interface d'application

Ce document décrit les fonctions à utiliser pour programmer des applications pour la console STM8-GAMEPAD.

Les applications doivent-être écrites en assembleur car les paramètres des fonctions sont passés dans les registres **A**,**X** et **Y** et non sur la pile comme le langage **C** le fait.

Il y a toujours moyen d'écrire les applications en **C** mais en utilisant l'assembleur inline pour charger les registres et faire l'appel vers la sous-routine.

NOTES

1. Les valeurs d'intervalles sont indiqué de la façon suivante **[début..fin[|]**
    - Si la valeur *fin* est suivie de **[** ça signifie que cette valeur est exclue de l'intervalle.
    - Si la valeur *fin* est suivie de **]** ça signifie que cette valeur est incluse dans l'intervalle.
2. **HRES** Résolution horizontale de l'affichage.
3. **VRES** Résolution verticale de l'affichage.
4. **cx** varible octet contenant la coordonnée x du curseur texte. **[0..CHAR_PER_LINE[**.
5. **cy** variable octet contenant la coordonnée y du curseur texte. **[0..LINE_PER_SCREEN[**.
6. **CHAR_PER_LINE** Nombre de caractères affichable par ligne.
7. **LINE_PER_SCREEN** Nombre de lignes par écran.
8. Les variables **cx** et **cy** peuvent-être modifiées par le programme pour positionner le curseur texte. La fonction **tv_cls** les initialisent à **0**.
9. L'espace réservé pour les variables d'application est dans l'intervalle **[4..KERNEL_VAR_ORG[**. Présentement **KERNEL_VAR_ORG=0x60**. Vérifiez cette valeur dans le fichier hardware_init.asm. Les applications définissent leur variables dans le segment **G_DATA** comme suit


```
    .area G_DATA (ABS)
    .org 4
score: .blkw 1 ; game score
speed: .blkb 1 ; snake speed delay
chrono: .blkb 1 ; chronometer delay
max_score: .blkw 1 ; maximum score
game_flags: .blkb 1 ; game boolean flags
snake_len: .blkb 1 ; snake length
snake_dir: .blkb 1 ; head direction
food_coord: .blkw 1 ; food coordinates
snake_body: .blkw 32 ;  snake rings coords
```

10. Lorsqu'une application est ajoutée au système ses nom et adresse doivent-être ajoutés au système de menu dans le fichier app.asm. Cette liste doit se terminée par une valeur nulle.

```
prog_list: ; liste des applications disponibles.
.asciz "SNAKE" ; nom qui apparaît dans le menu
.word snake ; nom de la fonction qui lance l'application.
; 2ième application
.asciz "FALL"
.word fall
; 3ième application
.asciz "QUICK BROWN FOX"
.word quick
; fin de la liste
.word 0
```

Le menu supporte un maximum de 24 applications. C'est le nombre de lignes texte de l'affichage et cette valeur ne devrait pas varier.

11. La mémoire étendue du stm8s207k8 de **32Ko [0x10000..0x17FFF]** n'est pas utilisée par le système mais une application peut l'utiliser pour sauvegarder des données persistantes. l'EEPROM de **2Ko [0x4000..0x47FF]** peut aussi être utilisée à cet effet.

---

index

SON

- **beep** Cette sous-routine n'accepte aucun paramètre. Elle génère un son de 1000 Hertz pour une durée de 8/60ième de seconde (i.e. ~133msec).

- **tone** génère une tonalité de fréquence spécifiée dans le registre **X** et de durée en 60ième de seconde spécifiée dans le registre **A**.
    - **A** durée du son en 60ième de seconde.
    - **X** fréquence en hertz.

- **tune** Joue une séquence de notes. Le registre **Y** indique l'adresse de la séquence.
    - **Y** adresse de la mélodie. Il s'agit d'une liste formée de la fréquence suivit de la durée. La liste doit se terminer par {0,0}. La macro **_note f,d** facilite la constuction de cette liste.
    - une fréquence nulle indique une pause.

```
```

```
    .macro _note  f,d
    .word f
    .byte d
    .endm

play_scale:
    ldw y,#gamme
    call tune
    ret

gamme:
    _note 523,10  ; do
    _note 554,10  ; do#
    _note 587,10  ; ré
    _note 622,10  ; ré#
    _note 659,10  ; mi
    _note 698,10  ; fa
    _note 740,10  ; fa#
    _note 784,10  ; sol
    _note 831,10  ; sol#
    _note 880,10  ; la
    _note 932,10  ; la#
    _note 988,10  ; si
    _note 0,0     ; end
```

- **noise** Produit un bruit blanc d'une durée indiquée par le paramètre passé dans **A**.
    - **A** Durée en 60ième de seconde.

index

KEYPAD

PROF

- **keypad_input** Retourne l'état des 6 boutons dans le registre **A**. Cette lecture n'est pas filtrée pour les rebonds des commutateurs. Les constantes suivantes sont définies pour identifier les boutons.
    - **BTN_LEFT=1**
    - **BTN_DOWN=2**
    - **BTN_RIGHT=4**
    - **BTN_UP=8**
    - **BTN_B=16**
    - **BTN_A=32**

- **read_keypad** Lecture du keypad avec filtrage antirebond. L'état des boutons est retournée dans le registre **A**. Exemple d'utilisation:

```
;--------------------------
; read keypad
; LEFT turn left
; RIGHT turn right
; UP increase speed
; DOWN decreas speed
;--------------------------
    KPAD=1
user_input:
    push #0
    call read_keypad
    jreq 8$
    ld (KPAD,sp),a
    ld a,#BTN_LEFT
    and a,(KPAD,sp)
    jreq 2$
    call rotate_head
    jra 6$
2$: ld a,#BTN_RIGHT
    and a,(KPAD,sp)
    jreq 3$
    call rotate_head
    jra 6$
3$:
    ld a,#BTN_UP
    and a,(KPAD,sp)
    jreq 4$
    ld a,#MIN_SPEED
    cp a,speed
    jreq 6$
    _decz speed
    call prt_info
    jra 6$
4$: ld a,#BTN_DOWN
    and a,(KPAD,sp)
    jreq 6$
    ld a,#MAX_SPEED
    cp a,speed
    jreq 6$
    _incz speed
    call prt_info
6$:
    ldw x,#10
    call wait_key_release
8$:
    _drop 1
    ret
```

PROF

- **wait_key** Attend qu'un bouton soit enfoncé et retourne l'état dans le registre **A**.

