Quelques apprentissages de langages par l'exemple

1 Introduction

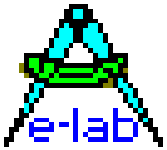
Ce document se propose de guider la création de la base matérielle

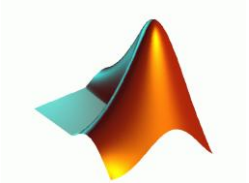
du premier projet sur PSoC.

Refaire le bandeau delphi / matlab / C / PSoC

PSoC Creator utilise GCC pour les familles de composants PSoC5 et PSoC5 LP

PSoC Creator utilise le compilateur Keil pour les familles de composants PSoC3





Ressource C :

http://en.wikipedia.org/wiki/C_standard_library

http://www.cplusplus.com/reference/cstdio/sprintf/

Version du 3 Novembre 2013

1 Manipulations des caractères et des chaines de caractères

1. Le type caractère et les chaines de caractères

- En C, il n'existe pas de type de variable pour les chaînes de caractères comme il en existe pour les entiers (int) ou pour les caractères (char). Les chaînes de caractères sont en fait stockées dans un tableau de char dont la fin est marquée par un caractère nul, de valeur 0 et représenté par le caractère '\0'. En mémoire la chaîne "Bonjour" est représentée ainsi :

B	o	n	j	o	u	r	\0

- La nature de tableau d'une chaîne de caractères (ie, l'adresse en mémoire du premier élément) interdit des affections du type A= "hello" en dehors de la phase d'initialisation.

  char A[ ]="Hello"; est correct mais

  char A[6]; A= "Hello"; ne l'est pas.

- Il faut toujours garder à l'esprit qu'il faut réserver un caractère pour le caractère de terminaison de la chaine. Ainsi "Bonjour" doit être stocké dans un tableau d'au moins 7 caractères de long.

- Les opérations sur les chaines de caractères se font avec les fonctions déclarées dans la bibliothèque <string.h>

  - http://en.wikipedia.org/wiki/String.h

Exemple de manipulations de chaines de caractères avec PSoC

```
char TableChar[20]="Start => Appui BP1";
char UneLigne[20];
char str1[50] = "Hello ";
char str2[50] = "World!";
```


```
// Déclaration de variables locales
float d;
float q=0.0012207;
char tstr[16];
uint16 Nx;
```

```
// Affichage du message d'accueil
CharLCD_Start();

CharLCD_Position(0,0);
strcat(str1,str2);
CharLCD_PrintString(str1);

CharLCD_Position(1,0);
sprintf(tstr, "%d", strlen(TableChar));
CharLCD_PrintString("Longueur : ");CharLCD_PrintString(tstr);
CharLCD_PutChar(32);CharLCD_PutChar(TableChar[2]);


CharLCD_Position(2,0);
sprintf(tstr, "%d", sizeof(TableChar));
CharLCD_PrintString("Longueur totale : ");
CharLCD_PrintString(tstr);


CharLCD_Position(3,0);
CharLCD_PrintString("Start => Appui BP1");

while (BP1_Read()== REPOS ) {};
```

Exemple de la réception de caractères via l'UART

```
// Déclarations pour la réception de caractères
char8 ch;
uint8 count=0;
char8 tampon[16];
```

```
// Test de la réception d'un caractère
// Réception avec l'API
ch = UART_1_GetChar();

// Si il y a un caractère reçu il faut l'accumuler dans un tableau
if( ch > 0)
{
    tampon[count]=ch; // le caractère est recopié dans le tableau
    count++;          // on incrémente le compteur - pointeur de caractères
    }

// Si le caractère reçu est le caractère de fin 0x0D ( retour chariot )
// Alors on affiche le message
if ( ch == 0x0D )
{
    CharLCD_Position(0,0);
    // Ecriture du message
    for (i=0;i<count-1;i++) CharLCD_PutChar(tampon[i]);
    // On efface le reste de la ligne du LCD
    for (i=count+3;i<20;i++) CharLCD_PutChar(' ');
    count=0;
    }
```

uint8 UART_GetChar(void)

