Introduction au développement matériel et logiciel sur Gameboy

Jean-Michel FRIEDT, 18 mars 2001

1 Aspects matériels

La connexion d'une EEPROM aux seules fin d'exécuter un programme est immédiate, puisque le Gameboy fournit, aux niveaux TTL, un bus de données de 8 bits, un bus d'adresses de 16 bits et les deux signaux de contrôle RD# et WR# habituels.

La plage d'adresses allouée à la ROM est 0000-7FFF : il est donc possible d'utiliser la ligne d'adresse 15 comme signal d'activation de la ROM (CS#). Le signal de lecture RD# du Gameboy est connecté à la broche OE# de la ROM (pour que celle-ci définisse la valeur sur le bus de données lors d'une lecture).

Pour permettre, en plus de la simple lecture d'un programme de la ROM, de contrôler des lignes de sorties numériques ou de lire des données numériques issues de capteurs connectés au Gameboy, il faut ajouter un décodeur d'adresse (représenté par le 74138 sur le schéma de principe figure 2). En effet, la communication avec le monde extérieur nécessite l'ajout de composants – latch pour la sortie (74574) et diviseur de bus 3-états (74245) en entrée – qu'il faut pouvoir sélectionner indépendamment de la ROM.

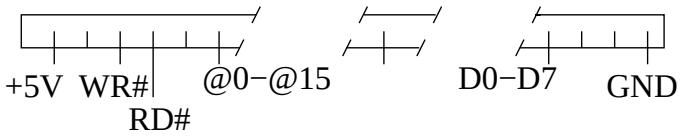

FIG. 1 – *Connecteur du Gameboy, vue de* **dessus** *de la console, écran en direction opposées de l'observateur. Les deux broches aux deux extrémités (+5V et GND) sont indiquées correctement, puis le bus d'adresse (16 bits) et de données (8 bits) sont adjacents.*

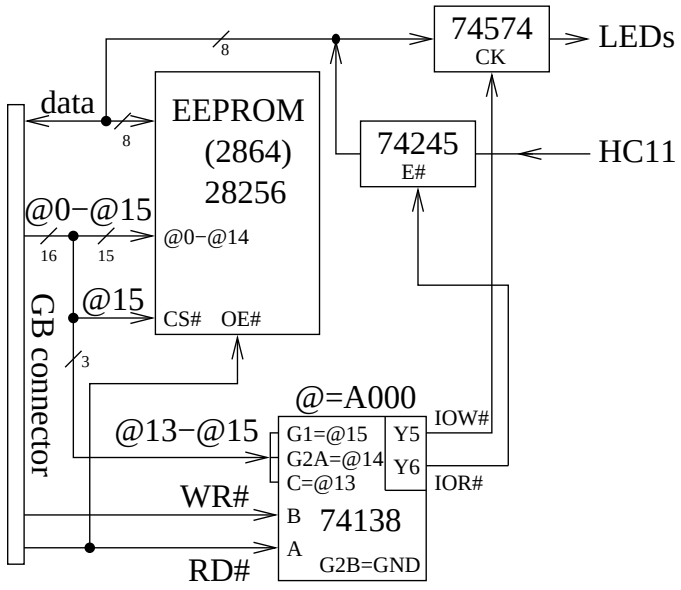

FIG. 2 – *Schéma de principe d'une cartouche de Gameboy*

La plage d'adresse allouée à la RAM externe sur une cartouche de Gameboy commence en A000 (et va jusqu'en BFFF). Nous utiliserons cette adresse comme port de communication entre le Gameboy et le monde extérieur. La ROM n'est pas activée lors d'un appel à cette adresse car le bit d'adresse A15 est alors au niveau haut (pour tout appel au-delà de l'adresse 8000 – rappelons que l'espace de la ROM s'étend jusqu'en 7FFF). Deux possibilités s'offrent encore à nous : l'accès au monde extérieur en lecture ou en écriture. Lors d'une écriture, WR# est bas et la sortie Y5 du 74138 est activée. Lors d'une lecture, RD# est bas et la sortie Y6 du 74138 est activée. Il reste à connecter ces deux signaux, Y5 et Y6, aux signaux d'activation des composants isolant le Gameboy du monde extérieur (CK pour le 74574, E# pour le 74245).

