SYSTEMES EMBARQUES 1

TP05 – IMAGES AU FORMAT XPM SUR ECRAN LCD

Samuel Mertenat
Classe T2f

Fribourg, le 20.01.15
13h – 16h35

TABLE DES MATIERES

SYSTEMES EMBARQUES 1

TP05 – IMAGES AU FORMAT XPM SUR ECRAN LCD

BUTS ET OBJECTIFS DU TRAVAIL PRATIQUE

Ce cinquième travail pratique est l'occasion d'étudier le format d'image « .xpm », produit à l'aide du logiciel « imagemagick » et d'images « .jpg » / « .png » et d'en afficher le résultat sur l'écran LCD de la cible.

Les objectifs de ce travail sont :

- Concevoir et développer une petite application en C sur un système embarqué
- Transformer une image .jpg / .png sous format .xpm
- Afficher une image sur un écran LCD embarqué
- Décrire le fonctionnement d'un écran LCD et de son contrôleur
- Développer et débugger une application en C sous Eclipse et la toolchain GNU

Spécifications à implémenter :

- Conversion d'une image en format .jpg / .png en format .xpm sous la machine hôte
  - Utiliser l'outil imagemagick
  - Adapter le format de l'image à la taille de l'écran
- Développer une bibliothèque de conversion d'images sous format xpm vers un bitmap pour l'écran au format de l'écran RGB565
- Développer l'application mettant en œuvre cette infrastructure et permettant d'afficher les 3 images sur l'écran de la cible APF27
- Les 3 images doivent être affichées sur l'écran LCD en moins de 5 secondes

ANALYSE

LE FORMAT « .XPM »

Le format .xpm est un format d'image spécialisé dans les icônes des environnements graphiques. La compression effectuée est très faible comparée à celle des formats JPG, PGN ou SVG ; il est principalement destiné aux petites images et aux icônes.

La première ligne indique le nom du fichier (« logo_sis ») et la ligne suivante nous donne des informations, telles que la largeur (128), la longueur (154), le nombre de couleurs (256) et le nombre de caractères nécessaires pour coder chaque couleur (2)[1].

[1] *http://fr.wikipedia.org/wiki/X_PixMap*

```
/* XPM */
static char *logo_sis[] = {
/* columns rows colors chars-per-pixel */
"128 154 256 2 ",
"   c #010101",
".  c #060709",
"X  c #07080B",
"o  c #090A0D",
…
/* pixels */
…
"IXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIX
IXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXI
XIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIXIX"
};
```

Figure 1: Aperçu d'un fichier XPM

Pour convertir des images au format .jpg /.png en .xpm, nous pouvons utiliser le logiciel « imagemagick » (« sudo apt-get install imagemagick ») à l'aide du terminal, en tapant, par exemple, la commande suivante : « convert logo_sis.jpg logo_sis.xpm ».

LE MONITEUR LCD

Le fichier « imx27_lcdc.h », inclut dans le fichier « main.c » du squelette du TP, nous apporte de précieuses informations au sujet du LCD (workspace/se12/apf27/source/imx27_lcdc.h). On y trouve des symboles vers des constantes, telles que la largeur ou la hauteur en pixels ou les différentes méthodes nécessaires à l'initialisation de l'écran.

```
/* LCD details (LW700AT) */
#define IMX27_LCD_WIDTH          800
#define IMX27_LCD_HEIGHT         480
#define IMX27_LCD_FREQ           33260000 /* Hz */
#define IMX27_LCD_BPP            16

/**
 * Initialize the liquid crystal display controller
 */
extern void imx27_lcdc_init();


/**
 * Enable the liquid crystal display controller
 */
extern void imx27_lcdc_enable();
```

Figure 2: Constantes et méthodes utiles à l'initialisation de l'écran

Pour afficher quelque chose à l'écran, nous devons d'abord spécifier ses coordonnées en x et y. Dans le cadre de ce TP, nous serons amené à afficher une image, dont la largeur correspond à des colonnes (x) et la hauteur à des lignes (y).

Pour déplacer le curseur (« bitmap ») d'une colonne ou plus (d'un pixel ; horizontalement), il suffit d'ajouter la valeur de la coordonnée x au « bitmap » ; quant aux déplacements à travers les lignes (d'un pixel, verticalement), il faudra multiplier la coordonnée y à la constante « IMX27_LCD_WIDTH », dont le résultat sera ajouté à l'adresse du « bitmap ».