Description:	Returns the last received byte of data. UART_GetChar() is designed for ASCII characters and returns a uint8 where 1 to 255 are values for valid characters and 0 indicates an error occurred or no data is present.
Parameters:	void
Return Value:	uint8: Character read from UART RX buffer. ASCII character values from 1 to 255 are valid. A returned zero signifies an error condition or no data available.
Side Effects:	None

Exemple de déclaration multiple entier 16 bits / double char

```
union composite {
    uint16 mot;
    struct {
        char lo;
        char hi;
    } octet;
};

union composite Nx;
```

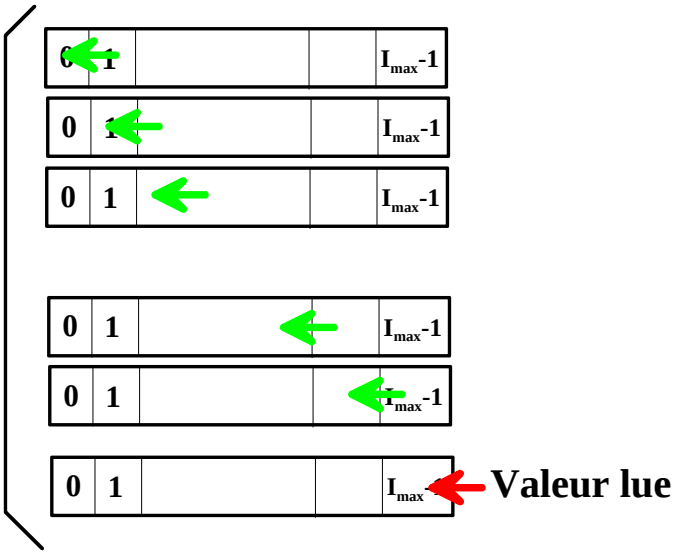
```
//Lancement conversion
ADC_SAR_1_StartConvert();
//attente fin de conversion
ADC_SAR_1_IsEndConversion(ADC_SAR_1_WAIT_FOR_RESULT);
Nx.mot=ADC_SAR_1_GetResult16();

// Affichage du nombre Nx
CharLCD_Position(1,0);
CharLCD_PrintString("Nx     ");
sprintf(tstr, "%6d", Nx.mot );
CharLCD_PrintString(tstr);


// Envoi via l'UART à l'IHM en Delphi
UART_1_WriteTxData(0x40);
UART_1_WriteTxData(0x02);
UART_1_WriteTxData(Nx.octet.hi);
UART_1_WriteTxData(Nx.octet.lo);
UART_1_PutString(tstr);
UART_1_WriteTxData(0x0D);
```

Faire une moyenne glissante

! Bien penser à mettre toute la table à 0 dans la phase d'initialisation du programme.

Table[I_{max}]

Glisse des valeurs




```
// Définition pour la moyenne glissante
#define Nombre_Element 40
uint32 Table[Nombre_Element];
int8 Indice_Boucle;
uint32 Valeur_Moyenne=0;


void Moyenne_Glissante(uint16 Valeur_Lue)
{
    uint32 Somme=0;

    // Décalage de la table d'un élément vers les indices faibles
    // La nouvelle valeur lue prend sa place en indice max
    for (Indice_Boucle=0; Indice_Boucle<Nombre_Element-1; Indice_Boucle++)
    {
        Table[Indice_Boucle]=Table[Indice_Boucle+1];
        }
    Table[Nombre_Element-1]=Valeur_Lue;

    // Calcul de la valeur moyenne glissante
    for (Indice_Boucle=0; Indice_Boucle<Nombre_Element; Indice_Boucle++)
    {
        Somme = Somme + Table[Indice_Boucle];
        }

    Valeur_Moyenne = Somme / Nombre_Element;

}
```

! Réfléchir aux types retenus pour les calculs, en effet un nombre déclaré uint16 ne peut accumuler que 65535 au maximum. D'autre part si on utilise dans les boucles des valeurs d'indices décroissantes il faut pouvoir faire les soustractions donc le type doit être déclaré int et non pas uint.

