSH HL + POP HL	7	2	Modifie le contenu de la pile
EX (SP),HL + EX (SP),HL	12	2	

25 DURÉES INSTRUCTIONS Z80A SUR CPC

Le Z80A est interrompu par le GATE ARRAY via sa broche READY lorsque ce dernier lit les 2 octets de la ram adressée par le CRTC à chaque µ-seconde. Le Z80A teste l'état de sa broche WAIT selon la définition des différents cycles de l'instruction lorsqu'il a besoin d'accéder à la ram ou effectuer une entrée-sortie. La broche WAIT (N°24) du Z80A est mise à l'état bas pendant 3 cycles sur 4, provoquant un alignement de chaque accès mémoire lorsque le signal WAIT est de nouveau haut. Le tableau suivant décrit la durée des instructions du Z80A en µsecondes.

Instruction	µs	Size		Instruction	µs	Size
ADC A,(HL)	2	1		DI	1	1
ADC A,(IX/IY+d)	5	3		DJNZ	4/3	2
ADC A, A/B/C/D/E/H/L	1	1		EI	1	1
ADC A,HX/LX/HY/LY	2	2		EX (SP),HL	6	1
ADC A,d	2	2		EX (SP),IX/IY	7	2
ADC HL,BC/DE/HL/SP	4	2		EX AF,AF'	1	1
ADD A,(HL)	2	1		EX DE,HL	1	1
ADD A,(IX/IY+d)	5	3		EXX	1	1
ADD A, A/B/C/D/E/H/L	1	1		HALT	1	1
ADD A, HX/LX/HY/LY	2	2		IM m	2	2
ADD A,d	2	2		IN A/B/C/D/E/H/L,(C)	4	2
ADD HL,BC/DE/HL/SP	3	1		IN A,(d)	3	2
ADD IX,BC/DE/HL/SP	4	2		IN F	4	2
ADD IY,BC/DE/HL/SP	4	2		INC (HL)	3	1
AND A,(HL)	2	1		INC (IX/IY+d)	6	3
AND A,(IX/IY+d)	5	3		INC A/B/C/D/E/H/L	1	1
AND A, A/B/C/D/E/H/L	1	1		INC HX/LX/HY/LY	2	2
AND A, HX/LX/HY/LY	2	2		INC BC/DE/HL/SP	2	1
AND A,d	2	2		INC IX/IY	3	2
BIT x,(HL)	3	2		IND	5	2
BIT x,(IX/IY+d)	6	4		INDR	6/5	2
BIT x, A/B/C/D/E/H/L	2	2		INI	5	2
CALL cond,aa	5/3	3		INIR	6/5	2
CALL aa	5	3		JP aa	3	3
CCF	1	1		JP cond,aa	3	3
CP A, (HL)	2	1		JP HL	1	1
CP A, (IX/IY+d)	5	3		JP IX/IY	2	2
CP A, A/B/C/D/E/H/L	1	1		JR a	3	2
CP A, HX/LX/HY/LY	2	2		JR cond,a	3/2	2
CP A,d	2	2		LD (BC/DE),A	2	1
CPD	4	2		LD (HL),A/B/C/D/E/H/L	2	1
CPDR	6/4	2		LD (HL),d	3	2
CPIR	6/4	2		LD (IX/IY+d), A/B/C/D/E/H/L	5	3
CPI	4	2		LD (IX/IY+d),d'	6	4
CPL	1	1		LD (aa),A	4	3
DAA	1	1		LD (aa),BC/DE/SP/IX/IY	6	4
DEC (HL)	3	1		LD (aa),HL	5	3
DEC (IX/IY+d)	6	3		LD A,(BC/DE)	2	1
DEC A/B/C/D/E/H/L	1	1		LD A/B/C/D/E/H/L,(HL)	2	1
DEC HX/LX/HY/LY	2	2		LD A/B/C/D/E/H/L,(IX/IY+d)	5	3
DEC BC/DE/HL/SP	2	1		LD A,(aa)	4	3
DEC IX/IY	3	2		LD A/B/C/D/E/H/L, A/B/C/D/E/H/L	1	1