Pour récupérer et gérer le curseur (« bitmap »), il nous faudra créer un pointeur, sur 16 bits, en appelant la méthode « imx27_lcdc_get_bitmap() ».

```
/**
 * Return the base address of the LCDC bitmap
 */
extern uint16_t* imx27_lcdc_get_bitmap();
```

Figure 3: Méthode nécessaire à la création du pointeur

Pour illustrer le fonctionnement du LCD, voici un petit schéma du sens des coordonnées, ainsi que des constantes utiles à son utilisation.

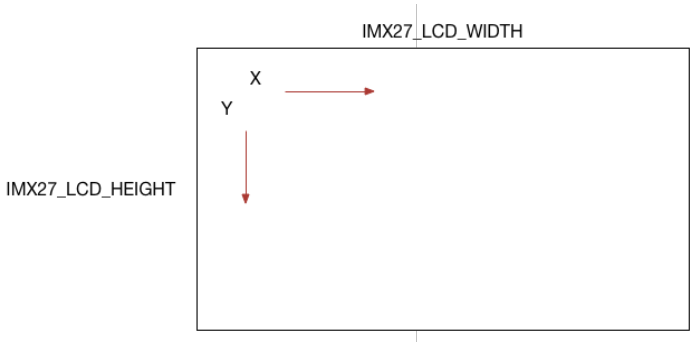

Figure 4: Fonctionnement des coordonnées du LCD

CONCEPTION

STRUCTURE DU PROGRAMME

Le programme se décompose en 3 fichiers :

- main.c : Module principale, faisant appel à la méthode « convert_xpm_image () » pour générer les images et les afficher à l'aide de la méthode « display_image() »
- xpm.c : Module pour génération des images au format XPM
- xpm.h : Entêtes du fichier xmp.c

EXPLICATION DES DIFFERENTS FICHIERS

MAIN.C

display_image() :

- Crée un pointeur afin de pouvoir utiliser le bitmap
- Positionne le bitmap aux coordonnées x et y passées en paramètres
- Crée un pointeur « from » (source des données à copier) à partir de l'adresse de l'image au format XPM passée en paramètre
- Crée un pointeur « to » (destination des données à copier) à partir de l'adresse du bitmap
- Parcourt l'image, dans sa hauteur, afin de copier ligne après ligne :
    - Utilise la fonction « memcpy » avec les paramètres « to » (pointeur), « from » (pointeur) et la largeur de l'écran, multiplié par deux (pixel codé sur 16 bits ; code couleur)

- Saute une ligne (de l'écran ; valeur du pointeur « to »), en « avançant » de « IMX27_LCD_WIDTH » et en fait de même avec l'image (valeur du pointeur « from » incrémentée d'une largeur d'image)

main() (méthode principale) :

- Initialisation du LCD avec les méthodes « imx27_lcdc_init() » et « imx27_lcdc_enable() »
- Génération (« convert_xpm_image () ») et affichage du logo de l'HEIA (« display_image »)
- Génération (« convert_xpm_image () ») et affichage du logo du sis (« display_image »)
- Génération (« convert_xpm_image () ») et affichage du logo de l'i2c (« display_image »)

XPM.C

Le fichier « xpm .c » reprend en grande partie le code de l'exercice 4 de la série 9. Dès lors, uniquement le code modifié ou ajouté est présenté ci-dessous. Avant de passer à l'optimisation du code afin de rendre possible l'affichage des 3 logos en moins de 5 secondes, seules quelques modifications ont été effectuées :

- Enlever la déclaration « static » de la méthode « convert_xpm_image() »
- Modifier le type de la valeur de retour de la méthode « convert_xpm_image() » de « image » à « xmp_image » (nom de la structure déclarée dans « xpm.h »)

XPM.H

Le fichier d'entête « xpm.h » nous permet de déclarer une structure de type « xpm_image » et de déclarer la méthode de conversion en « extern », ce qui nous permettra de pouvoir l'appeler depuis le fichier « main.c ».

