PIC32_Cpp Documentation

Version 0.1

Samuel Dolt

23 June 2016

Table des matières

CHAPITRE 1

Introduction

1.1 Présentation

Cette librairie contient des drivers objets en C++ pour piloter de manière aisée le starter-kit “PIC32” de l’Ecole Supérieur de l’ETML à Lausanne.

CHAPITRE 2

Installation et guide pour la contribution

2.1 Installation

Pour programmer en C++ sur un kit PIC32, vous devez installer les logiciels suivants :

- Mplab X, version 2 au minimum
- Le compilateur XC32, dans sa dernière version

2.1.1 Pour les utilisateurs

Pour simplement utiliser les librairies, vous pouvez télécharger

2.1.2 Complément pour les contributeurs

Pour l'édition du code : * Github pour windows http://windows.github.com

Pour l'édition de la documentation, il faut installer :

- Python en version 3.4 ou supérieur. Ne pas oublier de cocher la case "ajout python au PATH système"
- Puis, après l'installation de Python, entrer la commande suivante pour installer l'outils de documentation

```
pip install sphinx
pip install sphinx_rtd_theme
```

CHAPITRE 3

Utilisation des fonctions basiques

3.1 Utilisation d'une entrée sortie digitale avec DigitalPin

Dans votre fichier principal, vérifier que le fichier DigitalPin.h est inclus.

```
#include "DigitalPin.h"
```

3.1.1 Initialisation de la classe

Pour utiliser la broche B2, on peux initialiser la classe DigitalPin de deux manière différente :

```
DigitalPin broche_b2 = DigitalPin("B2");
DigitalPin broche_b2 = DigitalPin('B', 2); // Déclaration alternative
```

De manière similaire, pour la broche D15 :

```
DigitalPin broche_d15 = DigitalPin("D15");
DigitalPin broche_d15 = DigitalPin('D', 15); // Déclaration alternative
```

3.1.2 Lecture de l'état logique

Note : Pour lire l'état d'une broche, il est préférable de la mettre en entrée !

```
broche_b2.set_input();
broche_b2.set_direction(INPUT); // Alternative
```

Pour lire l'état logique de la broche B2, il suffis d'appeler sa méthode read.

```
bool value;
value = broche_b2.read();
```

3.1.3 Écriture de l'état logique

Avertissement : Pour imposer un état logique sur une broche, il faut impérativement la configurer en sortie :

```
broche_b2.set_output();
broche_b2.set_direction(OUTPUT); // Alternative
```

Pour imposer l'état haut, écrire un des codes suivant :

```
    broche_b2.set_high();
    broche_b2.write(HIGH); // Alternative
```

Pour imposer l'état haut, écrire un des codes suivant :

```
    broche_b2.set_high();
    broche_b2.write(HIGH); // Alternative
```

Pour imposer l'état bas, écrire un des codes suivant :

```
    broche_b2.set_low();
    broche_b2.write(LOW); // Alternative
```

Pour inverser l'état logique de la broche :

```
    broche_b2.toggle();
```

3.1.4 Gestion des temporisations

On peut introduire des temps de délais en secondes, millisecondes ou microsecondes à l'aide des fonctions du namespace delai. L'exemple suivant montre trois manières d'effectuer une temporisation d'une seconde :

```
delai::s(1);
delai::ms(1000);
delai::us(1000000);
```