- **wait _key_release** Attend que tous les boutons soient relâchés. Un délais d'expiration de l'attente doit-être founis dans **X**.
    - **X** délais d'expiration de l'attente en 60ième de seconde.

index

AFFICHAGE

- **tv_cls** Efface l'affichage au complet.

- **set_pixel** Allume un pixel. Les paramètres sont passés dans le registre **X**.
    - **XL** coordonnée X du pixel [0..HRES[.
    - **XH** coordonnée Y du pixel [0..VRES[.

- **reset_pixel** Éteint un pixel. Les paramètres sont passés dans le registre **X**.
    - **XL** coordonnée X du pixel {0..HRES-1}.
    - **XH** coordonnée Y du pixel {0..VRES-1}.

- **invert_pixel** Inverse l'état du pixel. Les paramètres sont passés dans le registre **X**.
    - **XL** coordonnée X du pixel {0..HRES-1}.
    - **XH** coordonnée Y du pixel {0..VRES-1}.

- **clr_text_line** Efface une ligne de texte.
    - **A** numéro de la ligne à effacer {0..23}.

- **scroll_text_up** Décale l'affichage texte vers le haut d'une ligne et efface la dernière ligne.

- **crlf** Carriage return line feed. Renvoie le curseur texte au début de la ligne suivante.

- **cursor_right** Déplace le curseur texte vers la droite d'un caractère.

- **tv_putc** Affiche le caractère qui est dans le registre **A** à la position courante du curseur texte. Avance le curseur à la position suivante.
    - **A** caractère à affiché.

- **tv_puts** Affiche une chaîne ce caractères ASCII zéro terminée à la position actuelle du curseur. Àprès l'impression **Y** pointe sur le zéro terminal de la chaîne.
    - **Y** adresse de la chaîne ASCIZ.

- **center_align** affiche une ligne de texte centrée horizontalement. Àprès l'impression **Y** pointe sur le zéro terminal de la chaîne.
    - **Y** adresse de la chaîne ASCIZ.

- **put_uint16** Affiche à la position du curseur la valeur entière non signée contenue dans le registre **X**.
    - **X** Entier à afficher.

- **roll_up** déroule l'affichage vers le haut jusqu'à ce qu'il soit vide ou qu'une touche du kpad soit enfoncée.
    - **A** délais contrôlant la vitesse de déroulement. Le délais est en 60ième de secondes.

- **line** Trace une ligne droite entre les coordonnées **{x0,y0}** et **{x1,y1}** excluant ce dernier point.
    - **XL** coordonnée **x0** [0..HRES[
    - **XH** coordonnée **x1** [0..HRES[
    - **YL** coordonnée **y0** [0..VRES[
    - **YH** coordonnée **y1** [0..VRES[

- **rectangle** Dessine un rectangle dont les coordonnées des coins supérieur gauche et inférieur droit sont données.
    - **XL** coordonnée **x** [0..HRES[coordonée gauche
    - **XH** coordonnée **y** [0..VRES[coordonnnée haut
    - **YL** coordonnée **width** [0..HRES[largeur
    - **YH** coordonnée **height** [0..VRES[hauteur

- **put_sprite** Affiche un petit graphique d'au maximum 8 pixels en largeur et un maximum de **VRES** pixels en hauteur. Chaque rangé du sprite est représenté par un seul octet et le sprite comprend autant d'octets que sa hauteur. La fonction logique **XOR** est utilisée pour afficher les sprites. Cette méthode permet de détecter automatiquement les collisions. La fonction retourne une valeur dand **A** et ajuste le drapeau **Z** en fonction de cette valeur.

Entrées:

PROF

- **A** hauteur du sprite
- **XL** coordonnée x à gauche du sprite.
- **XH** coordonnnée y haut du sprite.
- **Y** adresse du sprite

Sorties:

- **A** différent de zéro s'il y a eu collision
- **Z 0** s'il y a eu collision.

- **scroll_up** Glisse les rangées de l'intervalle [de...jusqu'à[vers le haut d'une rangée et efface les pixels de la dernière rangée.
    - **XL** *de*
    - **XH** *jusqu'à* (exclue)

- **scroll_down** Glisse les rangées de l'intervalle [de...jusqu'à[vers le bas d'une rangée et efface les pixels de la première rangée.
    - **XL** *de*
    - **XH** *jusqu'à* (exclue)

- **scroll_left** Glisse vers la gauche de 4 pixels l'intervalle de rangées [de...jusqu'à[et efface les pixels à droite de l'écran.
    - **XL** *de*
    - **XH** *jusqu'à* (exclue)

- **scroll_right** Glisse vers la droite de 4 pixels l'intervalle de rangées [de...jusqu'à[et efface les pixels à gauche de l'écran.
    - **XL** *de*
    - **XH** *jusqu'à* (exclue)

---

index

DIVERS

- **pause** Suspend l'exécution pour une durée déterminée par la valeur passée dans **A**.
    - **A** durée de la pause en 60ième de seconde.

index