OPTIMISATION DU CODE

Le code proposé dans la méthode « get_color() » étant peu optimisé, il en va de soit que le temps de traitement sera proportionnel à la taille de l'image à afficher.

```
uint16_t color = 0;
while ((map->code != code) && (colors > 0)) {
        map++; colors--;
}
if (colors > 0)
        color = map->color;
return color;
```

Pour remédier à ce problème et à ainsi pouvoir afficher les 3 logos dans un intervalle de 5 secondes, on commence par trier les valeurs à l'aide d'un « QuickSort », dans la méthode « convert_xpm_image() ».

```
qsort(map, colors, sizeof(struct color_code), compareColorCode);
```

Ce « QuickSort » nécessite une méthode permettant de comparer la valeur de deux codes couleur, que l'on nomme arbitrairement « compareColorCode() ».

```
/**
 * Compare 2 structures of color_code
 *
 * @param 2 structures color_code
 * @result value of the comparison
 */
int compareColorCode(const void *c1, const void *c2) {
    struct color_code* pC1 = (struct color_code*) c1;
    struct color_code* pC2 = (struct color_code*) c2;
    return pC1->code - pC2->code;
}
```

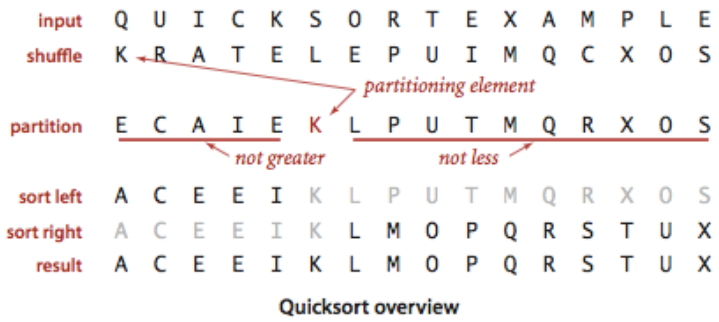

Figure 5: Principe de fonctionnement d'un QuickSort

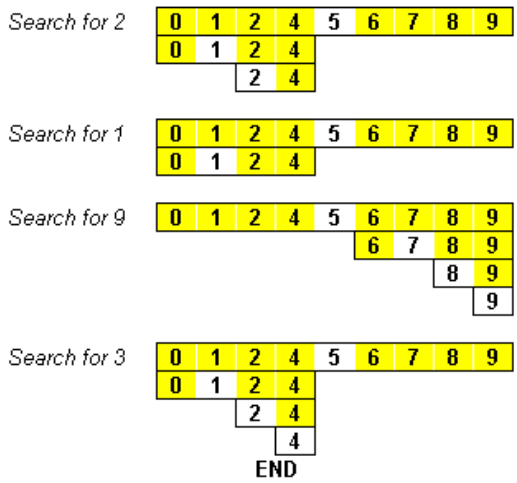

Figure 6: Principe du « binary search »

Maintenant que la partie de trie est réalisée, il suffit d'implémenter un algorithme de recherche efficace. Pour cette tâche, on utilisera le « binary search » dans la méthode « get_color »

Le principe est assez simple : on définit deux borner, aux deux extrémités de l'intervalle de données, que l'on compare ensemble après chaque subdivision de l'intervalle.

```
// searches for the color; binary search
uint32_t l = 0;                          // left
uint32_t r = colors - 1; // right
uint32_t m = 0;                          // middle
while (r != l + 1) {
      m = (r + l) / 2;          // cuts the interval by 2
      if (map[m].code >= code) {
        r = m;
      } else {
        l = m;
      }
}
```

TEST ET VALIDATION

Dans ce travail pratique, il y a que deux points à vérifier afin de valider le bon fonctionnement du programme : l'affichage de manière correcte des trois images et le temps de latence, qui devrait être inférieur à 5 secondes.

Je n'ai malheureusement aucune preuve pour attester de mes dires, mais l'affichage des logos était bien correct et dans le temps imparti.

CONCLUSION

Ce cinquième travail pratique aura été l'occasion de se familiariser avec le format d'image XPM et d'en comprendre la réalisation. D'autre part, ce TP nous aura permis d'utiliser l'écran LCD et d'en explorer brièvement le fonctionnement, qui je pense, à l'avenir, pourra nous servir pour d'autres travaux.

ANNEXE

Le code source du programme se trouve sur Git, à l'adresse : https://forge.tic.eia-fr.ch/git/samuel.mertenat/se12-tp.git, dans le dossier « tp5 ».

