



# SYSTEMES EMBARQUES 2

---

TP12 – GRIBOUILLAGE

Fabio Valverde & Samuel Mertenat  
Classe T2f

Fribourg, les 4 mai / 18 mai 2015  
Salle C0016  
08h15-11h50

## TABLE DES MATIERES

1	Buts et Objectifs du travail pratique.....	3
2	Analyse .....	4
2.1	Calibration .....	4
2.2	Utilisation des images XPM .....	5
3	Tests et validation .....	7
3.1	Calibration .....	7
3.2	Utilisation des images XPM .....	8
4	Acquis .....	10
5	Problèmes.....	10
6	Perspectives .....	10
7	Conclusion .....	10
8	Références .....	10
9	Annexe .....	10

# SYSTEMES EMBARQUES 2

## TP12 – GRIBOUILLAGE

### 1 BUTS ET OBJECTIFS DU TRAVAIL PRATIQUE

Ce travail pratique a pour but la conception d'une application de dessin, en employant l'écran LCD, la partie tactile de celui-ci, ainsi que les doigts de l'utilisateur. L'utilisation de l'écran tactile nécessitera la réalisation d'une phase de calibration afin de pouvoir ajuster les coordonnées lues par la partie tactile, qui seront ensuite employées pour afficher la pression du crayon au bon endroit.

Fonctionnalités obligatoires :

- Effacement de l'écran
- Choix de la couleur du dessin (min 2 couleurs)

Contraintes :

- L'écran tactile doit être calibré avant de commencer à dessiner.
- Pour le dessin, il n'est pas nécessaire de relier les points entre eux ; vous pouvez simplement dessiner un carré de couleur (minimum 4 x 4 pixels) à l'endroit de la pression par le doigt.
- Le programme doit être principalement développé en C. L'assembleur doit être limité au strict minimum.
- Le programme ne doit pas comporter de variable globale (excepté les pointeurs sur les structures des contrôleurs).
- Le code doit être écrit avec un style consistant, des noms de variables et de procédures en anglais et bien choisis, et il doit être bien documenté (en anglais).

Fonctionnalités supplémentaires :

- Choix de la taille du crayon (4 x 4, 8 x 8, 16 x 16)
- Choix de la couleur du dessin (rouge, vert, bleu, jaune, blanc)
- Affichage dans le coin, en haut à gauche, de l'état du crayon (taille et couleur)

## 2 ANALYSE

### 2.1 CALIBRATION

L'écran n'étant pas calibré, il ne nous est pas possible d'afficher une image avec précision à l'endroit de la pression du doigt. Pour remédier à ce problème, nous devons donc calibrer l'écran ; nous utiliserons la méthode de calibration dite « Three-Point Calibration » pour résoudre ce problème.

Marche à suivre :

- Définir arbitrairement trois points, en triangle. Dans notre cas, (350,65), (195, 200) et (550, 350).
- Récupérer les positions retournées par la partie tactile pour ces 3 points.
- Calculer K et les constantes A, B, C, D, E et F au moyen d'une matrice.
- Calculer la position X-Y pour le LCD en fonction de la position X-Y retournée par la partie tactile («tsc2101\_read\_position() »).

Résolution de la matrice pour le calcul des constantes A-F :

- $X_{D0-2}$  /  $Y_{D0-2}$  : Coordonnées des points affichés sur l'écran
- $X_{0-2}$  /  $Y_{0-2}$  : Coordonnées des points retournées par la partie tactile

$$K = \begin{pmatrix} X & -X \\ 0 & 2 \end{pmatrix} \begin{pmatrix} Y-Y \\ 1 & 2 \end{pmatrix} - \begin{pmatrix} X & -X \\ 1 & 2 \end{pmatrix} \begin{pmatrix} Y-Y \\ 0 & 2 \end{pmatrix}$$

$$A = \left( \begin{pmatrix} X & -X \\ D0 & D2 \end{pmatrix} \begin{pmatrix} Y-Y \\ 1 & 2 \end{pmatrix} - \begin{pmatrix} X & -X \\ D1 & D2 \end{pmatrix} \begin{pmatrix} Y-Y \\ 0 & 2 \end{pmatrix} \right) / K$$

$$B = \left( \begin{pmatrix} X & -X \\ 0 & 2 \end{pmatrix} \begin{pmatrix} X & -X \\ D1 & D2 \end{pmatrix} - \begin{pmatrix} X & -X \\ D0 & D2 \end{pmatrix} \begin{pmatrix} X & -X \\ 1 & 2 \end{pmatrix} \right) / K$$

$$C = +Y \begin{pmatrix} X & X & -X & X \\ 0 & 2 & D1 & 1 & D2 \end{pmatrix} + Y \begin{pmatrix} X & X & -X & X \\ 1 & 0 & D2 & 2 & D0 \end{pmatrix} + Y \begin{pmatrix} X & X & -X & X \\ 2 & 1 & D0 & 0 & D1 \end{pmatrix} / K$$

$$D = \left( \begin{pmatrix} Y & -Y \\ D0 & D2 \end{pmatrix} \begin{pmatrix} Y-Y \\ 1 & 2 \end{pmatrix} - \begin{pmatrix} Y & -Y \\ D1 & D2 \end{pmatrix} \begin{pmatrix} Y-Y \\ 0 & 2 \end{pmatrix} \right) / K$$

$$E = \left( \begin{pmatrix} X & -X \\ 0 & 2 \end{pmatrix} \begin{pmatrix} Y & -Y \\ D1 & D2 \end{pmatrix} - \begin{pmatrix} Y & -Y \\ D0 & D2 \end{pmatrix} \begin{pmatrix} X & -X \\ 1 & 2 \end{pmatrix} \right) / K$$

$$F = +Y \begin{pmatrix} X & Y & -X & Y \\ 0 & 2 & D1 & 1 & D2 \end{pmatrix} + Y \begin{pmatrix} X & Y & -X & Y \\ 1 & 0 & D2 & 2 & D0 \end{pmatrix} + Y \begin{pmatrix} X & Y & -X & Y \\ 2 & 1 & D0 & 0 & D1 \end{pmatrix} / K$$