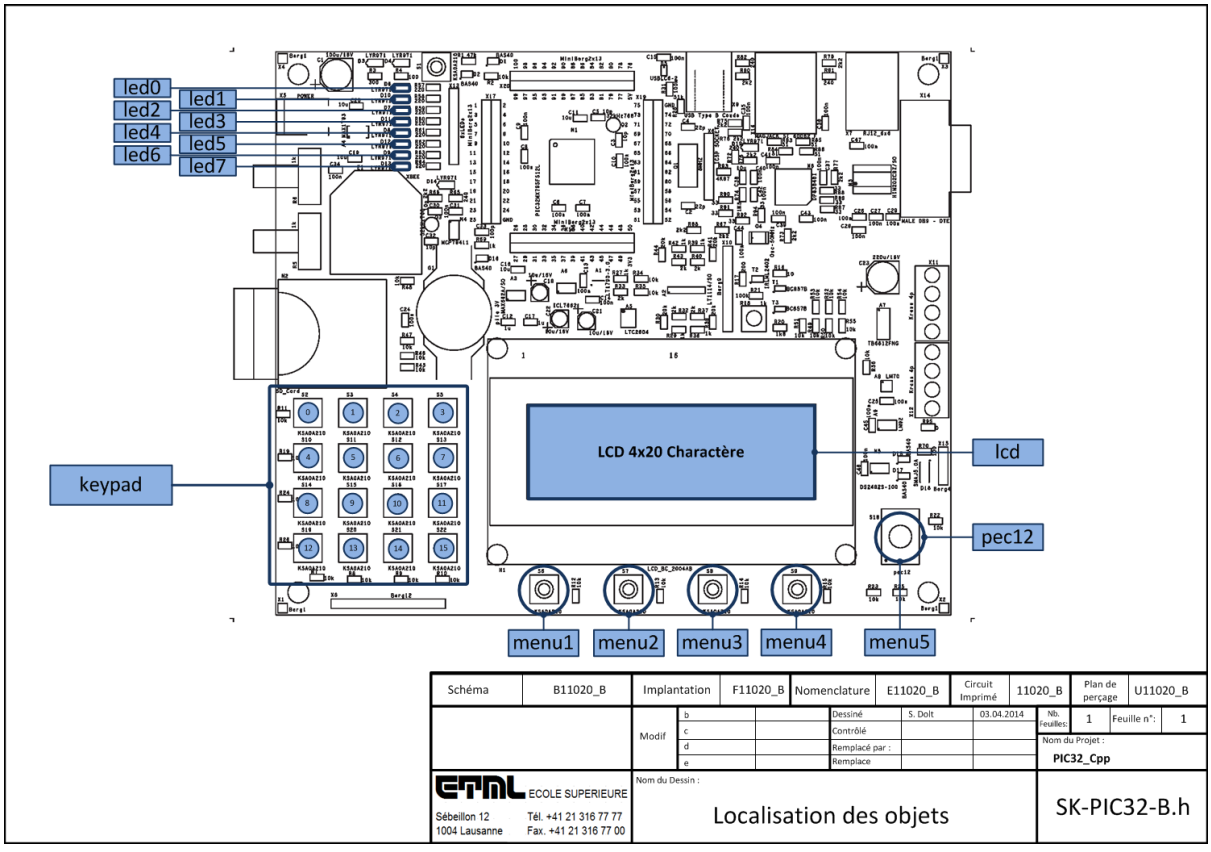
Définition pour le starter kit PIC32 de l'ETML-ES

Le fichier "ETML-ES/SK-PIC32-B.h" contient des instances globales permettant d'accélerer le prototypage. Pour les utiliser, il faut tout d'abord importer les définitions concernant la carte :

```
#include "ETML-ES/SK-PIC32-B.h"
```

4.1 Emplacement

La figure suivante montre la relation entre les noms des instances et les éléments hardware qui s'y rapporte.



Ainsi, on peut écrir sur le LCD avec le code suivant :

```
lcd << "Bonjour";
```

Et ceci, sans avoir à initialiser une instance de la classe TextDisplay.

4.2 Définitions

Les objets définits sont les suivants :

TextDisplay **lcd**
: Affichage LCD 4x20 charactères

Key **menu1**
: Bouton “SEscMenu”

Key **menu2**
: Bouton “S+”

Key **menu3**
: Bouton “S-“

Key **menu4**
: Bouton “S_Ok”

Key **menu5**
: Bouton du PEC12 (PEC12_PB)

IncrementalEncoder **pec12**
: Gestion des rotations du PEC12

Led **led0**
: Led D6

Led **led1**
: Led D10

Led **led2**
: Led D7

Led **led3**
: Led D11

Led **led4**
: Led D8

Led **led5**
: Led D12

Led **led6**
: Led D9

Led **led7**
: Led D13

Keypad **keypad**
: Clavier matriciel, touche S10 à S22

CHAPITRE 5

Utilisation avec le starter kit PIC32 de l'ETML-ES

5.1 Utilisation des LEDS

Pour allumer la led 0, il suffit d'écrir :

```
led0.set_on();
```

De manière intuitive, pour l'éteindre :

```
led0.set_off();
```

Et pour inverser son état :

```
led0.toggle();
```

5.2 Utilisation du LCD

Remarque : Afin d'éviter le clignottement de l'écran, les caractères sont écrit à l'écran uniquement s'il n'y était pas déjà. Il est ainsi plus nécessaire de le faire manuellement.

5.2.1 Écriture

Pour écrire sur le première ligne du LCD, il faut d'abords position le curseur en position (1,1),

```
lcd.set_cursor(1,1);
lcd << "Hello";
```

Pour la position (1,1), le raccourcis suivant est disponible :

```
lcd.home();
```

Pour écrire à partir du cinquième caractère de la ligne deux, il faut écrire :

```
lcd.set_cursor(2,5);
lcd << "world";
```

Une syntaxe allégée est aussi disponible :

```
lcd << cursor(2,5) << "world";
```

5.2.2 Mot clef et conversion

Le retour à la ligne peut se faire automatique de deux manière :

```
lcd << endl;
lcd << '\n'; // Alernative
```

Pour afficher un nombre en héxadécimal, il faut entrer :

```
lcd << hex << 125;
```