SOURCES

- Exercice 9.4 (langage C)

```
/**
 * Copyright 2014 University of Applied Sciences Western Switzerland /
    Fribourg
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 *
 * Project: EIA-FR·/ Embedded Systems 1 Laboratory
 *
 * Abstract:    TP5 - Introduction to XPM File Format & LDC Display
 *
 * Purpose: This module is simple program to convert xpm file into a
 *      16 bits bitmap to be printed out a LCD display
 *
 * Author:  <Samuel Mertenat>
 * Date:    <20.01.15>
 */

#include <stdio.h>
#include <string.h>

#include <imx27_lcdc.h>
#include "xpm.h"

#include "logo_heia.xpm"
#include "logo_sis.xpm"
#include "logo_i2c.xpm"

void display_image(struct xpm_image* img, uint16_t x, uint16_t y) {
    /* Documentation: imx27_lcdc.h */
    //·returns the base address of the LCD
    uint16_t *bitmap = imx27_lcdc_get_bitmap();

    // goes to the coordinate x, y
    bitmap += IMX27_LCD_WIDTH * y;
    bitmap += x;

    uint16_t *from = img->image;
    uint16_t *to = bitmap;
    // 2 bytes per pixel; line width x 2
    uint16_t line_length_bitmap = img->width * 2;

    // loops through each line of the picture
    for(uint16_t i = 0; i < img->height; i++) {
        memcpy(to, from, line_length_bitmap);
        to += IMX27_LCD_WIDTH;      // goes to next line
        from += img->width;
    }
}

int main() {
```

```
    printf ("\n");
    printf ("EIA-FR - Embedded Systems 1 Laboratory\n");
    printf ("TP5: Introduction to XPM File Format & LDC Display\n");
    printf ("     Convert xpm-files and display them on LCD\n");
    printf ("\n");

    imx27_lcdc_init();
    imx27_lcdc_enable();

    // creates & displays the logo for heia
    struct xpm_image img = convert_xpm_image (logo_heia);
    display_image(&img, 0, 0);

    // creates & displays the logo for sis
    img = convert_xpm_image (logo_sis);
    display_image(&img, IMX27_LCD_WIDTH/4 - img.width/2, IMX27_LCD_HEIGHT -
        img.height);

    // creates & displays the logo for i2c
    img = convert_xpm_image (logo_i2c);
    display_image(&img,(IMX27_LCD_WIDTH/4)*3 - img.width/2, IMX27_LCD_HEIGHT -
        img.height);

    return 0;
}
```

```
#pragma once
#ifndef XPM_H
#define XMP_H
/**
 * Copyright 2014 University of Applied Sciences Western Switzerland /
     Fribourg
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 *
 * Project: EIA-FR·/ Embedded Systems 1 Laboratory
 *
 * Abstract:    TP5 - Introduction to XPM File Format & LDC Display
 *
 * Purpose: This module is simple program to convert xpm file into a
 *      16 bits bitmap to be printed out a LCD display
 *
 * Author:  <Samuel Mertenat>
 * Date:    <20.01.15>
 */

#include <stdint.h>

/* Documentation: ex 9.4 */
// xpm image
struct xpm_image {
    uint32_t width;     // image width [pixels]
    uint32_t height;    // image height [pixels]
    uint16_t* image;    // image coded [RGB565]
};

/**
 * Convert a XPM image into a 16-bit bitmap format ready to be displayed
 * on the LCD display.
 *
 * @param xpm_data xpm-image to be converted
 * @result converted xmp_image
 */
extern struct xpm_image convert_xpm_image (char* xpm_data[]);

#endif
```

```
/**
 * Copyright 2014 University of Applied Sciences Western Switzerland /
     Fribourg
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 *
 * Project: EIA-FR·/ Embedded Systems 1 Laboratory
 *
 * Abstract:    TP5 - Introduction to XPM File Format & LDC Display
 *
 * Purpose: This module is simple program to convert xpm file into a
 *      16 bits bitmap to be printed out a LCD display
 *
 * Author:  <Samuel Mertenat>
 * Date:    <20.01.15>
 */

#include <stdint.h>
#include "xpm.h"

/* Documentation: ex 9.4 */
// color code structure used to build color map
struct color_code {
    uint32_t code;      // color code
    uint32_t color;     // color value in RGB565
};

//·Convert an ascii hex value into a interger value
static inline uint32_t a2i(char c) {