2 Aspects logiciels

Au moins deux outils de développement pour le Gameboy sont disponibles sous Linux : `rgbds` est un assembleur, tandis que `gbdk` est un compilateur C. Il est plus pratique d'utiliser ces outils qui génèrent automatiquement un entête requis au début de la ROM pour que le programme soit exécuté par le Gameboy (cet entête contient un checksum qui doit être juste faute de quoi la ROM est refusée par le Gameboy à l'allumage).

Ces deux outils ont été utilisé avec succès pour exécuter un programme à partir de la ROM et pour accéder aux ports de communication avec le monde extérieur.

Le programme en assembleur est copié du tutorial `galp` d'introduction à `rgbds` et modifié pour accéder en fin de la routine d'affichage du texte au port de sortie.

```
; jmfriedt, 5/02/01
INCLUDE "gbhw.inc"
        INCLUDE "ibmpc1.inc"
        SECTION "Org $100",HOME[$100]
        nop
        jp      begin

        ROM_HEADER      ROM_NOMBC, ROM_SIZE_32KBYTE, RAM_SIZE_2KBYTE

        INCLUDE "memory.asm"

TileData:
        chr_IBMPC1      1,8
begin:
        di
        ld      sp,$ffff

        call    StopLCD

        ld      a,$e4
        ld      [rBGP],a        ; Setup the default background palette

        ld      a,0
        ld      [rSCX],a
        ld      [rSCY],a

        ld      hl,TileData
        ld      de,$8000
        ld      bc,8*256        ; length (8 bytes per tile) x (256 tiles)
        call    mem_CopyMono    ; Copy tile data to memory

        ld      a,$20           ; Clear tile map memory
        ld      hl,$9800
        ld      bc,SCRN_VX_B * SCRN_VY_B
        call    mem_Set

        ld      hl,Title        ; Draw title
        ld      de,$9800+3+(SCRN_VY_B*7)
        ld      bc,13
        call    mem_Copy

; Now we turn on the LCD display to view the results!
        ld      a,LCDCF_ON|LCDCF_BG8000|LCDCF_BG9800|LCDCF_BGON|LCDCF_OBJ16|LCDCF_OBJOFF
        ld      [rLCDC],a       ; Turn screen on

ld a,$ff
wait:
dec a
ld [$a000],a ;   <- doit permettre d'ecrirer ff en RAM a000 <- LEDs
ld hl,$ffff
bll:
dec hl
jp NZ,bll
        jp      wait

Title:
        DB      "Hello World !"


; *** Turn off the LCD display ***

StopLCD:
        ld      a,[rLCDC]
        rlca                    ; Put the high bit of LCDC into the Carry flag
        ret     nc              ; Screen is off already. Exit.

; Loop until we are in VBlank

.wait:
        ld      a,[rLY]
        cp      145             ; Is display on scan line 145 yet?
        jr      nz,.wait        ; no, keep waiting

; Turn off the LCD

        ld      a,[rLCDC]
        res     7,a             ; Reset bit 7 of LCDC
        ld      [rLCDC],a

        ret

;* End of File *
```

Le programme en C est plus simple à comprendre au premier abord. `gbdk` fournit de nombreuses fonctions, notamment graphiques, par défaut, et est donc très souple d'emploi au premier contact. Cependant, le code généré sur un exemple aussi simple que celui présenté ici est 4 fois plus gros (près de 8 KB) que son équivalent en assembleur.

```
/* Sample Program to demonstrate the drawing functions in GBDK */
/* Jon Fuge jonny@q-continuum.demon.co.uk */

#include <gb/gb.h>
// #include <stdio.h>
#include <gb/drawing.h>


void main(void)
{UBYTE  *c,cpt;int i;

 c=0xA000;cpt=0;                  // address of pointer <- write to @ 0xA000
 gotogxy(2,2);gprintf("hello world");
```

```
gotogxy(2,4);gprintf("jmfriedt");
   /* Draw two circles, a line, and two boxes in different drawing modes */
color(LTGREY,WHITE,SOLID);
circle(140,20,15,M_FILL);
color(BLACK,WHITE,SOLID);
circle(140,20,10,M_NOFILL);
color(DKGREY,WHITE,XOR);
circle(120,40,30,M_FILL);
line(0,0,159,143);
color(BLACK,LTGREY,SOLID);
box(0,130,40,143,M_NOFILL);
box(50,130,90,143,M_FILL);
while (1) {cpt++;*c=cpt;for (i=0;i<6500;i++) {}} // use different vars for
}                                                // counter and output pointer @
```