Instruction	µs	Size	I/O	Instruction	µs	Size
LD HX/LX,A/B/C/D/E/HX/LX	2	3		RR (HL)	4	2
LD HY/LY,A/B/C/D/E/HY/LY	2	3		RR (IX/IY+d)	7	4
LD BC/DE/HL/SP,dd	3	3		RR (IX/IY+d),A/B/C/D/E/H/L	7	4
LD IX/IY,dd	4	4		RR A/B/C/D/E/H/L	2	2
LD SP,IX/IY	3	2		RRA	1	1
LD SP,HL	2	1		RRC (HL)	4	2
LD HX/LX/HY/LY,d	3	3		RRC (IX/IY+d)	7	4
LD BC/DE/HL/SP/IX/IY,(aa)	6	4		RRC(IX/IY+d),A/B/C/D/E/H/L	7	4
LD HL,(aa)	5	3		RRC A/B/C/D/E/H/L	2	2
LD I,A / LD A,I	3	2		RRCA	1	1
LD R,A / LD R,A	3	2		RRD	5	2
LDD	5	2		RST 0/8/10h/18h/28h/30h/38h	4	1
LDDR	6/5	2		SBC A,d	2	2
LDI	5	2		SBC A,(HL)	2	1
LDIR	6/5	2		SBC A,(IX/IY+d)	5	3
NEG	2	2		SBC A, A/B/C/D/E/H/L	1	1
NOP	1	1		SBC A, HX/LX/HY/LY	2	2
OR A,(HL)	2	1		SBC HL,BC/DE/HL/SP	4	2
OR A,(IX/IY+d)	5	3		SCF	1	1
OR A, A/B/C/D/E/H/L	1	1		SET x,(HL)	4	2
OR A, HX/LX/HY/LY	2	2		SET x,(IX/IY+d)	7	4
OR A,d	2	2		SET x,(IX/IY+d),A/B/C/D/E/H/L	7	4
OTDR	6/5	2	5*	SET x,A/B/C/D/E/H/L	2	2
OTIR	6/5	2	5*	SLA (HL)	4	2
OUT (C),A/B/C/D/E/H/L	4	2	3	SLA (IX/IY+d)	7	4
OUT (C),0	4	2	3	SLA (IX/IY+d),A/B/C/D/E/H/L	7	4
OUT (d),A	3	2	3	SLA A/B/C/D/E/H/L	2	2
OUTD	5	2	5*	SLL (HL)	4	2
OUTI	5	2	5*	SLL (IX/IY+d)	7	4
POP AF/BC/DE/HL	3	1		SLL (IX/IY+d),A/B/C/D/E/H/L	7	4
POP IX/IY	4	2		SLL A/B/C/D/E/H/L	2	2
PUSH AF/BC/DE/HL	4	1		SRA (HL)	4	2
PUSH IX/IY	5	2		SRA (IX/IY+d)	7	4
RES x,(HL)	4	2		SRA A/B/C/D/E/H/L	2	2
RES x,(IX/IY+d)	7	4		SRL (HL)	4	2
RES x,(IX/IY+d),A/B/C/D/E/H/L	7	4		SRL (IX/IY+d)	7	4
RES x, A/B/C/D/E/H/L	2	2		SRL (IX/IY+d),A/B/C/D/E/H/L	7	4
RET	3	1		SRL A/B/C/D/E/H/L	2	2
RET cond	4/2	1		SUB A,(HL)	2	1
RETI	4	2		SUB A,(IX/IY+d)	5	3
RETN	4	2		SUB A,A/B/C/D/E/H/L	1	1
RL (HL)	4	2		SUB A, HX/LX/HY/LY	2	2
RL (IX/IY+d)	7	4		SUB A,d	2	2
RL (IX/IY+d),A/B/C/D/E/H/L	7	4		XOR A,(HL)	2	1
RL A/B/C/D/E/H/L	2	2		XOR A,(IX/IY+d)	5	3
RLA	1	1		XOR A,A/B/C/D/E/H/L	1	1
RLC (HL)	4	2		XOR A, HX/LX/HY/LY	2	2
RLC (IX/IY+d)	7	4		XOR A,d	2	2
RLC (IX/IY+d),A/B/C/D/E/H/L	7	4				
RLC A/B/C/D/E/H/L	2	2		x=[0..7] d=[0..ff] m=[0..2]		
RLCA	1	1		aa=[0..ffff] a=[0..ff]		
RLD	5	2		* some exceptions exist		

26 INTERRUPTIONS

26.1 GÉNÉRALITÉS

Les interruptions sont générées sur CPC par le GATE ARRAY, selon des paramètres définis par le CRTC. Le GATE ARRAY dispose à cet effet d'un compteur dont l'objectif est de compter de 0 à 51 avant de repasser à 0, qui est souvent appelé **R52**.

Ce compteur est incrémenté à la fin d'une HSYNC qui est produite selon les paramètres définis par les registres R0, R2 et R3 du CRTC. Il n'existe pas de taille minimale de HSYNC pour que R52 soit géré. Il est géré même si R3=1. Cependant, à cause du mécanisme de protection de réentrance des HSYNC, le délai minimum entre 2 HSYNC est de 2 µsec (car il faut au minimum 1 µsec entre 2 HSYNC). Ce qui implique que le compteur peut boucler en 104 µsec minimum.
À noter que le mécanisme de prévention de réentrance est buggé sur les CRTC 1 à 4, ce qui permet de créer une HSYNC infinie selon la méthode utilisée (par exemple R0=R2=0 et R3=1).

En fonctionnement "standard", avec une HSYNC programmée toutes les 64 µsec, 1 interruption peut avoir lieu toutes les 3328 µsec (300 Hz).
Sur un CPC au standard Européen, cela représente 6 interruptions par frame :
6 x 52 lignes = 312 lignes.

26.2 GESTION DU COMPTEUR R52

Il existe plusieurs particularités de gestion de la valeur du compteur **R52** par le GATE ARRAY.

Il peut repasser à 0 :

- Lorsqu'il dépasse 51.
- En positionnant à 1 le bit 4 du registre RMR du GATE ARRAY.
- À la fin de la 2ème HSYNC après le début de la VSYNC.

Le bit 5 du compteur R52 passe à 0 :

- Lorsqu'une interruption était armée (en attente) et qu'elle est autorisée alors que **R52** avait continué à évoluer.

Remarque : Si la demande de remise à 0 du compteur R52 est réalisée via la fonction RMR du GATE ARRAY et que cette demande a lieu sur la dernière µseconde de la HSYNC (C0=R2+R3-1), alors c'est la mise à 0 qui est prioritaire sur l'incrémentation. Si la demande intervient 1 µseconde avant la fin de la HSYNC (C0=R2+R3-2), alors R52 est remis à 0, puis incrémenté sur R2+R3-1.

26.3 CONDITIONS DE DÉCLENCHEMENT

Pour qu'une interruption puisse se produire, il faut qu'elle soit armée.

26.3.1 ARMEMENT SUR R52=0

Le GATE ARRAY envoie une demande d'interruption lorsque R52=0.

- Si les interruptions étaient autorisées au moment de la demande, alors l'interruption a lieu et le bit 5 de R52 est mis à 0 (ce qui ne sert pas à grand chose car il valait déjà 0).