Dès que les constantes A-F sont calculées, il est possible de corriger les valeurs lues par la partie tactile pour les afficher correctement sur le LCD. Les équations à utiliser sont les suivantes :

- $X_D / Y_D$  : Coordonnées X-Y corrigées
- $X / Y$  : Coordonnées lues par la partie tactile

$$X_D = AX + BY + C$$

$$Y_D = DX + EY + F$$

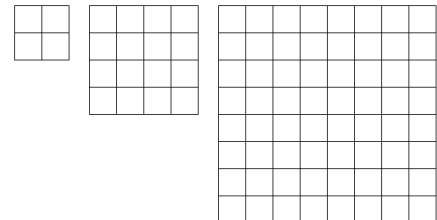
## 2.2 UTILISATION DES IMAGES XPM

Pour dessiner sur l'écran LCD, nous utiliserons des images XPM qui seront générées à chaque modification du crayon (taille ou couleur) ; ce qui nous évitera de devoir créer préalablement les images sous Photoshop pour chaque variante possible (3 tailles et 5 couleurs, 15 variantes).

Une image sera caractérisée par :

- Une hauteur (1 byte)
- Une largeur (1 byte)
- Un pointeur (2 bytes) vers la mémoire où l'image sera stockée

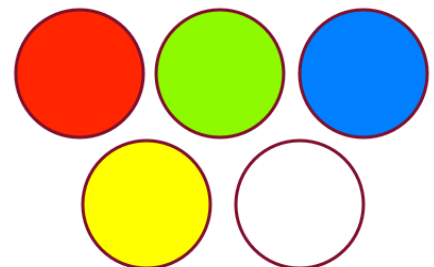
Les différentes tailles pour la mine de crayon seront stockées dans une énumération, où les différentes variantes seront :



- Petit : 4 x 4 pixels
- Moyen : 8 x 8 pixels
- Grand : 16 x 16 pixels

Et les différentes couleurs seront aussi stockées dans une énumération sous la forme RGB565 (16 bits), dont les possibilités seront les suivantes :

- Rouge : 255-0-0 (RGB888), 63488 (RGB565)
- Vert : 128-255-0 (RGB888), 34784 (RGB565)
- Bleu : 0-128-255 (RGB888), 1055 (RGB565)
- Jaune : 255-255-0 (RGB888), 65505 (RGB565)
- Blanc : 255-255-255 (RGB888), 65535 (RGB565)



La « création » de l'image consistera ensuite à allouer suffisamment d'espace mémoire pour pouvoir stocker l'image ; cet espace sera visé par le pointeur contenu dans la structure de l'image.

L'espace mémoire à allouer correspond au nombre de pixels, multiplié par la taille du codage de la couleur du pixel (RGB565, 16 bits, donc 2 bytes).

$$\text{Bytes nécessaires} = \text{hauteur} \times \text{largeur} \times 2$$

Il suffit ensuite de coder l'image, en parcourant la mémoire par pas de 16 bits, en y inscrivant le code couleur précédemment converti sur 16 bits. A la fin de l'opération, il est important de libérer la mémoire allouée à la précédente image, afin de ne pas saturer la mémoire.

```
// allocates the needed memory (size x size x 2 bytes)
xpm.img = malloc(xpm.height * xpm.width * sizeof(*xpm.img));
// codes the new representation [rgb565]
uint16_t* p = xpm.img;
for (uint8_t y = 0; y < xpm.height; y++) {
    for (uint8_t x = 0; x < xpm.width; x++) {
        *p++ = p_pencil->color;
    }
}

// liberates the memory used by the oldest representation and assigns the new representation to the pencil
free(p_pencil->lead_of_pencil.img);
p_pencil->lead_of_pencil = xpm;
```

Autres fonctionnalités liées aux images XPM / à l'écran LCD :

- Effacement de l'écran : « imx27\_lcd\_clear\_screen() »
- Affichage d'une image XPM : « display\_image(struct xpm\_image\* img, uint16\_t x, uint16\_t y) » (TP5 & TP10)

## 3 TESTS ET VALIDATION

### 3.1 CALIBRATION

A l'exécution du programme, nous sommes amenés à calibrer l'écran. La phase se déroule ainsi :

- Affiche le point 1 (350, 65) et attend tant que l'utilisateur n'a pas pressé sur l'écran
- Affiche le point 2 (195, 200) et attend encore la pression du doigt de l'utilisateur
- Affiche le point 3 (550, 350) et attend une dernière fois l'utilisateur

