

# Introduction aux PSoC



PERSONAL

ACCESS

ENTERPRISE

METRO

CORE

# Diminution du coût des cartes

## Solution Traditionnelle

<b>8-bit Micro</b>	<b>\$2.00</b>
<b>Crystal + Caps</b>	<b>\$0.57</b>
<b>Filters</b>	<b>\$0.30</b>
<b>Amps</b>	<b>\$0.20</b>
<b>Speaker Driver</b>	<b>\$0.15</b>
<b>LED Drivers</b>	<b>\$0.05</b>
<b>Circuit Board</b>	<b>\$1.20</b>
<b>Assembly</b>	<b>\$1.60</b>

Coût système = **\$6.07**

## Solution avec Cypress PSoC

<b>PSoC Micro.</b>	<b>\$2.50</b>
<b>Circuit Board</b>	<b>\$0.90</b>
<b>Assembly</b>	<b>\$1.40</b>



Coût système = **\$4.80**

## PSoC = Programmable System-on-Chip

- Les PSoC sont des circuits mixtes configurables avec un micro contrôleur embarqué.
- Vous permet de configurer votre propre circuit.

Vous définissez :

**Quelles** Fonctions utiliser

**Quand** Les utiliser

**Comment** Les Interconnecter

# Exemple de “Quelles Fonctions Utiliser”

**PSoC permet au concepteur de répondre à son cahier des charges avec un nombre de configurations illimitées**

Ces deux composants peuvent être dans le **même circuit !**

## Composant 1

- Un Compteur 8-Bits
- Un Timer 16-Bits
- Un UART Full-Duplex
- Une liaison série SPI esclave (Full Duplex)
- Un convertisseur A/N Sigma-Delta 8 bits a 4 entrées
- Un convertisseur N/A 6-Bits
- Un convertisseur N/A 8-Bits
- Deux Filtres passe bas

## Composant 2

- Un Compteur 16-Bits
- Une PWM 8-Bits
- Un UART Semi-Duplex
- Une liaison série SPI maître
- Un convertisseur A/N Incrémental 12 bits
- Un Filtre passe bas
- Un convertisseur N/A 8-Bits
- Deux Amplis d'Instrumentation



# “Quand les fonctionnalités apparaissent”

***La reconfiguration dynamique*** permet de mettre en oeuvre des fonctions différentes sur le **MÊME CIRCUIT** à **DIFFÉRENTS MOMENTS** sur le **MÊME SYSTEME**

Exemple:

**23 Heures 59 minutes par jour**

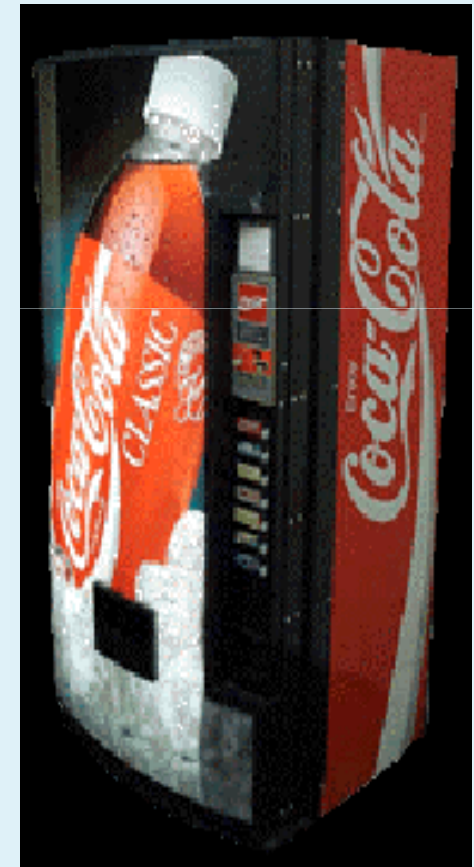
- Encaissement
- Distribution de boissons

**Quelques secondes par nuit**

- Reconfiguration via un Modem
- Transmission de l'état de la machine (argent, boissons, status de maintenance ...) à l'entreprise de gestion