- Si les interruptions ne sont pas autorisées, le compteur continue à s'incrémenter, mais l'interruption reste armée (le GATE ARRAY maintient alors son signal INT). Lorsque l'interruption est autorisée (via l'instruction EI du Z80A) alors :
    - Une interruption se produit **<u>après</u> l'instruction qui suit le EI**. Un HALT suivant un EI dans cette circonstance sera considéré comme 1 NOP.
    - Le bit 5 du compteur R52 est remis à 0 à la fin « réelle » de cette instruction. Si le bit 5 du compteur avait atteint 1 (R52>=32), alors cela revient à "retirer" 32 lignes du compteur courant.
      Cela a pour effet de décaler le moment ou la prochaine interruption pourra se produire.
      Ce mécanisme empêche que moins de 20 lignes séparent 2 interruptions

26.3.2 ARMEMENT PENDANT LA VSYNC

Deux HSYNC après le début de la VSYNC :

Une interruption est demandée par le GATE ARRAY au Z80A uniquement si le bit 5 de R52 vaut 1.
En général, R52 « revient » à 0 lorsque 312 HSYNC ont eu lieu depuis la dernière VSYNC.
Mais si le frame n'est pas formaté correctement, la valeur de R52 peut alors être différente de 0.
Comme décrit dans le chapitre précédent, ce mécanisme rudimentaire empêche que deux interruptions puissent se produire dans un intervalle trop rapproché.

R52 est mis à 0 sans condition.

26.3.3 Z80A ET INTERRUPTIONS

L'instruction Z80A **EI** sert à autoriser une interruption à se produire dès que cette dernière est armée. Lorsqu'une interruption se produit, les interruptions sont désactivées jusqu'à ce qu'un nouvel EI survienne. Les flag IFF du Z80A sont positionnés à 1, ce qui a le même effet qu'un **DI**.
Le compteur R52 continue cependant d'évoluer.
Si R52 repasse à 0, une nouvelle interruption est alors en attente, et va se produire dès que les interruptions seront autorisées de nouveau. Une seule interruption peut être en attente.
Afin d'éviter des problèmes de réentrance dans un code d'interruption, les concepteurs du Z80A ont imposé qu'une nouvelle interruption ne puisse pas survenir sur l'instruction suivant le EI.
Ceci a été fait pour permettre aux instructions **RET, RETI, RETN** de s'exécuter juste derrière l'instruction EI, afin d'éviter que la pile déborde.

Une séquence répétitive de l'instruction EI ne permettra pas la survenue d'une interruption avant la fin de cette séquence, chaque EI différant l'interruption.

L'instruction Z80A **HALT** permet d'attendre qu'une interruption survienne en répétant indéfiniment des NOP de 1 µsec. Si les interruptions ne sont pas autorisées lorsque cette instruction s'exécute, le Z80A reste bloqué sur cette instruction. À ma connaissance, peu de jeux ont utilisé l'instruction HALT. On peut cependant noter que le jeu Trailblazer (edité par Gremlin Graphics en 1986) cale ses « splitraster » avec un HALT. En effet cette instruction est intéressante dans la mesure où elle permet de caler une interruption à la microseconde près.
En effet, une interruption ne peut pas couper une instruction (sauf une instruction répétitive comme LDIR ou OTIR). Ainsi, si l'instruction dure plusieurs microsecondes, l'interruption se produira après la fin de l'instruction.

26.4 INTERRUPTIONS MODE 1

La plupart des codes écrits pour le CPC AMSTRAD "OLD" utilisent le mode IM 1 du Z80A.
On passe dans ce mode grâce à l'instruction **IM 1**

Dans ce mode, lorsqu'une interruption survient, le code est interrompu avec une instruction équivalente à RST #38.

L'adresse suivant l'instruction interrompue (ou l'adresse de l'instruction si il s'agit d'une instruction répétitive) est mise sur la pile et PC=#38.

L'instruction Z80A RST #38 dure 4 µsec lorsqu'elle est appelée par du code.
Lorsqu'une interruption se produit, l'appel en #38 dure 5 µsec.
[pour le tester, il suffit de comparer le temps pris par un RST #38 et le temps pris par une interruption avec un code en temps fixe sur 19968 nop]

26.5 INTERRUPTIONS MODE 2

Le mode **IM 2**, aussi appelé mode vectorisé, est prévu pour permettre à plusieurs périphériques de générer des interruptions, via une table de pointeurs sur des routines d'interruption.

L'adresse de la table est définie, pour le poids fort de l'adresse, par le **registre I du Z80A**.
Le poids faible de l'adresse est normalement défini par un des 128 périphériques possibles en indiquant son numéro sur les 7 bits de poids fort, le bit 0 valant 0.

Sur CPC AMSTRAD "OLD", la valeur de poids faible de l'adresse est indéterminée (valeur de Haute Impédance) et peut varier d'un CPC à l'autre.
[Je n'ai pas vérifié si cette valeur peut varier tant que le CPC est sous tension, mais l'intérêt en gain de ram est modéré et c'est une chose à éviter]

Dans la perspective de déplacer l'adresse d'interruption ailleurs qu'en #38, il est néanmoins possible d'utiliser ce mode moyennant quelques précautions :

Il faut créer une table de vecteurs d'interruption contenant 257 octets de même valeur.
En effet, le bit 0 de l'adresse de la table sélectionné étant imprévisible, cela peut amener le Z80A à aller lire son vecteur sur le 256ème et 257ème octet de la table.

Dans le cas ou le bit 0 vaudrait 0, le poids fort et faible du vecteur lu dans la table serait inversé par rapport à un vecteur lu sur une table ou le bit 0 vaut 1.

Il est donc conseillé de créer un vecteur d'interruption ou le poids fort du vecteur est égal à son poids faible.

Exemple : Table de vecteurs créée en #2000, qui contient entre #2000 et #2100, 257 fois la valeur #CA. L'interruption se produisant alors en #CACA (prout!)
Le jeu « ***The Great Escape*** », édité par Océan Software en 1986, a utilisé ce mode pour libérer la première page de 256 octets. La table de vecteurs débute en #100 (I=1) et occupe 257 octets avec la valeur #BC (pour un vecteur situé en #BCBC)

Lorsqu'une interruption se produit, l'appel du pointeur situé dans la table dure 7 µsec.

26.6 CRTC & INTERRUPTIONS...

26.6.1 GÉNÉRALITÉS

Lorsque la condition interne C0=R2 est remplie, une HSYNC débute.