Quelques éléments spécifiques au PSoC

Base Types

Type	Description
char8	8-bit (signed or unsigned, depending on the compiler selection for char)
uint8	8-bit unsigned
uint16	16-bit unsigned
uint32	32-bit unsigned
int8	8-bit signed
int16	16-bit signed
int32	32-bit signed

Hardware Register Types

Hardware registers typically have side effects and therefore are referenced with a volatile type.

Define	Description
reg8	Volatile 8-bit unsigned
reg16	Volatile 16-bit unsigned
reg32	Volatile 32-bit unsigned

void CyDelay(uint32 milliseconds)

Description: Delay by the specified number of milliseconds. By default the number of cycles to delay is calculated based on the clock configuration entered in PSoC Creator. If the clock configuration is changed at run-time, then the function CyDelayFreq is used to indicate the new Bus Clock frequency. CyDelay is used by several components, so changing the clock frequency without updating the frequency setting for the delay can cause those components to fail.

Parameters: milliseconds: Number of milliseconds to delay.

Return Value: None

Side Effects and Restrictions: CyDelay has been implemented with the instruction cache assumed enabled. When instruction cache is disabled on PSoC 5, CyDelay will be two times larger. For example, with instruction cache disabled CyDelay(100) would result in about 200 ms delay instead of 100 ms.

void CyDelayUs(uint16 microseconds)

Description: Delay by the specified number of microseconds. By default the number of cycles to delay is calculated based on the clock configuration entered in PSoC Creator. If the clock configuration is changed at run-time, then the function CyDelayFreq is used to indicate the new Bus Clock frequency. CyDelayUs is used by several components, so changing the clock frequency without updating the frequency setting for the delay can cause those components to fail.

Parameters: microseconds: Number of microseconds to delay.

Return Value: Void

Side Effects and Restrictions: CyDelayUS has been implemented with the instruction cache assumed enabled. When instruction cache is disabled on PSoC 5, CyDelayUs will be two times larger. For example, with instruction cache disabled CyDelayUs(100) would result in about 200 us delay instead of 100 us.

void CyDelayFreq(uint32 freq)

Description: Sets the Bus Clock frequency used to calculate the number of cycles needed to implement a delay with CyDelay. By default the frequency used is based on the value determined by PSoC Creator at build time.

Parameters: freq: Bus clock frequency in Hz.