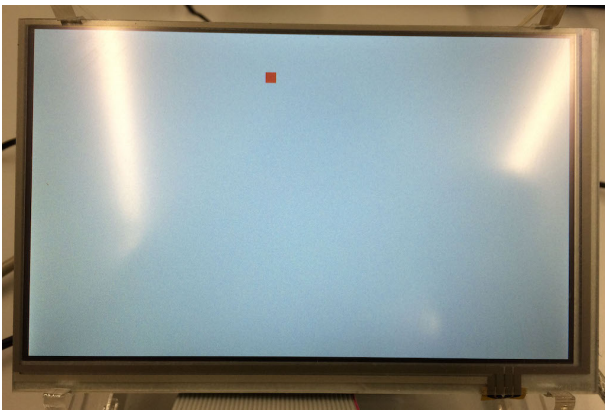


Figure 2: Affichage du point 1 (350, 65)

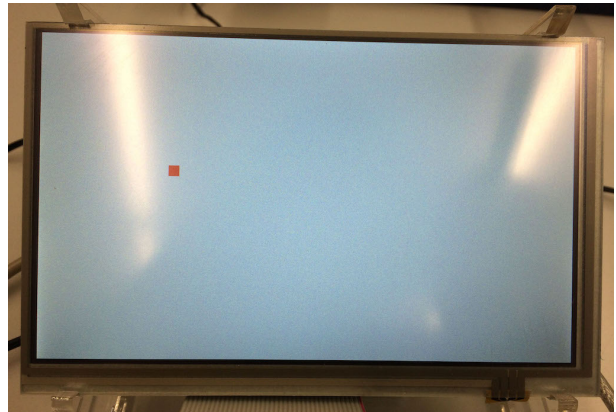


Figure 3: Affichage du point 2 (195, 200)



Figure 1: Affichage du point 3 (550, 350)

Dès lors que le programme dispose des valeurs correspondantes aux points affichés, il calcule les constantes A-F. En observant les valeurs calculées, visibles sur la capture ci-après, nous pouvons observer que l'ordre de grandeur des nombres est correct (si les valeurs avaient été très élevées, ce n'aurait pas été le cas).

```
Touchscreen - Three-Point Calibration
Please, press briefly on the red square on the screen!
A: 0.804007, B: -0.006825, C: -5.988061
D: -0.007028, E: -0.511991, F: 504.378601
```

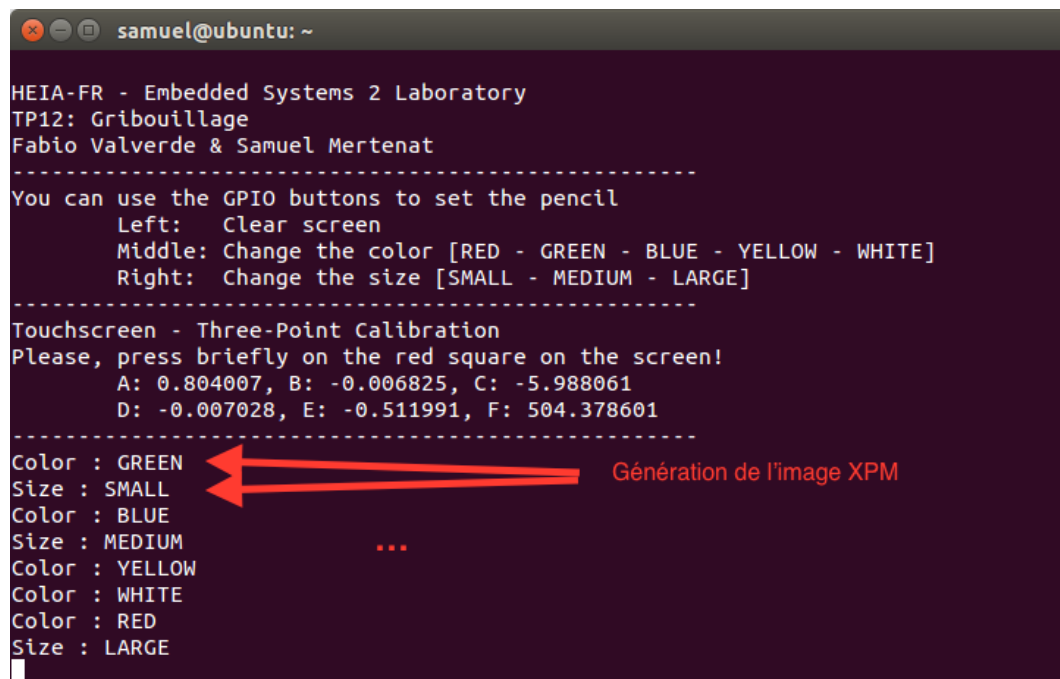
Constantes A-F  
(calibration)

Figure 4: Affichage des constantes A-F calculées lors de la phase de calibration

En appuyant dès à présent sur l'écran, nous pouvons constater que celui-ci nous obéit au doigt et à l'œil !