Rappelons une nouvelle fois que sur les CRTC 0, 1, 2, la HSYNC débute visuellement 1 µsec environ avant l'affichage du caractère CRTC correspondant.

L'affichage des caractères par le GATE ARRAY sur ces CRTC recommence à partir de C0 Disp=R2+R3-1 (sauf pour R3=0 pour les CRTC 0 et 1).

La mesure du moment où débute l'interruption est réalisée en considérant C0vs à partir du signal VSYNC renvoyé par le PPI.

Sur les CRTC 3 et 4, la HSYNC débute au début de l'affichage par le GATE ARRAY du caractère CRTC correspondant à C0=R2.

En règle générale, une interruption débute toujours 1 µsec après la fin de la HSYNC quel que soit le CRTC.

Étant donné que la HSYNC débute 1 µsec plus tôt que prévu sur un CPC sans ASIC (CRTC 0, 1, 2) il existe donc un décalage de 1 µsec avec une interruption sur un CPC avec ASIC (CRTC 3, 4).

Avec une même programmation de R2 et R3, une interruption se produit 1 µsec plus tard sur un CRTC 3 ou 4 que sur les autres CRTC, car l'ASIC gère une HSYNC synchrone avec l'affichage.

La HSYNC générée par un ASIC est plus conforme à la gestion de l'affichage.

Les schémas suivants décrivent cette gestion selon les CRTC.

26.6.2 CRTC 0, 1, 2

R3=14					R2																									
C0 from Vs	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	
C0 from GA	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	
				C3:	1	2	3	4	5	6	7	8	9	10	11	12	13	14		Code interrupted 15 µsec after C0vs=R2										

R3=8					R2																									
C0 from Vs	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	
C0 from GA	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	
				C3:	1	2	3	4	5	6	7	8		Code interrupted 9 µsec after C0vs=R2																

R3=1					R2																									
C0 from Vs	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	
C0 from GA	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	
				C3:	1		Code interrupted 2 µsec after C0vs=R2																							

26.6.3 CRTC 0, 1

R3=0					R2																									
C0 from Vs	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	
C0 from GA	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	
					No interruption																									

26.6.4 CRTC 2

R3=0					R2																								
C0 from Vs	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
C0 from GA	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
C3:					1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	Code interrupted 17 µsec after C0vs=R2								

26.6.5 CRTC 3, 4

R3=14					R2																								
C0 from Vs	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
C0 from GA	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
C3:					1	2	3	4	5	6	7	8	9	10	11	12	13	14	Code interrupted 16 µsec after C0vs=R2										

R3=8					R2																								
C0 from Vs	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
C0 from GA	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
C3:					1	2	3	4	5	6	7	8	Code interrupted 10 µsec after C0vs=R2																

R3=1					R2																								
C0 from Vs	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
C0 from GA	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
C3:					1	Code interrupted 3 µsec after C0vs=R2																							

R3=0					R2																								
C0 from Vs	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
C0 from GA	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
C3:					1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	Code interrupted 18 µsec after C0vs=R2								

Remarque :

Dans la mesure où il est possible de générer des HBL "non actives" pour le moniteur (lorsque R3<=2), cela implique la possibilité de générer des interruptions dans la zone visible.

26.6.6 MISE EN PERSPECTIVE

Le schéma ci-dessous permet une mise en perspective d'une interruption programmée dans les mêmes conditions selon les différents CRTC.

Il permet d'appréhender les mécanismes en œuvre à partir du caractère **C0vs** de référence, compte tenu des différents délais mis en œuvre.

Lorsqu'il est question de compatibilité entre les CRTC, il faut donc vérifier les différences relatives aux délais de prises en comptes des registres et l'effet de certaines valeurs, notamment si ces registres sont modifiés lorsqu'une HSYNC est en cours.

	R3=14					R2																							
	C0 from Vs	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
CRTC 0, 1, 2	C0 from GA	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
	C3:					1	2	3	4	5	6	7	8	9	10	11	12	13	14		Code interrupted 15 µsec after C0vs=R2									
CRTC 3, 4	C0 from GA	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
	C3:					1	2	3	4	5	6	7	8	9	10	11	12	13	14		Code interrupted 16 µsec after C0vs=R2									
						1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16									

26.7 MÉNAGE À TROIS...

26.7.1 TRIPOTAGE DE R52...

Nous l'avons vu dans les chapitres précédents, le GATE ARRAY « demande » une interruption au Z80A selon différentes conditions.

Il se charge également de gérer son compteur interne R52 en le remettant à 0, en l'incrémentant ou en éliminant sauvagement son bit 5.

L'incrémentation de R52 dépend du CRTC qui signale au GATE ARRAY la fin de la HSYNC.

La mise à 0 du bit 5 de R52 dépend du Z80A, qui informe le GATE ARRAY qu'une interruption a lieu. C'est après l'instruction qui suit le EI d'une interruption en attente que le GATE ARRAY va trucider le bit 5 (le remettre à 0).

Si R52 vaut 31, et que le GATE ARRAY reçoit une fin de HSYNC du CRTC pour incrémenter R52, mais qu'au même moment, il reçoit l'ordre d'éliminer le bit 5, que se passe-t-il ?
La réponse ne va pas plaire aux auteurs d'émulateurs au NOP.

Selon la longueur réelle d'exécution de l'instruction qui suit l'instruction EI, la remise à 0 du bit 5 peut mettre plus ou moins de temps. La conséquence est que R52 peut être incrémenté avant ou après la mise à 0 du bit 5.

On peut donc avoir les deux situations suivantes:

- R52 passe de 31 à 32, puis son bit 5 est éliminé, et R52 passe à 0.
- Le bit 5 de R52=31 est éliminé (ce qui n'a aucun effet) et R52 passe à 32.

Dans la première situation, la prochaine interruption ne pourra pas avoir lieu avant 52 lignes.
Dans la seconde situation, la prochaine interruption ne pourra pas avoir lieu avant 20 lignes.