0: Use the default value

non-0: Set frequency value

Return Value: None

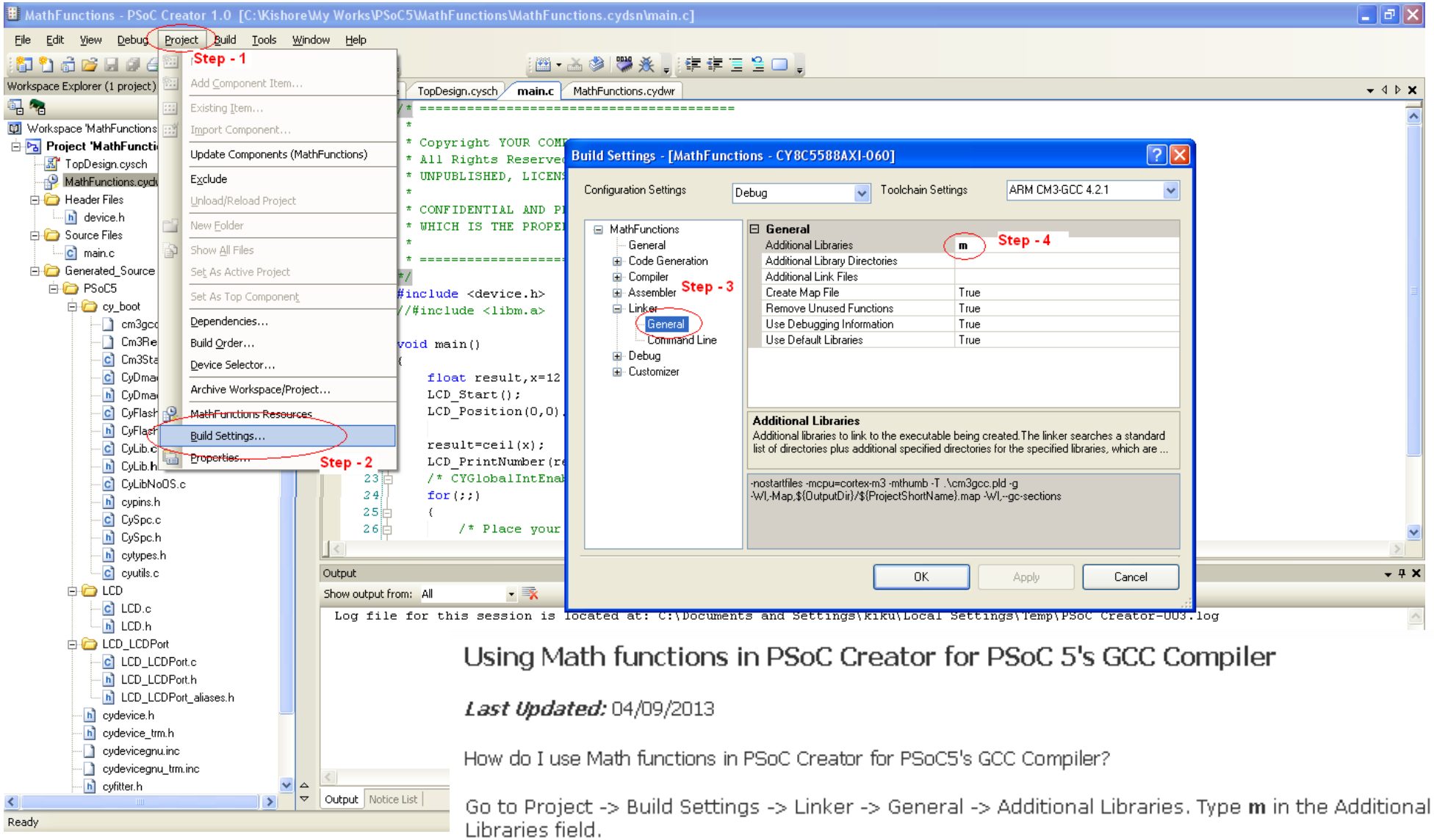
void CyDelayCycles(uint32 cycles)

Description: Delay by the specified number of cycles using a software delay loop.

Parameters: cycles: Number of cycles to delay.

Return Value: None

Pour utiliser des fonctions mathématiques



Using Math functions in PSoC Creator for PSoC 5's GCC Compiler

Last Updated: 04/09/2013

How do I use Math functions in PSoC Creator for PSoC5's GCC Compiler?

Go to Project -> Build Settings -> Linker -> General -> Additional Libraries. Type **m** in the Additional Libraries field.
If you are not adding this Additional Library then you will get the following Build error "*undefined reference to `sqrt*"' where sqrt is a math function.

Quelques questions réponses

- Les PSoC 5 sont programmés avec le compilateur GCC, celui-ci suit la norme ANSI C aussi de la documentation peut être obtenue sur des sites généraux sur le langage c par exemple :

  http://en.wikipedia.org/wiki/C_standard_library

- Pour afficher par exemple un nombre de type float il faut procéder en deux temps :
  - Ecrire la chaine dans un tableau de caractères avec sprintf

    sprintf(tstr, "%+4.2f", compValue);

  - Puis envoyer la chaine obtenue sur l'afficheur :

    CharLCD_PrintString(tstr);

- Utilisation de printf sprintf : voir http://www.cplusplus.com/reference/cstdio/printf/

En C, il est nécessaire d'inclure l'en-tête standard `<stdio.h>` au début du code source du programme, car c'est lui qui permet de déclarer la fonction printf.