### 3.2 UTILISATION DES IMAGES XPM

Lorsqu'une pression est effectuée sur le bouton du milieu (GPIO, changer de couleur) ou celui de droite (GPIO, changer de taille), une nouvelle image XPM est générée, à l'aide de la méthode « `scribble_update_pencil(...)` ».



```
samuel@ubuntu: ~
HEIA-FR - Embedded Systems 2 Laboratory
TP12: Gribouillage
Fabio Valverde & Samuel Mertenat
-----
You can use the GPIO buttons to set the pencil
Left:  Clear screen
Middle: Change the color [RED - GREEN - BLUE - YELLOW - WHITE]
Right: Change the size [SMALL - MEDIUM - LARGE]
-----
Touchscreen - Three-Point Calibration
Please, press briefly on the red square on the screen!
A: 0.804007, B: -0.006825, C: -5.988061
D: -0.007028, E: -0.511991, F: 504.378601
-----
Color : GREEN
Size : SMALL
Color : BLUE
Size : MEDIUM
Color : YELLOW
Color : WHITE
Color : RED
Size : LARGE
```

Figure 5: Aperçu du fonctionnement du programme sur la console

Le résultat peut ensuite être observé dans le coin supérieur gauche du LCD, où se trouve l'aperçu de la mine du crayon.

Nous pouvons ensuite débiter notre gribouillage en appuyant sur l'écran. Pour rappel, les boutons du GPIO permettent de :

- Effacer l'écran (bouton de gauche)
- Changer de couleur (bouton du milieu)
- Change de taille (bouton de droite)

Sur la figure ci-dessous, nous pouvons observer que le dessin a été réalisé avec différentes couleurs, ainsi qu'avec des épaisseurs de crayon différentes.



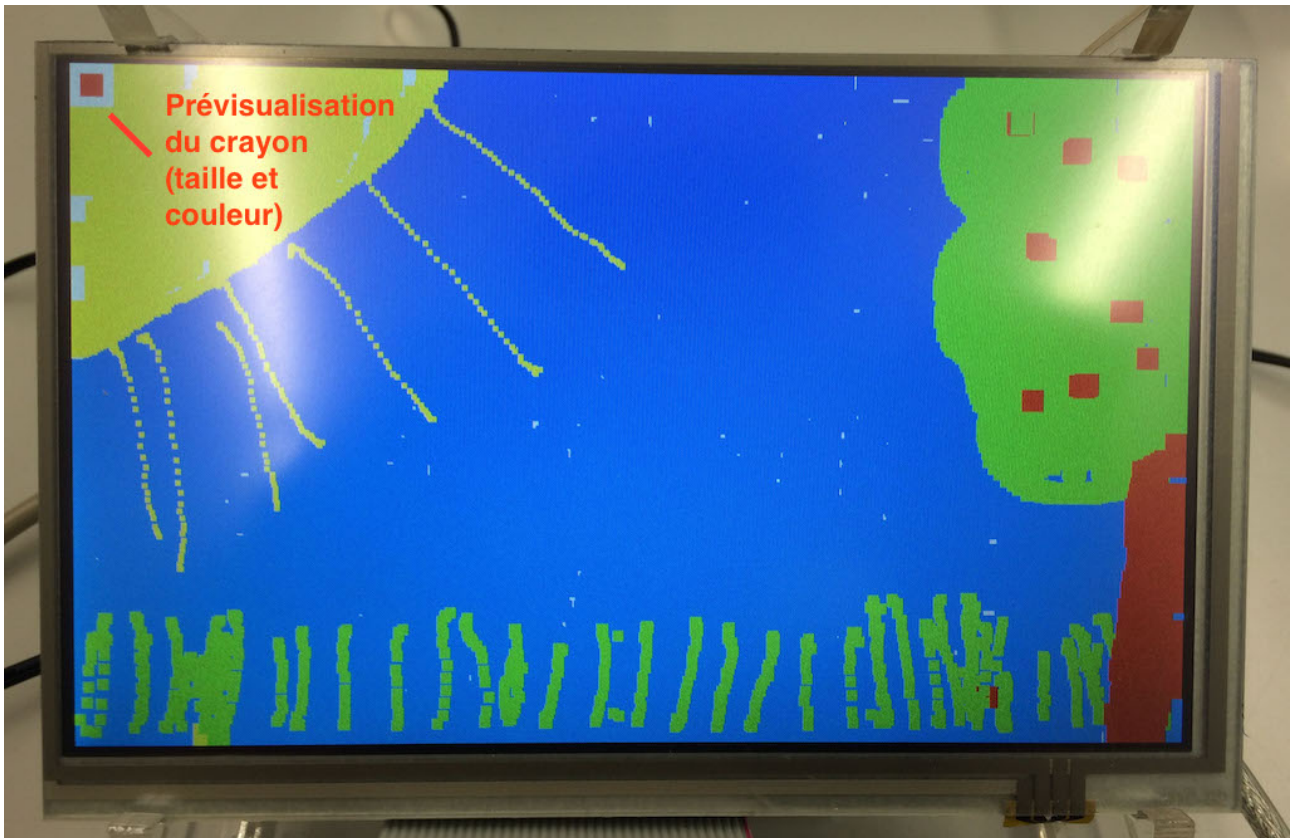


Figure 6: Une petite esquisse à l'aide d'un crayon de tailles et de couleurs différentes

Dès lors que les tests relatifs au crayon sont terminés, nous pouvons effacer l'écran, en appuyant sur le bouton de gauche. L'écran est ensuite effacé et seul l'état du crayon reste visible en haut à gauche.