    if ((c >= '0') && (c <= '9'))
        return c - '0';
    if ((c >= 'A') && (c <= 'F'))
        return c - 'A' + 10;
    if ((c >= 'a') && (c <= 'f'))
        return c - 'a' + 10;
    return 0;
}

// Convert RGB888 value into RGB565
static inline uint32_t rgb565 (uint32_t r, uint32_t g, uint32_t b) {
    return (((r & 0xf8ul) >> 3) << 11) | (((g & 0xfcul) >> 2) << 5) | ((b &
        0xf8ul) >> 3);
}

/**
 * Get associated color value to a color code contained within the color map
 *
 * @param map color map
 * @param colors numbers of colors contained into the color map
 * @param line one line of the image to convert
```

```
 * @param chars number of characters per color-code
 * @result associated color value
 */

// variables for cache purpose
uint16_t cache = 0;
uint32_t oldCode = 0;

static uint16_t get_color (struct color_code* map, int colors, const char*
    line, int chars) {
    uint32_t code = 0;
    for (int i = chars; i > 0; i--)
        code = (code << 8) + *line++;

    // if cached, returns the color directly
    if (code == oldCode) {
        return cache;
    }

    // searches for the color; binary search
    uint32_t l = 0;             // left
    uint32_t r = colors - 1;    // right
    uint32_t m = 0;             // middle
    while (r != l + 1) {
        m = (r + l) / 2;        //·cuts the interval by 2
        if (map[m].code >= code) {
            r = m;
        } else {
            l = m;
        }
    }

    // if found, updates the cache and returns the color
    if (r < (colors) && map[r].code == code) {
        cache = map[r].color;
        oldCode = code;
        return cache;
    }
}

/**
 * Parse a color line to extract color code and color value
 *
 * @param c_str color line to be parsed
 * @param chars number of characters per color code
 * @result converted color code & value
 */
static struct color_code parse_color(const char* c_str, int chars) {
    struct color_code map = {.code=0, .color=0,};
    for (int j = chars; j > 0; j--) map.code = (map.code << 8) + *c_str++;

    while (*c_str != 0)
        if (*c_str++ == '#') {
            uint32_t r = (a2i(c_str[0]) << 4) + a2i(c_str[1]);
            uint32_t g = (a2i(c_str[2]) << 4) + a2i(c_str[3]);
            uint32_t b = (a2i(c_str[4]) << 4) + a2i(c_str[5]);
            map.color = rgb565(r,g,b);
            break;
        }
    return map;
```

```
}

/**
 * Compare 2 structures of color_code
 *
 * @param 2 structures color_code
 * @result value of the comparison
 */
int compareColorCode(const void *c1, const void *c2) {
    struct color_code* pC1 = (struct color_code*) c1;
    struct color_code* pC2 = (struct color_code*) c2;
    return pC1->code - pC2->code;
}

/**
 * Convert a XPM image into a 16-bit bitmap format ready to be displayed
 * on the LCD display.
 *
 * @param xpm_data xpm-image to be converted
 * @result converted xpm-image
 */
struct xpm_image convert_xpm_image (char* xpm_data[]) {
    struct xpm_image xpm = {.width=0, .height=0, .image=0,};
    uint32_t colors = 0;
    uint32_t chars = 0;

    sscanf(xpm_data[0], "%u %u %u %u",
        &xpm.width, &xpm.height, &colors, &chars);

    xpm.image = malloc(xpm.height * xpm.width * sizeof(*xpm.image));
    struct color_code* map = calloc (colors, sizeof(*map));
    if ((xpm.image != 0) && (map != 0)) {
        for (uint32_t i = 1; i <= colors; i++) {
            struct color_code ele = parse_color (xpm_data[i], chars);
            map[i-1] = ele;
        }

        // QuickSort for the binary search
        qsort(map, colors, sizeof(struct color_code), compareColorCode);

        uint16_t* p = xpm.image;
        for (uint32_t y = 0; y < xpm.height; y++) {
            const char* l = xpm_data[1 + colors + y];
            for (uint32_t x = 0; x < xpm.width; x++) {
                *p++ = get_color (map, colors, l, chars);
                l+=chars;
            }
        }

        free (map);
    }
    return xpm;
}
```