Type	Lettre
int	%d
long	%ld
float/double	%f / %lf
char	%c
string (char*)	%s
pointeur (void*)	%p
entier hexadécimal	%x

```
%[flags][width][.precision][length]specifier
```

specifier	Output	Example
d *or* i	Signed decimal integer	392
u	Unsigned decimal integer	7235
o	Unsigned octal	610
x	Unsigned hexadecimal integer	7fa
X	Unsigned hexadecimal integer (uppercase)	7FA
f	Decimal floating point, lowercase	392.65
F	Decimal floating point, uppercase	392.65
e	Scientific notation (mantissa/exponent), lowercase	3.9265e+2
E	Scientific notation (mantissa/exponent), uppercase	3.9265E+2
g	Use the shortest representation: %e or %f	392.65
G	Use the shortest representation: %E or %F	392.65
a	Hexadecimal floating point, lowercase	-0xc.90fep-2
A	Hexadecimal floating point, uppercase	-0XC.90FEP-2
c	Character	a
s	String of characters	sample
p	Pointer address	b8000000
n	Nothing printed. The corresponding argument must be a pointer to a signed int. The number of characters written so far is stored in the pointed location.	
%	A % followed by another % character will write a single % to the stream.	%

The *format specifier* can also contain sub-specifiers: *flags*, *width*, *.precision* and *modifiers* (in that order), which are optional and follow these specifications:

flags	**description**
-	Left-justify within the given field width; Right justification is the default (see *width* sub-specifier).
+	Forces to preceed the result with a plus or minus sign (+ or -) even for positive numbers. By default, only negative numbers are preceded with a - sign.
(space)	If no sign is going to be written, a blank space is inserted before the value.
#	Used with `o`, `x` or `X` specifiers the value is preceeded with `0`, `0x` or `0X` respectively for values different than zero. Used with `a`, `A`, `e`, `E`, `f`, `F`, `g` or `G` it forces the written output to contain a decimal point even if no more digits follow. By default, if no digits follow, no decimal point is written.
0	Left-pads the number with zeroes (`0`) instead of spaces when padding is specified (see *width* sub-specifier).

width	**description**
(number)	Minimum number of characters to be printed. If the value to be printed is shorter than this number, the result is padded with blank spaces. The value is not truncated even if the result is larger.
*	The *width* is not specified in the *format* string, but as an additional integer value argument preceding the argument that has to be formatted.

.precision	description
.number	For integer specifiers (`d`, `i`, `o`, `u`, `x`, `X`): *precision* specifies the minimum number of digits to be written. If the value to be written is shorter than this number, the result is padded with leading zeros. The value is not truncated even if the result is longer. A *precision* of `0` means that no character is written for the value `0`. For `a`, `A`, `e`, `E`, `f` and `F` specifiers: this is the number of digits to be printed **after** the decimal point (by default, this is 6). For `g` and `G` specifiers: This is the maximum number of significant digits to be printed. For `s`: this is the maximum number of characters to be printed. By default all characters are printed until the ending null character is encountered. If the period is specified without an explicit value for *precision*, `0` is assumed.
`.*`	The *precision* is not specified in the *format* string, but as an additional integer value argument preceding the argument that has to be formatted.

```c
/* printf example */
#include <stdio.h>

int main()
{
   printf ("Characters: %c %c \n", 'a', 65);
   printf ("Decimals: %d %ld\n", 1977, 650000L);
   printf ("Preceding with blanks: %10d \n", 1977);
   printf ("Preceding with zeros: %010d \n", 1977);
   printf ("Some different radices: %d %x %o %#x %#o \n", 100, 100, 100, 100, 100);
   printf ("floats: %4.2f %+.0e %E \n", 3.1416, 3.1416, 3.1416);
   printf ("Width trick: %*d \n", 5, 10);
   printf ("%s \n", "A string");
   return 0;
}
```

Output:

```
Characters: a A
Decimals: 1977 650000
Preceding with blanks:       1977
Preceding with zeros: 0000001977
Some different radices: 100 64 144 0x64 0144
floats: 3.14 +3e+000 3.141600E+000
Width trick:    10
A string
```

Un caractère pour l'affichage ou l'impression ou le transfert de données au format texte

	Pascal	C	Matlab	PSoC
Un caractère ASCII	Char	char	Pas de déclaration	char8
Exemple	Lettre : Char ; Lettre := 'A' ; Lettre := chr(65) ; Lettre := chr($41) ;	char lo ;	ch1 = 'bon' ch2 = 'jour' ch = [ch1,ch2] ch3 = 'soi'; ch = [ch(1:3), ch3, ch(7)] ch contient 'bonsoir'	char8 ch;
Tableau de caractères	DoubleDigit : string[2]; Reponse : string;	char tstr[16];		char8 tampon[16]; uint16 Temp; sprintf(tstr,"%+4.2f",0.125*Temp); CharLCD_PrintString(tstr);