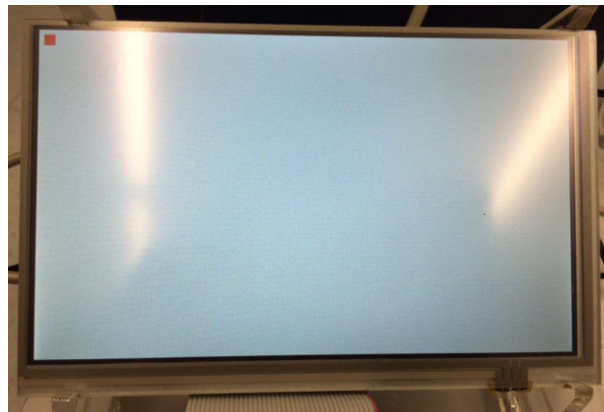


Figure 7: Etat de l'écran LCD après un effacement complet

## 4 ACQUIS

Ce travail nous a permis d'appréhender les concepts suivants :

- Calibration : la calibration à l'aide de trois points nous a permis d'ajuster la coordonnée lue lors de la pression avec la coordonnée propre à l'écran LCD. Pour ce faire, nous avons dû résoudre 6 équations afin de trouver les valeurs des constantes A-F nécessaires à l'ajustement des coordonnées X-Y.
- Génération d'images XPM : la génération d'image XPM sans l'aide de Photoshop et de l'outil sous Ubuntu nous a forcé à comprendre son fonctionnement et à créer des images à partir d'un code couleur sur 16 bits, ainsi qu'une hauteur et une largeur.

## 5 PROBLEMES

Nous avons rencontré quelques problèmes lors de la génération d'un nouveau crayon (taille ou couleur). En effet, en changeant la couleur ou la taille du crayon, les pixels affectés par la dernière pression se voyaient modifiés. En implémentant une machine d'états (états DRAWING, CHANGING\_COLOR, etc), le problème a été réglé.

## 6 PERSPECTIVES

Ce travail pratique laisse entrevoir d'autres types d'application utilisant l'écran LCD et sa partie tactile.

Bien que fonctionnel, le code n'est pas optimal. En effet, l'utilisation d'un timer lors de la phase de calibration aurait été plus adéquate qu'utiliser une boucle « for ».

## 7 CONCLUSION

Avec le temps consacré à ce travail pratique, nous pouvons finalement dire que ce genre de mini-projet prend énormément d'heures pour arriver à optimiser le code et ses fonctions. Nous avons eu beaucoup à réaliser, mais cela en valait la peine. Nous espérons que nos concepts étaient utilisés comme il se doit et que notre projet est efficace ainsi qu'efficient.

## 8 REFERENCES

- Résolution de la matrice de calibration :  
<http://www.embedded.com/design/system-integration/4023968/How-To-Calibrate-Touch-Screens>
- TP5 & TP10 : image XPM & calibration, Samuel Mertenat

## 9 ANNEXE

Le code source du programme se trouve sur Git, à l'adresse : <https://forge.tic.eia-fr.ch/git/samuel.mertenat/se12-tp/tree/master/tp12>.

```

/**
 * Copyright 2015 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:      HEIA-FR / Embedded Systems 2 Laboratory
 *
 * Abstract:     Main module
 *
 * Purpose:      Main module to demonstrate and to test the scribble module
 *
 * Author:       Fabio Valverde & Samuel Mertenat
 * Date:         17.05.2015
 */

#include <imx27_gpio.h>
#include "interrupt.h"
#include "interrupts/exception.h"
#include "interrupts/aitc.h"
#include <imx27_lcdc.h>
#include "drivers/tsc2101.h"
#include "scribble.h"

int main() {
    printf("\n");
    printf("HEIA-FR - Embedded Systems 2 Laboratory\n");
    printf("TP12: Gribouillage\n");
    printf("Fabio Valverde & Samuel Mertenat\n");
    printf("-----\n");
    printf("You can use the GPIO buttons to set the pencil\n");
    printf("    Left:\tClear screen\n");
    printf("    Middle:\tChange the color [RED - GREEN - BLUE - YELLOW -\n");
    printf("        WHITE]\n");
    printf("    Right:\tChange the size [SMALL - MEDIUM - LARGE]\n");
    printf("-----\n");

    // Initialization of the different modules
    imx27_gpio_init();
    imx27_lcdc_init();
    imx27_lcdc_enable();
    interrupt_init();
    exception_init();
    airc_init();
    interrupt_enable();
    tsc2101_init();
    scribble_init();
    return 0;
}