Concrètement, le GATE ARRAY active le signal INT pour indiquer au Z80A sa demande d'interruption, qui l'accepte selon l'état d'un flag interne du Z80 positionné par EI/DI (ce flag s'appelle IFF1, pour « Interrupt Flip-Flop N°1 »).

Si l'interruption est acceptée, le Z80A active ses signaux IORQ et M1, également connectés au GATE ARRAY. À noter que la fin du signal M1 lors d'une interruption arrive après les cycles d'attente TWait du Z80A, autrement dit après l'exécution de l'instruction qui suit le EI.

26.7.2 FIABILITE DES INTERRUPTIONS

Le GATE ARRAY est cadencé à 16 Mhz, et il est le chef d'orchestre des autres circuits de la machine. Il cadence notamment le Z80A à 4 Mhz.

Le GATE ARRAY gère certains signaux du CRTC, et notamment son signal HSYNC. Lorsque l'état de ce signal change, il permet au GATE ARRAY de gérer son compteur R52 afin de positionner le signal INT du Z80A lorsque R52 a atteint sa limite. Ce traitement est à priori réalisé sur les premiers cycles de traitement du GATE ARRAY.

Si les interruptions sont autorisées et que le signal INT est activé, le Z80A va traiter une interruption après la fin de l'instruction en cours d'exécution.

Malheureusement, les CRTC ne sont pas des modèles de fiabilité concernant la gestion de leur signal HSYNC. En particulier le CRTC 1 pour lequel des disparités existent entre les modèles du même type et de types différents. Ainsi, à programmations équivalentes, la fin du signal HSYNC peut survenir plus ou moins tard sur une échelle au 1/16ème de MHz. Même si le bus du CRTC est à 1 Mhz, la programmation directe des registres ou le traitement interne des signaux n'est pas forcément aligné sur cette fréquence.

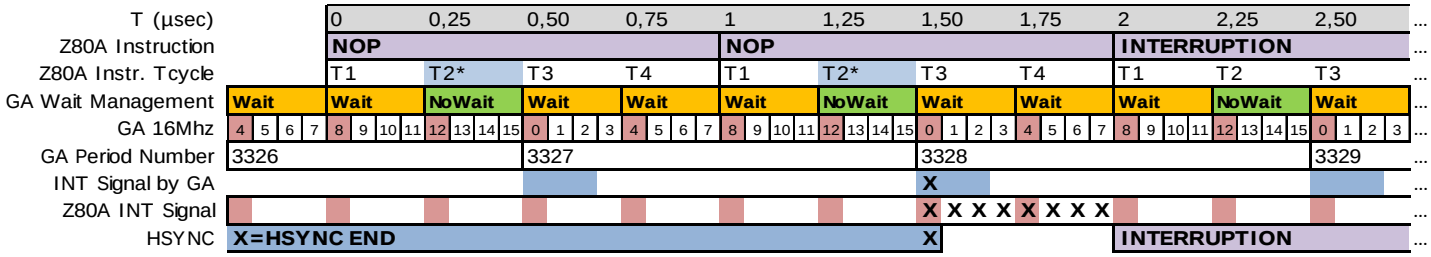
Pour rappel, le GATE ARRAY positionne également le signal WAIT du Z80A pendant ¾ de sa fréquence (soit 12/16 Mhz) et oblige le Z80A à ajouter des cycles d'attente sur les cycles prévus pour ça au sein des cycles M du répertoire d'instructions du Z80A. Le Z80A est « libre » du motif imposé par le GATE ARRAY pendant seulement 4/16ème de Mhz (0.25 µsec, 1 cycle T). Ce motif provoque une déformation des instructions. (voir chapitre 4.4.2).

Si le CRTC active la fin de HSYNC durant le dernier cycle T d'une instruction (0.25 µsec), la fenêtre est très courte pour permettre au GATE ARRAY d'activer le signal INT assez tôt pour que le Z80A en tienne compte. Si la fin de HSYNC arrive trop tard, alors le Z80A n'est pas prévenu à temps, et l'interruption n'a pas lieu. Selon le CRTC, le Z80A peut donc traiter une instruction complémentaire avant de générer son interruption.

La logique de cadencement du GATE ARRAY pour aligner les cycles des instructions du Z80A permet d'éviter les conséquences de ces fins de HSYNC discordantes pour certaines instructions dont les premiers cycles du GATE ARRAY ne sont pas sur le dernier cycle T de l'instruction du Z80A.

C'est le cas de l'instruction NOP (et par extension de son fournisseur officiel, l'instruction HALT).

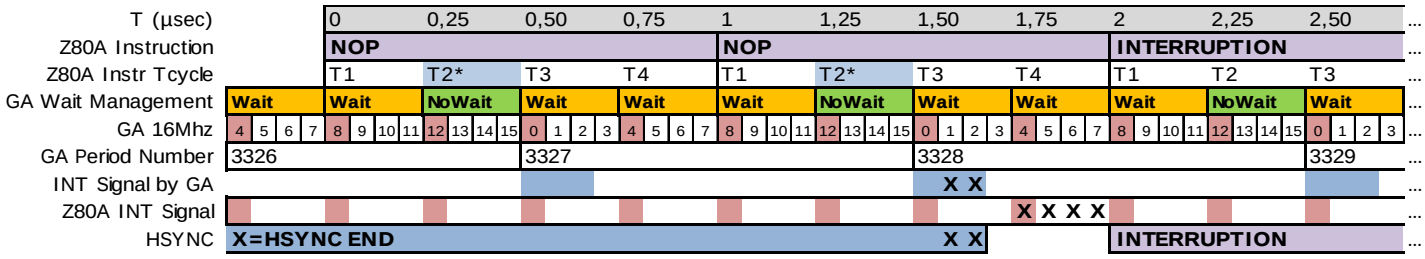
<u>La fin de HSYNC se produit avant ou sur le premier cycle de 1/16 Mhz qui débute sous T3 :</u>



Le GATE ARRAY était sur son dernier cycle avant la fin de HSYNC. Avec R0=63, 64 µsec x 52 lignes=3328 cycles se sont écoulés depuis la dernière fin de HSYNC.