Entier non signés

Entiers non signés	Pascal	C	Matlab	PSoC
8 bits 16 bits 32 bits	Byte Word	unsigned char unsigned short int unsigned long int	Sans objet ce sont des réels	uint8 uint16 uint32
Exemple				
Tableau de caractères				

Entier signés

Entiers signés	Pascal	C	Matlab	PSoC
8 bits 16 bits 32 bits	ShortInt Integer	char short int long int	Sans objet ce sont des réels	int8 int16 int32
Exemple				
Tableau de caractères				

Les nombres réels

Les réels	Pascal	C	Matlab	PSoC
32 bits 64 bits	real double	float double	Déclaration directe x = 2 ;	Idem c
Exemple				
Tableau de caractères				

Les variables logiques

Les booléens	Pascal	C	Matlab	PSoC
	Boolean Valeurs possible true / false	N'existe pas Sauf norme C99 Ou type défini spécifiquement		Idem c
Exemple				
Tableau de caractères				

Types entiers	Domaine
Byte	0..255
Shortint	-128..127
Integer	-32768..32767
Word	0..65535
Longint	-2147483648..2147483647

Types caractères	Domaine
Char	Caractère alphanumérique
String[n]	Chaîne de n caractères (n = 255 au maximum)
String	Chaîne de 255 caractères

Types réels	Domaine
Single	$1,5.10^{-45}..3,4.10^{38}$
Real	$2,9.10^{-39}..1,7.10^{38}$
Double	$5,0.10^{-324}..1,7.10^{308}$
Extended	$3,4.10^{-4951}..1,1.10^{4932}$

Type booléen	Domaine
Boolean	True | False

Vous pouvez travailler avec ces caractères avec la fonction chr(xx) où xx est le numéro du caractère voulu pris dans la table ASCII :

writeln(chr($41)); écrit la lettre A

Programmation_C

type	Taille (n bits)	domaine de valeurs (-2^{n-1} à $2^{n-1} - 1$)
short int	16 bits	-32 768 à 32 767
int	16 ou 32 bits	
long int	32 bits	-2 147 483 648 à 2 147 483 647

type	Taille (n bits)	domaine de valeurs (0 à $2^n - 1$)
unsigned short int	16 bits	0 à 65 535
unsigned int	16 ou 32 bits	
unsigned long int	32 bits	0 à 4 294 967 295

type	taille	domaine de valeurs *(pour les valeurs positives)*
float	32 bits	$3.4E^{-38}$ à $3.4E^{38}$
double	64 bits	$1.7E^{-308}$ à $1.7E^{308}$
long double	64 ou 80 bits	

Les caractères sont de type char.

Les caractères sont représentés en mémoire sur 8 bits :

- domaine de valeurs du type char de -128 à 127 ;
- domaine de valeurs du type unsigned char de 0 à 255.

PSoC_Creator_system_reference_guide.pdf

Base Types

Type	Description
char8	8-bit (signed or unsigned, depending on the compiler selection for char)
uint8	8-bit unsigned
uint16	16-bit unsigned
uint32	32-bit unsigned
int8	8-bit signed
int16	16-bit signed
int32	32-bit signed

Hardware Register Types

Hardware registers typically have side effects and therefore are referenced with a volatile type.

Define	Description
reg8	Volatile 8-bit unsigned
reg16	Volatile 16-bit unsigned
reg32	Volatile 32-bit unsigned

void CyDelay(uint32 milliseconds)

Description: Delay by the specified number of milliseconds. By default the number of cycles to delay is calculated based on the clock configuration entered in PSoC Creator. If the clock configuration is changed at run-time, then the function CyDelayFreq is used to indicate the new Bus Clock frequency. CyDelay is used by several components, so changing the clock frequency without updating the frequency setting for the delay can cause those components to fail.

Parameters: milliseconds: Number of milliseconds to delay.