```

```
#ifndef SCRIBBLE_H_
#define SCRIBBLE_H_
/**
 * Copyright 2015 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:      HEIA-FR / Embedded Systems 2 Laboratory
 *
 * Abstract:      Scribble generation
 *
 * Purpose:      Module to allow to scribble on the LCD touchscreen
 *
 * Author:       Fabio Valverde & Samuel Mertenat
 * Date:         17.05.2015
 */

#include <stdint.h>

// Enumeration of possible pencil's colors
enum colors {
    RED =        63488,        // RGB 255-0-0
    GREEN =      34784,        // RGB 128-255-0
    BLUE =       1055,         // RGB 0-128-255
    YELLOW =     65504,        // RGB 255-255-0
    WHITE =      65535         // RGB 255-255-255
};

// Enumeration of possible pencil's widths [pixels]
enum widths {
    SMALL = 4,
    MEDIUM = 8,
    LARGE = 16
};

// Structure to store the components of the xpm_image
struct xpm_image {
    uint8_t width;        // image width
    uint8_t height;       // image height
    uint16_t* img;        // image coded [RGB565]
};

// Enumeration of possible state
enum states {
    DRAWING,
    ERASING,
    CHANGING_COLOR,
    CHANGING_WIDTH
};
```

```
// Pencil's structure
struct pencil {
    uint16_t x;                // x position
    uint16_t y;                // y position
    enum colors color;         // color
    enum widths width;         // width
    struct xpm_image rubber;    // to erase the preview section
    struct xpm_image lead_of_pencil; // xpm image of the pencil
    enum states state;         // the pencil's state
};

// Method to initialize the scribble
extern void scribble_init();

// Method to refresh the LCD
extern void scribble_display_pencil(struct xpm_image *p_img, uint16_t p_x,
    uint16_t p_y);

#endif
```

```
/**
 * Copyright 2015 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:      HEIA-FR / Embedded Systems 2 Laboratory
 *
 * Abstract:     Scribble generation
 *
 * Purpose:      Module to allow to scribble on the LCD touchscreen
 *
 * Author:       Fabio Valverde & Samuel Mertenat
 * Date:         17.05.2015
 */

#include <imx27_lcdc.h>
#include <imx27_gpio.h>
#include "interrupts/aitc.h"
#include "scribble.h"
#include "calibration.h"

#define PREVIEW_SIZE 30      // pencil's preview section (24 x 24)

// Method is called by the PE3 GPIO button to clear the LCD
void scribble_clear_LCD(struct pencil *p_pencil) {
    p_pencil->state = ERASING;
}

// Method is called by the PE4 GPIO button to change the color of the pencil
void scribble_change_color(struct pencil *p_pencil) {
    switch (p_pencil->color) {
        case RED:
            p_pencil->color = GREEN;
            printf("Color : GREEN\n");
            break;
        case GREEN:
            p_pencil->color = BLUE;
            printf("Color : BLUE\n");
            break;
        case BLUE:
            p_pencil->color = YELLOW;
            printf("Color : YELLOW\n");
            break;
        case YELLOW:
            p_pencil->color = WHITE;
            printf("Color : WHITE\n");
            break;
        case WHITE:
            p_pencil->color = RED;
    }
}
```

```
        printf("Color : RED\n");
        break;
    }
    p_pencil->state = CHANGING_COLOR;
}

// Method is called by the PE6 GPIO button to change the width of the pencil
void scribble_change_width(struct pencil *p_pencil) {
    switch (p_pencil->width) {
        case SMALL:
            p_pencil->width = MEDIUM;
            printf("Size : MEDIUM\n");
            break;
        case MEDIUM:
            p_pencil->width = LARGE;
            printf("Size : LARGE\n");
            break;
        case LARGE:
            p_pencil->width = SMALL;
            printf("Size : SMALL\n");
            break;
    }
    p_pencil->state = CHANGING_WIDTH;
}

// Method is called when a pressure occurs on the touchscreen
// This method prints to the coordinates x-y the pencil's representation
void scribble_display_pencil(struct xpm_image *p_img, uint16_t p_x, uint16_t
    p_y) {
    // returns the base address of the LCD
    uint16_t *bitmap = imx27_lcdc_get_bitmap();

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

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

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

// Method is called when the user change the colors or the size of the pencil
// This method creates a new representation of the pencil (xpm_image)
void scribble_update_pencil(struct pencil *p_pencil) {
    // creates and initializes a xpm image
    struct xpm_image xpm = {
        .width = p_pencil->width,
        .height = p_pencil->width,
        .img = 0
    };

    // allocates the needed memory (size x size x 2 bytes)
```

```

xpm.img = malloc(xpm.height * xpm.width * sizeof(*xpm.img));
// codes the new representation [rgb565]
uint16_t* p = xpm.img;
for (uint8_t y = 0; y < xpm.height; y++) {
    for (uint8_t x = 0; x < xpm.width; x++) {
        *p++ = p_pencil->color;
    }
}

// liberates the memory used by the oldest representation and assigns the
// new representation to the pencil
free(p_pencil->lead_of_pencil.img);
p_pencil->lead_of_pencil = xpm;

// if the rubber (for the preview section) isn't created yet, creates the
// representation for its
if (p_pencil->rubber.img == 0) {
    xpm.height = PREVIEW_SIZE;
    xpm.width = PREVIEW_SIZE;
    xpm.img = 0;
    xpm.img = malloc(xpm.height * xpm.width * sizeof(*xpm.img));
    p = xpm.img;
    for (uint8_t y = 0; y < xpm.height; y++) {
        for (uint8_t x = 0; x < xpm.width; x++) {
            *p++ = WHITE;
        }
    }
    p_pencil->rubber = xpm;
}
}

// Method is called when the user change the colors or the size of the pencil
// This method erases the preview section and displays the pencil's
// representation
void scribble_update_pencil_preview(struct pencil *p_pencil) {
    scribble_display_pencil(&p_pencil->rubber, 0, 0);
    p_pencil->x = PREVIEW_SIZE / 2 - p_pencil->lead_of_pencil.width / 2;
    p_pencil->y = PREVIEW_SIZE / 2 - p_pencil->lead_of_pencil.height / 2;
    scribble_display_pencil(&p_pencil->lead_of_pencil, p_pencil->x, p_pencil->
        y);
}

// Method to initialize the scribble
void scribble_init() {
    // creates a red pencil with a medium size
    struct pencil pencil = {
        .color = RED,
        .width = LARGE,
        .lead_of_pencil.img = 0,
        .rubber.img = 0,
        .state = DRAWING
    };
    scribble_update_pencil(&pencil);

    // creates & initializes the matrix used for the calibration
    struct calibration_matrix matrix = calibration_set_matrix(&pencil);

    airc_attach(AIRC_GPIO0, AIRC_IRQ0, imx27_gpio_isr, 0);
    // attaches the buttons to the different functions
    imx27_gpio_attach(IMX27_GPIO_PORT_E, 3, IMX27_GPIO_IRQ_FALLING,