Un outil très utile, tout au moins pour la partie affichage graphique, est un émulateur. Deux logiciels de ce type sont disponibles pour Linux : `vgb` (sous forme binaire uniquement) et xgnuboy. Les sorties d'écran des deux programmes précédents sont visibles figure 3.

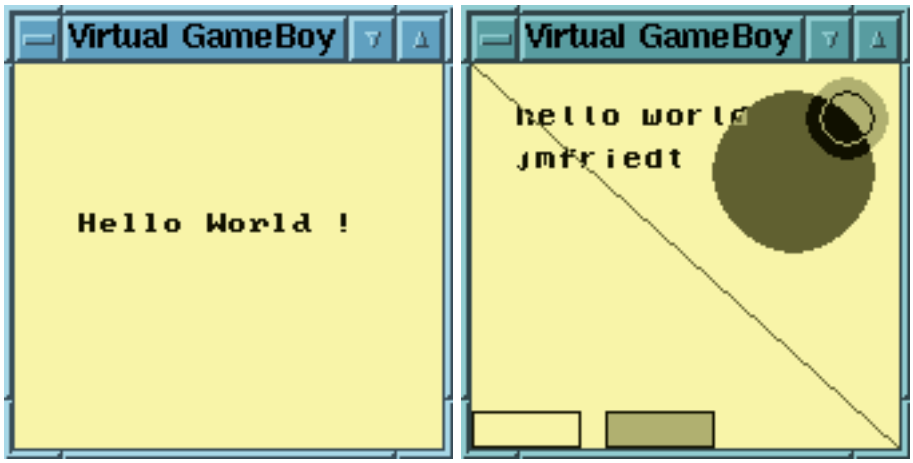

FIG. 3 – *Sortie d'écran de l'émulateur* `vgb`

3 Extensions possibles

Les capacités du Gameboy, au-dela de l'affichage, sont relativement restreintes (absence de timer, d'accès aux interruptions matérielles, de port de communication à la norme RS232 ...). Il semble donc intéressant de pouvoir le faire communiquer avec un microcontroleur possédant ce type de périphériques. À ce fins, une mémoire intermédiaire de type FIFO (AM7202A par exemple) semble adéquate. En-effet, le microcontroleur (dans notre cas un 68HC11F1 ou un TMPZ84) écrit les données à afficher à l'écran du Gameboy dans la FIFO. Le Gameboy attend les donnés jusqu'à ce que le flag indiquant que la mémoire est a moitié pleine (HF#) soit déclenché. À partir de ce moment, le microcontroleur est interdit d'accès à la FIFO, tandis que le Gameboy lit rapidement le contenu de la mémoire, jusqu'à épuisement des données stockées dans celle-ci (broche EF# active indiquant que la FIFO est vide). Le Gameboy rend alors l'accès à la FIFO par le microcontroleur possible, affiche les points lus et attend une nouvelle série de points.

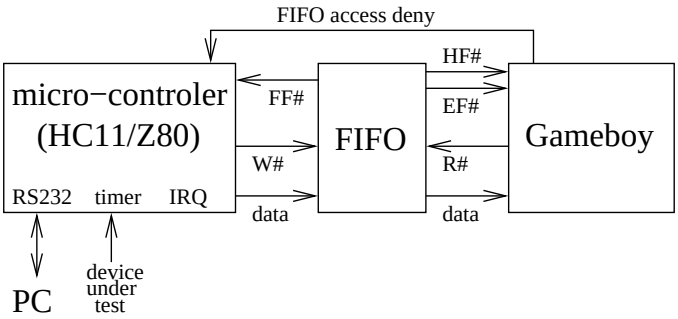

FIG. 4 – *Schéma de principe de communication entre un microcontroleur et le Gameboy*