Le signal INT du Z80A est positionné au niveau de T3 et une interruption aura lieu à la fin du NOP.

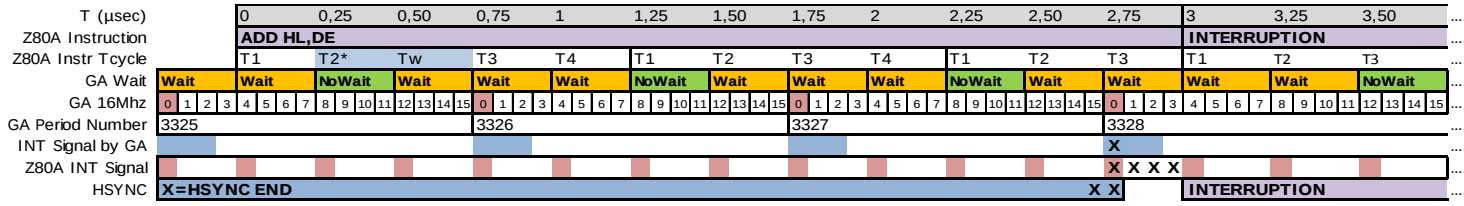
La fin de HSYNC se produit après les premiers cycles de 1/16 Mhz qui débute sous T3 :



La fin de la HSYNC est arrivée trop tard. Le GA a traité le signal, mais le Z80A en sera informé sur T4. L'interruption se produira quand même à la fin de l'instruction NOP.

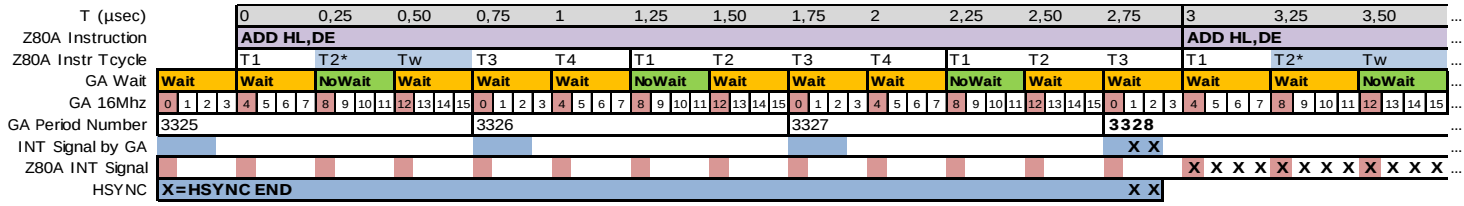
Malheureusement, il existe beaucoup d'instructions dont le dernier cycle T se retrouve positionné en face des premiers cycles du GATE ARRAY. Un simple retard de quelques 1/16ème de Mhz suffit alors à saborder une interruption.

Exemple avec deux instructions ADD HL,DE qui se suivent :
La fin de HSYNC se produit avant ou sur le premier cycle de 1/16 Mhz qui débute sous T3 :



Le GATE ARRAY a eu le temps de tenir compte de la fin de HSYNC pour positionner le signal INT du Z80A, pris en compte au début de T3. Le Z80A sait qu'il va devoir générer une interruption et non lire la prochaine instruction.

La fin de HSYNC se produit après les premiers cycles de 1/16 Mhz qui débute sous T3 :



Le GATE ARRAY n'a pas eu le temps de tenir compte de la fin de HSYNC pour que le signal INT soit transmis au Z80A en T3. Du coup, le Z80A ne va pas pouvoir en tenir compte et il va traiter l'instruction suivante, et générera son interruption après cette 2ème instruction.

Une instruction comme SET n,(IX+n') dure officiellement 23 Cycles-T sans être allongée. Sur CPC l'alignement provoqué par le GATE ARRAY allonge cette instruction à 27 ou 28 Cycles-T.
Si une interruption est manquée, elle se produira alors 7 µsec plus tard selon le CRTC.

Il n'a pas été noté de différences entre différents CRTC de type 0, ce qui a conduit à prendre ce modèle comme référent. Les CRTC 2, 3 et 4 donnent en en général des résultats identiques au CRTC 0, à quelques petites différences près.

Le CRTC 1 présente des différences significatives, même entre différents type 1.
Dans un 464, par exemple, il peut se comporter comme un CRTC 0.

Le GATE ARRAY 40008 peut ici avoir une incidence étant donné qu'il s'agit d'un problème d'échange entre trois circuits et qu'il existe des différences de l'ordre de 1/16MHz entre certains modèles de GATE ARRAY.

Cependant, des tests ont été réalisés sur des 6128 Multi-CRTC, qui démontrent ces différences par rapport au GATE ARRAY 40010, constatées par ailleurs sur des machines non modifiées.

En conclusion, il est hasardeux de se fier à la position temporelle d'une interruption dès lors qu'elle peut se produire sur la toute fin d'une instruction.

27 IDENTIFICATION CRTC

Il existe une pléthore de méthodes permettant d'identifier un CRTC, la plupart avec du code, d'autres uniquement visuellement.

Certaines méthodes impliquent parfois une désynchronisation du frame sur l'écran et sont donc pour cette raison moins bien considérées par certains puristes.

Il est aussi possible d'identifier un CRTC en étant capable d'identifier la machine, comme c'est le cas avec le CPC+, qui présente des différences de gestion sur d'autres circuits (et possède, le cas échéant, une page de fonctions étendues).