```



```
    scribble_clear_LCD, &pencil);
imx27_gpio_attach(IMX27_GPIO_PORT_E, 4, IMX27_GPIO_IRQ_FALLING,
    scribble_change_color, &pencil);
imx27_gpio_attach(IMX27_GPIO_PORT_E, 6, IMX27_GPIO_IRQ_FALLING,
    scribble_change_width, &pencil);

// enables the buttons
imx27_gpio_enable(IMX27_GPIO_PORT_E, 3);
imx27_gpio_enable(IMX27_GPIO_PORT_E, 4);
imx27_gpio_enable(IMX27_GPIO_PORT_E, 6);

struct tsc2101_position current_p;
struct tsc2101_position last_p = tsc2101_read_position();
uint8_t same_pressure = 0;
scribble_update_pencil_preview(&pencil);    // displays the preview
section

while (1) {
    if (pencil.state == DRAWING) {
        current_p = tsc2101_read_position();
        if (current_p.z == last_p.z) {
            same_pressure++;
            if (same_pressure >= 9) {
                same_pressure = 0;

                // adjusts the coordinates x-y
                pencil.x = calibration_adjust_x(current_p.x, current_p.y,
                    &matrix);
                pencil.y = calibration_adjust_y(current_p.x, current_p.y,
                    &matrix);

                if (pencil.x <= PREVIEW_SIZE && pencil.y <= PREVIEW_SIZE)
                {
                    continue;
                    // displays nothing on the preview section ...
                } else if (pencil.x < pencil.lead_of_pencil.width) {
                    if (pencil.y < pencil.lead_of_pencil.height)
                        scribble_display_pencil(&pencil.lead_of_pencil, 0,
                            0);
                    else if (pencil.y > IMX27_LCD_HEIGHT - pencil.
                        lead_of_pencil.height)
                        scribble_display_pencil(&pencil.lead_of_pencil, 0,
                            IMX27_LCD_HEIGHT - pencil.lead_of_pencil.
                                height);
                    else
                        scribble_display_pencil(&pencil.lead_of_pencil, 0,
                            pencil.y - pencil.lead_of_pencil.height / 2);
                } else if (pencil.x > IMX27_LCD_WIDTH - pencil.
                    lead_of_pencil.width) {
                    if (pencil.y < pencil.lead_of_pencil.height)
                        scribble_display_pencil(&pencil.lead_of_pencil,
                            IMX27_LCD_WIDTH - pencil.lead_of_pencil.width,
                                0);
                    else if (pencil.y > IMX27_LCD_HEIGHT - pencil.
                        lead_of_pencil.height)
                        scribble_display_pencil(&pencil.lead_of_pencil,
                            IMX27_LCD_WIDTH - pencil.lead_of_pencil.width,
                                IMX27_LCD_HEIGHT - pencil.lead_of_pencil.
                                    height);
                    else
```

```
        scribble_display_pencil(&pencil.lead_of_pencil,
                                IMX27_LCD_WIDTH - pencil.lead_of_pencil.width,
                                pencil.y - pencil.lead_of_pencil.height / 2);
    } else {
        if (pencil.y < pencil.lead_of_pencil.height)
            scribble_display_pencil(&pencil.lead_of_pencil,
                                    pencil.x - pencil.lead_of_pencil.width / 2, 0)
            ;
        else if (pencil.y > IMX27_LCD_HEIGHT - pencil.
                  lead_of_pencil.height)
            scribble_display_pencil(&pencil.lead_of_pencil,
                                    pencil.x - pencil.lead_of_pencil.width / 2,
                                    IMX27_LCD_HEIGHT - pencil.lead_of_pencil.
                                    height);
        else
            scribble_display_pencil(&pencil.lead_of_pencil,
                                    pencil.x - pencil.lead_of_pencil.width / 2,
                                    pencil.y - pencil.lead_of_pencil.height / 2);
    }
}

last_p.x = current_p.x;
last_p.y = current_p.y;
last_p.z = current_p.z;
} else if (pencil.state == ERASING) {
    imx27_lcdc_clear_screen();
    scribble_update_pencil_preview(&pencil);
    printf("LCD cleared\n");
    pencil.state = DRAWING;
} else if (pencil.state == CHANGING_COLOR || pencil.state ==
           CHANGING_WIDTH) {
    scribble_update_pencil(&pencil);
    scribble_update_pencil_preview(&pencil);
    pencil.state = DRAWING;
}
}
```

```
#pragma once
#ifndef CALIBRATION_CALIBRATION_H_
#define CALIBRATION_CALIBRATION_H_
/**
 * Copyright 2015 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:      HEIA-FR / Embedded Systems 2 Laboratory
 *
 * Abstract:     TSC2101 - Touch Screen Calibration