On peut revenir au mode décimale avec :

```
lcd << dec;
```

Pour afficher les chiffres positifs avec un signe plus en mode décimale

```
lcd << with_sign_plus << 10; // Affiche +10
lcd << without_sign_plus; // Désactive l'affichage du signe
```

Pour fixer la longueur du prochain paramètre, on utilise setwritemode

```
lcd << setw(4) << "ABC"; // Avec setw(4), un espace est rajouté après le 'C'
```

5.2.3 Éffaçage de l'écran

La méthode clear permet d'effacer les caractères afficher.

```
lcd.clear()
```

5.2.4 Gestion du rétro-éclairage

Le rétro-éclairage peut être éteint puis rallumer avec le code suivant :

```
lcd.disable_backlight()
delay::ms(5000);
lcd.enable_backlight();
```

5.2.5 Désactivation de l'écran

La méthode disable_display permet de désactiver l'écran sans perdre les caractères qui y sont affiché. La méthode enable_display remet l'écran dans sont état normal.

```
lcd.disable_display();
delay::ms(5000);
lcd.enable_display();
```

5.2.6 Gestion du curseur

Le curseur peut être afficher de deux manière.

Comme un tirait en bas :

```
lcd.enable_underline_cursor();
delay::ms(5000);
lcd.disable_underline_cursor();
```

Avec un carré noir clignottant :

```
lcd.enable_blinking_cursor();
delay::ms(5000);
lcd.disable_blinking__cursor();
```

5.3 Utilisation des touches

Pour vérifier si une touche est appuyée :

```
if(menu1.is_pressed())
{
    lcd << "Touche appuyée";
}
```

Pour vérifier si une touche est relachée :

```
if(menu1.is_relached())
{
    lcd << "Touche relachée";
}
```

Pour vérifier si une touche a un nouvel état :

```
if(menu1.has_a_new_state())
{
    lcd << "L'état a changé";
}
```

5.4 Utilisation du PEC12

Pour obtenir la direction de la dernière rotation :

```
int8_t dir = pec12.get_state();

if(dir == +1)
{
     lcd << "Rotation de le sens horaire";
}
else if(dir == -1)
{
     lcd << "Rotation de le sens anti-horaire";
}
else
{
    lcd << "Personne a utilisé le PEC12";
}
```

Pour vérifier si le PEC a un nouvel état :

```
if(pec12.has_a_new_state())
{
    lcd << "Le PEC a bougé";
}
```

5.5 Utilisation du clavier matriciel

CHAPITRE 6

Fichier Delay.h

La bibliothèque Delay permet d'utiliser des temporisations et des temps d'attente. Pour se faire, la bibliothèque utilise le CoreTimer du PIC32.

6.1 Fonctions

void `delay::s`(uint32_t *delay*)

Paramètresuint32_t delay – Nombre de s à attendre

void `delay::ms`(uint32_t *delay*)

Paramètresuint32_t delay – Nombre de ms à attendre

void `delay::us`(uint32_t *delay*)

Paramètresuint32_t delay – Nombre de µs à attendre

CHAPITRE 7

Gestion des entrées/sorties digitales

Les entrées/sorties numériques sont implémenté dans le fichier hw/DigitalPin.h qu'il faut tous d'abord inclure :

```
#include "DigitalPin.h"

// Mise à l'état au de la broche RB6
DigitalPin IO_B6 = DigitalPin("B6");
IO_B6.set_direction(OUTPUT);
IO_B6.write(LOW);

// Lecture de la broche RB6
DigitalPin IO_B7 = DigitalPin("B7");
bool value;

IO_B6.set_direction(INPUT);
value = IO_B6.read();
```

7.1 Description

class `DigitalPin`

`DigitalPin` (**const** char *PIN*[])

Paramètres `PIN` – Chaîne décrivant la broche utilisée

La broche doit être dans un chaîne du type "A1" ou "B12". Elles signifies port A, broche 1 et port B broche 12.

`DigitalPin` (**const** char *PORT*, uint16_t *PIN_NUMBER*)

Paramètres

—`PORT` – Caractère nommant le port, par exemple 'B'

—`PIN_NUMBER` – Numéro de la pin dans le port

`void set_direction(enum direction)`

Configure la broche en entrée ou en sortie

Paramètres `direction` – INPUT ou OUTPUT

void `set_input` (void)

Configure la broche en entrée

void `set_output` (void)

Configure la broche en sortie

void `write` (bool *STATE*)

Met la broche à l'état indiqué.

Paramètres `STATE` – HIGH (1) ou LOW (0)

void `set_low` (void)

Met la broche à l'état bas

void **set_high**(void)
Met la broche à l'état haut

void **toggle**(void)
Inverse la sortie.

bool **read**(void)
Lis l'état du port

Retourne0 pour l'état bas, 1 pour l'état haut

CHAPITRE 8

Listes et tables

— genindex
— modindex
— search

D

K

L

M

P