Return Value: None

Side Effects and Restrictions: CyDelay has been implemented with the instruction cache assumed enabled. When instruction cache is disabled on PSoC 5, CyDelay will be two times larger. For example, with instruction cache disabled CyDelay(100) would result in about 200 ms delay instead of 100 ms.

void CyDelayUs(uint16 microseconds)

Description: Delay by the specified number of microseconds. By default the number of cycles to delay is calculated based on the clock configuration entered in PSoC Creator. If the clock configuration is changed at run-time, then the function CyDelayFreq is used to indicate the new Bus Clock frequency. CyDelayUs is used by several components, so changing the clock frequency without updating the frequency setting for the delay can cause those components to fail.

Parameters: microseconds: Number of microseconds to delay.

Return Value: Void

Side Effects and Restrictions: CyDelayUS has been implemented with the instruction cache assumed enabled. When instruction cache is disabled on PSoC 5, CyDelayUs will be two times larger. For example, with instruction cache disabled CyDelayUs(100) would result in about 200 us delay instead of 100 us.

void CyDelayFreq(uint32 freq)

Description: Sets the Bus Clock frequency used to calculate the number of cycles needed to implement a delay with CyDelay. By default the frequency used is based on the value determined by PSoC Creator at build time.

Parameters: freq: Bus clock frequency in Hz.

0: Use the default value

non-0: Set frequency value

Return Value: None

void CyDelayCycles(uint32 cycles)

Description: Delay by the specified number of cycles using a software delay loop.

Parameters: cycles: Number of cycles to delay.

Return Value: None

Matlab

```
% Lecture des données températures via RS232
% Les données ont le format TP LM75
% 0x40 LM75 Temp.hi Temp.lo checksum Terminaison
%
clear all
close all
s = serial ('COM3');
set(s,'BaudRate',9600);
set(s,'FlowControl','none');
set(s,'Terminator',13);
set(s,'Parity','none');
set(s,'StopBits',1);
fopen(s);
trame=fread(s,6);
fclose(s);

% Affichage du résultat dans une fenêtre de dialogue
%
% calcul de la valeur de la température
temp = ( (trame(3)*16 + trame(4))*0.125 );
% conversion en chaine de caractère
strtemp = num2str(temp);
% Préparation du texte final
texte = strcat ( strtemp, '°C');
% Affichage dans une boite de dialogue
msgbox(texte,'Mesure PSoC')
```

Langage c la fonction printf

http://www.cplusplus.com/reference/cstdio/printf/

`format`

C string that contains the text to be written to stdout.
It can optionally contain embedded *format specifiers* that are replaced by the values specified in subsequent additional arguments and formatted as requested.

A *format specifier* follows this prototype: [see compatibility note below]

```
%[flags][width][.precision][length]specifier
```

Where the *specifier character* at the end is the most significant component, since it defines the type and the interpretation of its corresponding argument:

specifier	Output	Example
d *or* i	Signed decimal integer	392
u	Unsigned decimal integer	7235
o	Unsigned octal	610
x	Unsigned hexadecimal integer	7fa
X	Unsigned hexadecimal integer (uppercase)	7FA
f	Decimal floating point, lowercase	392.65
F	Decimal floating point, uppercase	392.65
e	Scientific notation (mantissa/exponent), lowercase	3.9265e+2
E	Scientific notation (mantissa/exponent), uppercase	3.9265E+2
g	Use the shortest representation: %e or %f	392.65
G	Use the shortest representation: %E or %F	392.65
a	Hexadecimal floating point, lowercase	-0xc.90fep-2
A	Hexadecimal floating point, uppercase	-0XC.90FEP-2
c	Character	a
s	String of characters	sample
p	Pointer address	b8000000
n	Nothing printed. The corresponding argument must be a pointer to a signed int. The number of characters written so far is stored in the pointed location.	
%	A % followed by another % character will write a single % to the stream.	%