 *
 * Purpose:      Module to calibrate the TSC2101 Controller
 *
 * Author:       Fabio Valverde & Samuel Mertenat
 * Date:         17.05.2015
 */

#include <stdint.h>
#include "tsc2101.h"
#include "../scribble.h"

// Structure to store the components of the calibration matrix
struct calibration_matrix {
    double A;
    double B;
    double C;
    double D;
    double E;
    double F;
};

// Method to initialize the calibration
extern struct calibration_matrix calibration_set_matrix(struct pencil *
    p_pencil);

// Method to adjust the coordinate given by the touchscreen
extern uint16_t calibration_adjust_x(uint16_t x, uint16_t y, struct
    calibration_matrix *p_matrix);
extern uint16_t calibration_adjust_y(uint16_t x, uint16_t y, struct
    calibration_matrix *p_matrix);
#endif
```

```
/**
 * Copyright 2015 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:      HEIA-FR / Embedded Systems 2 Laboratory
 *
 * Abstract:     TSC2101 – Touch Screen Calibration
 *
 * Purpose:      Module to calibrate the TSC2101 Controller
 *
 * Author:       Fabio Valverde & Samuel Mertenat
 * Date:         17.05.2015
 */

#include <stdint.h>
#include <stdbool.h>
#include "calibration.h"
#include "tsc2101.h"
#include "../scribble.h"

// Method to calibrate the touchscreen with the LCD
/* Documentation: 24_touchscreen_calibration.pdf.pdf – p7-8 */
struct calibration_matrix calibration_set_matrix(struct pencil *p_pencil) {
    printf("Touchscreen – Three-Point Calibration\n");
    printf("Please, press briefly on the red square on the screen!\n");
    // defines 3 different points to display the squares
    uint16_t dx1 = 350, dy1 = 65;
    uint16_t dx2 = 195, dy2 = 200;
    uint16_t dx3 = 550, dy3 = 350;
    // and defines the touchscreen's variables to store the values returned
    // from the touchscreen for these points
    uint16_t tx1 = 0, ty1 = 0;
    uint16_t tx2 = 0, ty2 = 0;
    uint16_t tx3 = 0, ty3 = 0;

    // initialization of the matrix
    struct calibration_matrix matrix = {
        .A = 0,
        .B = 0,
        .C = 0,
        .D = 0,
        .E = 0,
        .F = 0
    };
};

struct tsc2101_position p = tsc2101_read_position();
uint32_t counter = 0;
```

```

// displays the first square and gets the touchscreen position
scribble_display_pencil(&p_pencil->lead_of_pencil, dx1 - p_pencil->
    lead_of_pencil.width / 2, dy1 - p_pencil->lead_of_pencil.height / 2);
do {
    p = tsc2101_read_position();
    for (counter = 1000000; counter > 0; counter--);
} while (p.x == 0 && p.y == 0);
tx1 = p.x;
ty1 = p.y;
imx27_lcdc_clear_screen();
for (counter = 2000000; counter > 0; counter--);

// and the same for the others
scribble_display_pencil(&p_pencil->lead_of_pencil, dx2 - p_pencil->
    lead_of_pencil.width / 2, dy2 - p_pencil->lead_of_pencil.height / 2);
do {
    p = tsc2101_read_position();
    for (counter = 1000000; counter > 0; counter--);
} while (p.x == tx1 || p.y == ty1);
tx2 = p.x;
ty2 = p.y;
imx27_lcdc_clear_screen();
for (counter = 2000000; counter > 0; counter--);

scribble_display_pencil(&p_pencil->lead_of_pencil, dx3 - p_pencil->
    lead_of_pencil.width / 2, dy3 - p_pencil->lead_of_pencil.height / 2);
do {
    p = tsc2101_read_position();
    for (counter = 1000000; counter > 0; counter--);
} while (p.x == tx2 || p.y == ty2);
tx3 = p.x;
ty3 = p.y;
imx27_lcdc_clear_screen();

double K = 0;
K = (tx1 - tx3) * (ty2 - ty3) - (tx2 - tx3) * (ty1 - ty3);
matrix.A = ((dx1 - dx3) * (ty2 - ty3) - (dx2 - dx3) * (ty1 - ty3)) / K;
matrix.B = ((tx1 - tx3) * (dx2 - dx3) - (dx1 - dx3) * (tx2 - tx3)) / K;
matrix.C = (ty1 * (tx3 * dx2 - tx2 * dx3) + ty2 * (tx1 * dx3 - tx3 * dx1)
    + ty3 * (tx2 * dx1 - tx1 * dx2)) / K;
matrix.D = ((dy1 - dy3) * (ty2 - ty3) - (dy2 - dy3) * (ty1 - ty3)) / K;
matrix.E = ((tx1 - tx3) * (dy2 - dy3) - (dy1 - dy3) * (tx2 - tx3)) / K;
matrix.F = (ty1 * (tx3 * dy2 - tx2 * dy3) + ty2 * (tx1 * dy3 - tx3 * dy1)
    + ty3 * (tx2 * dy1 - tx1 * dy2)) / K;
printf("    A: %f, B: %f, C: %f\n", matrix.A, matrix.B, matrix.C);
printf("    D: %f, E: %f, F: %f\n", matrix.D, matrix.E, matrix.F);
printf("-----\n");
return matrix;
}

/* Documentation: 24_touchscreen_calibration.pdf.pdf - p7-8 */
uint16_t calibration_adjust_x(uint16_t x, uint16_t y, struct
    calibration_matrix *p_matrix) {
    return (x * p_matrix->A + y * p_matrix->B + p_matrix->C);
}

uint16_t calibration_adjust_y(uint16_t x, uint16_t y, struct
    calibration_matrix *p_matrix) {
    return (x * p_matrix->D + y * p_matrix->E + p_matrix->F);
}

```