27.1.1 VIA LE DEBORDEMENT DE C4 ET/OU C9

Il est possible de faire déborder C4 dans de nombreuses situations différentes selon les CRTC, que ce soit en programmant R0, R9, R4, R7 ou même R8.
En général, sur CRTC 0, par exemple, C4 est incrémenté par défaut.
En gestion additionnelle, la valeur de C4 dépasse la valeur programmée dans R4.
Sur un CRTC 0, ce dépassement aura lieu 1 fois.
Sur un CRTC 1 ou 2, ce dépassement aura lieu plusieurs fois selon le contenu de R9 et R5.
Sur un CRTC 3 ou 4, il n'y aura pas de dépassement.
Si C4 atteint R7, alors une VSYNC peut se produire.

Exemple : Ecran de 312 lignes tel que R4=36, R9=7, R5=16
312=((36+1)x(7+1))+16
Sur un CRTC 3 ou 4, si R7>36, la VSYNC ne se produit plus.
Sur un CRTC 0, si R7>37, la VSYNC ne se produit plus.
Sur un CRTC 1 ou 2, si R7>39, la VSYNC ne se produit plus.

27.1.2 VIA LA GESTION VSYNC DURANT LA HSYNC

Sur un CRTC 2, positionner R2 et R3 de manière à ce que la VSYNC survienne durant la période de la HSYNC « désactive » la VSYNC pour le reste de la période C4=R7.
Il suffit donc de placer R2+R3 de manière à ce qu'il dépasse la valeur de R0.

27.1.3 VIA LA PRISE EN COMPTE DE LA VSYNC

Lorsque la VSYNC est déclenchée en positionnant R7 avec C4, lorsque C0>0, le compteur de ligne de la VSYNC est différent selon les CRTC. Il part de 0 sur un CRTC 0, et de 1 sur un CRTC 1, alors qu'aucune VSYNC ne survient dans cette condition sur un CRTC 3 ou 4.

27.1.4 VIA LA LONGUEUR DE LA VSYNC

Seuls les CRTC 0, 3 et 4 permettent de gérer la longueur de la VSYNC.
Une VSYNC sur CRTC 1 et 2, qui démarre lorsque R7 était programmé avant que C4=R7, dure 16 lignes.

27.1.5 VIA LA LONGUEUR DE LA HSYNC

Sur les CRTC 2, 3 et 4, la HSYNC dure 16 µsec lorsque R3=0.
Sur les CRTC 0 et 1, il n'y a pas de HSYNC lorsque R3=0.
Il n'y a donc pas d'interruption, ce qui est une des méthodes permettant de tester les conséquences de la valeur 0 sur R3.

27.1.6 VIA DU BORDER, VISUELLEMENT

- Un CRTC 2 dont C0=0 pendant la HSYNC ne peut plus désactiver son BORDER.
- Un CRTC 0 et 2, dont R6=0 est en conflit lorsque C4=C9=0, alterne des octets de fond et de BORDER (à minima sur la première ligne sans programmation adéquate).
- Un CRTC 0 et 2 va créer un octets de BORDER lorsque C0 atteint R0 (et R1>R0).
- Les CRTC 0, 3 et 4 peuvent désactiver le BORDER (ou ajouter des délais sur la prise en compte du BORDER) avec la fonction SKEW BIT de R8, alolrs que cette fonction n'existe pas sur les CRTC 1 et 2.

27.1.7 VIA LE MODE INTERLACE

Les méthodes de comptage de C4 sont différentes selon les CRTC.
Sur CRTC 2, le comptage de C4 n'est pas affecté.
Étant le seul, il est donc détectable une fois les autres identifiés.
Sur les CRTC 0 et 1, avec R9 programmé respectivement avec 6 et 7 sans avoir touché à R7, la VSYNC se produit 2 fois plus vite, puisque C4=R7 avec des caractères de 4 lignes au lieu de 8.

27.1.8 VIA LE REGISTRE DE STATUS &BE00

Uniquement sur le CRTC 1, il est possible de tester le passage du bit 6 au moment opportun.
Sur CRTC 3 et 4, le port en &BE00 se comporte comme celui en &BF00 (voir ci-après).

27.1.9 VIA LE REGISTRE DE LECTURE &BF00

Sur CRTC 0, il est possible de lire R12 et R13, ainsi que R14/R15 (curseur) et R15/R17 (stylo optique). La valeur du numéro de registre est tronquée à 5 bits. Ainsi, sélectionner le registre 108 revient à sélectionner le registre 12.
Sur CRTC 1, ce registre renvoie 0 sur tous les registres, sauf pour le registre 31 (et tous les registres dont les bits 0 à 4 sont à 1). Les valeurs 255 et 127 ont été observées.
Sur CRTC 2, ce registre permet de lire les registres R16 et R17. La valeur 0 est renvoyée sur tous les autres numéros de registres. La valeur du numéro de registre est tronquée à 5 bits.
Sur CRTC 3 et 4, il est possible de lire, dans l'ordre, les registres R16, R17, R10, R11, R12, R13, R14 et R15. Le numéro de registre lu est cependant issu d'une table de 8 registres sur 3 bits (voir chapitre 21.2.3). Lire R4 ou R20 revient à lire R12.
Sur CRTC 3 et 4, la lecture de R10 et R11 (ou équivalence pour les 3 premiers bits) renvoie des valeurs internes à l'ASIC (voir chapitre 21.3.4).

27.1.10 VIA LES REGISTRE DE STATUS R10/R11

Uniquement sur les CRTC 3 et 4, il est possible de tester le passage de nombreux bits au moment opportun, comme par exemple le bit 0 du status 1 qui vaut 1 lorsque C0=R0 (0 sinon). Par ailleurs (et sous réserve de tests complémentaires), le bit 3 du status 2 devrait permettre de faire la différence entre le CRTC 3 et le CRTC 4.

28 IDENTIFICATION CPC

Il est possible d'identifier des modèles de CPC à partir de différences qui ne sont pas liées au CRTC ou au contenu de la ROM.
C'est le cas pour les 2 machines qui disposent d'ASIC :

CPC PLUS	ASIC 40489	CRTC 3
CPC LOWCOST	ASIC 40226	CRTC 4

28.1 MÉTHODES D'IDENTIFICATION

28.1.1 ACTIVATION DES FONCTIONS ÉTENDUES

CPC PLUS : Activation via une séquence de délockage envoyée au CRTC
CPC LOWCOST : Pas de fonctions étendues (sans la séquence secrète).
AUTRES CPC : Pas de fonctions étendues.