The *format specifier* can also contain sub-specifiers: *flags*, *width*, *.precision* and *modifiers* (in that order), which are optional and follow these specifications:

flags	description
-	Left-justify within the given field width; Right justification is the default (see *width* sub-specifier).
+	Forces to preceed the result with a plus or minus sign (+ or -) even for positive numbers. By default, only negative numbers are preceded with a - sign.
(space)	If no sign is going to be written, a blank space is inserted before the value.
#	Used with `o`, `x` or `X` specifiers the value is preceeded with `0`, `0x` or `0X` respectively for values different than zero. Used with `a`, `A`, `e`, `E`, `f`, `F`, `g` or `G` it forces the written output to contain a decimal point even if no more digits follow. By default, if no digits follow, no decimal point is written.
0	Left-pads the number with zeroes (`0`) instead of spaces when padding is specified (see *width* sub-specifier).

width	description
(number)	Minimum number of characters to be printed. If the value to be printed is shorter than this number, the result is padded with blank spaces. The value is not truncated even if the result is larger.
*	The *width* is not specified in the *format* string, but as an additional integer value argument preceding the argument that has to be formatted.

.precision	description
.number	For integer specifiers (`d`, `i`, `o`, `u`, `x`, `X`): *precision* specifies the minimum number of digits to be written. If the value to be written is shorter than this number, the result is padded with leading zeros. The value is not truncated even if the result is longer. A *precision* of `0` means that no character is written for the value `0`. For `a`, `A`, `e`, `E`, `f` and `F` specifiers: this is the number of digits to be printed **after** the decimal point (by default, this is 6). For `g` and `G` specifiers: This is the maximum number of significant digits to be printed. For `s`: this is the maximum number of characters to be printed. By default all characters are printed until the ending null character is encountered. If the period is specified without an explicit value for *precision*, `0` is assumed.
.*	The *precision* is not specified in the *format* string, but as an additional integer value argument preceding the argument that has to be formatted.

Example

```c
/* printf example */
#include <stdio.h>

int main()
{
   printf ("Characters: %c %c \n", 'a', 65);
   printf ("Decimals: %d %ld\n", 1977, 650000L);
   printf ("Preceding with blanks: %10d \n", 1977);
   printf ("Preceding with zeros: %010d \n", 1977);
   printf ("Some different radices: %d %x %o %#x %#o \n", 100, 100, 100, 100, 100);
   printf ("floats: %4.2f %+.0e %E \n", 3.1416, 3.1416, 3.1416);
   printf ("Width trick: %*d \n", 5, 10);
   printf ("%s \n", "A string");
   return 0;
}
```

Output:

```
Characters: a A
Decimals: 1977 650000
Preceding with blanks:       1977
Preceding with zeros: 0000001977
Some different radices: 100 64 144 0x64 0144
floats: 3.14 +3e+000 3.141600E+000
Width trick:    10
A string
```

Trigonometric functions

cos	Compute cosine (function)
sin	Compute sine (function)
tan	Compute tangent (function)
acos	Compute arc cosine (function)
asin	Compute arc sine (function)
atan	Compute arc tangent (function)
atan2	Compute arc tangent with two parameters (function)

Hyperbolic functions

cosh	Compute hyperbolic cosine (function)
sinh	Compute hyperbolic sine (function)
tanh	Compute hyperbolic tangent (function)
acosh C++11	Compute arc hyperbolic cosine (function)
asinh C++11	Compute arc hyperbolic sine (function)
atanh C++11	Compute arc hyperbolic tangent (function)

Exponential and logarithmic functions

exp	Compute exponential function (function)
frexp	Get significand and exponent (function)
ldexp	Generate value from significand and exponent (function)
log	Compute natural logarithm (function)
log10	Compute common logarithm (function)
modf	Break into fractional and integral parts (function)
exp2 C++11	Compute binary exponential function (function)
expm1 C++11	Compute exponential minus one (function)
ilogb C++11	Integer binary logarithm (function)
log1p C++11	Compute logarithm plus one (function)
log2 C++11	Compute binary logarithm (function)
logb C++11	Compute floating-point base logarithm (function)
scalbn C++11	Scale significand using floating-point base exponent (function)
scalbln C++11	Scale significand using floating-point base exponent (long) (function)

Power functions

pow	Raise to power (function)
sqrt	Compute square root (function)
cbrt C++11	Compute cubic root (function)
hypot C++11	Compute hypotenuse (function)

 Indique un document ressource

 Retour au sommaire

 Retour à la page courante