28.1.2 BUG PPI PORT C

CPC PLUS : Lorsque le registre de commande est utilisé pour configurer le port C, les bits de ce port sont remis à 0. L'ASIC émule mal le PPI et ne remet pas ces bits à 0. Très génant pour la compatibilité des routines clavier entre le CPC PLUS et anciens modèles.
CPC LOWCOST: Le PPI 8255 n'est pas émulé par l'ASIC 40226. Ces CPC disposent d'un PPI et se comportent donc comme les CPC de première génération.
AUTRES CPC : Les bits du port C sont remis normalement à 0 par le registre de commande.

28.1.3 BUG PPI PORT B

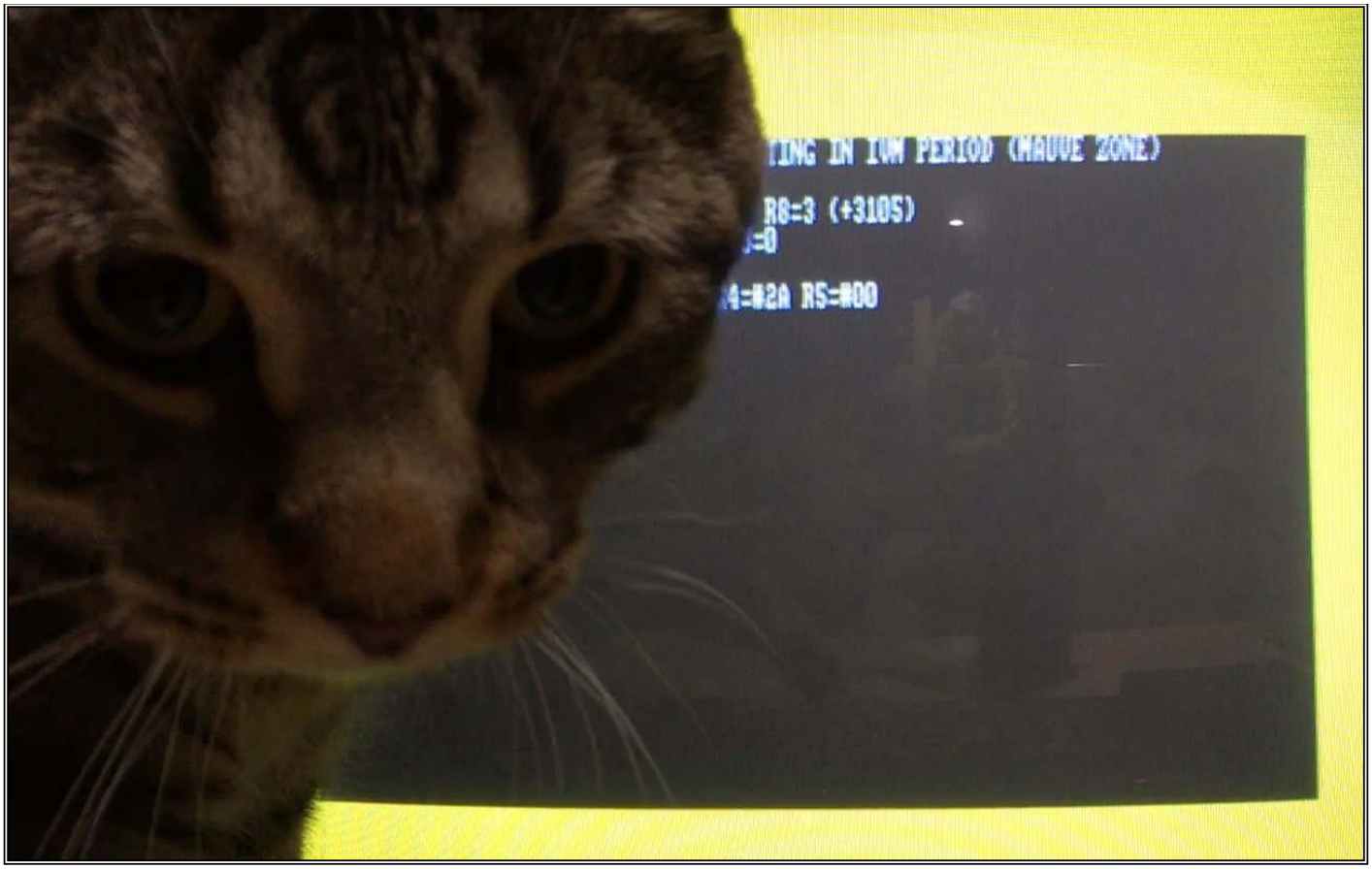
Sur un PPI, on peut programmer le sens du port B (entrée ou sortie)
En sortie, on peut donc y loger une valeur, qu'on pourra relire.

Le PPI du CPC LOWCOST ne permettrait pas de relire une valeur stockée dans le port B placé en sortie. On relit donc systématiquement la valeur du port B en entrée (et donc les périphériques qui y sont reliés).
Ceci reste à vérifier cependant, à défaut d'indiquer quels PPI étaient installés dans les CPC de test.

AMSTRAD a utilisé, tout comme pour les CRTC, des PPI 8255 de plusieurs fabricants.
On peut dénombrer les circuits suivants, utilisés dans tous les modèles indistinctement (464, 664, 6128)

- NEC D8255AC-2
- NEC D8255AC-5
- TOSHIBA TMP8255AP-5

À noter que la différence entre « -2 » et « -5 » tient à la fréquence maximale gérable par le circuit (4 Mhz pour le « -5 » et 5 Mhz pour le « -2 »)



Raaahhhhhhhhh !!
Con de Chat Canadien Curieux devant un CPC