**Bénéfices**

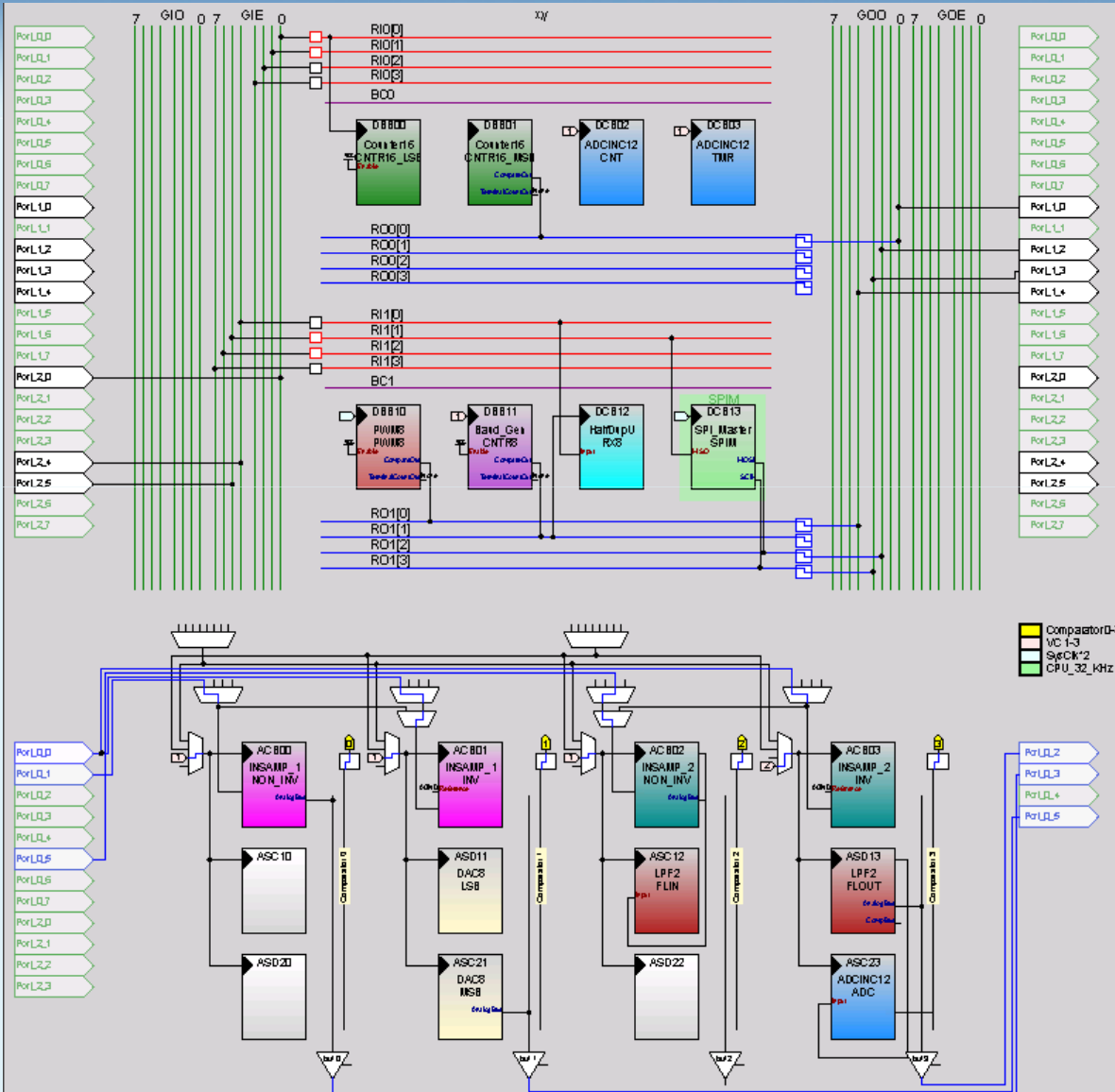
- Augmentation des profits avec cette machine





Connecting From Last Mile to First Mile™

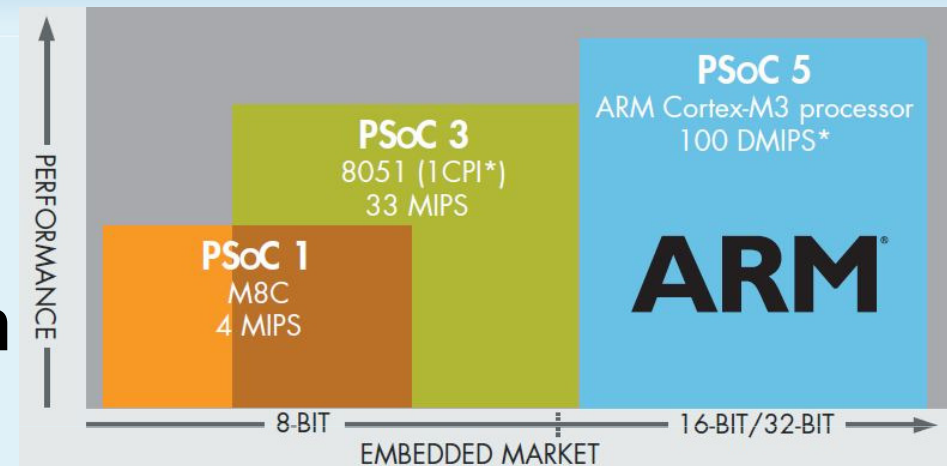
# Comment interconnecter les différentes fonctions



- Définir les connexions entre les broches et les blocs de fonctions
- Définir les connexions entre les différents blocs
- Définir les chemins d'horloge
- On peut aussi changer les connexions de manière dynamique !

# Les différentes familles PSoC

**Trois familles, trois coeurs “micro” différents, des puissances de calcul en conséquence.**



FEATURE	PSoC 1	PSoC 3	PSoC 5
INTERFACE	SPI, UART, GPIO FS-USB, I <sup>2</sup> C	PSoC 1, plus: CAN, I <sup>2</sup> S	Same as PSoC 3
INPUTS	Sensors, CapSense, touchscreen, analog	PSoC 1, plus: precision analog	PSoC 3, plus: high-speed analog
OUTPUTS	LCD segment drive LED control, motor control, analog buffers	PSoC 1, plus: LED drive, advanced motor control	PSoC 3, plus: QVGA LCD control
PROCESSING	M8 24 MHz	8051 67 MHz	ARM® Cortex™-M3 processor 80 MHz

**Des interfaces différentes et des applications ciblées plus “gourmandes”.**



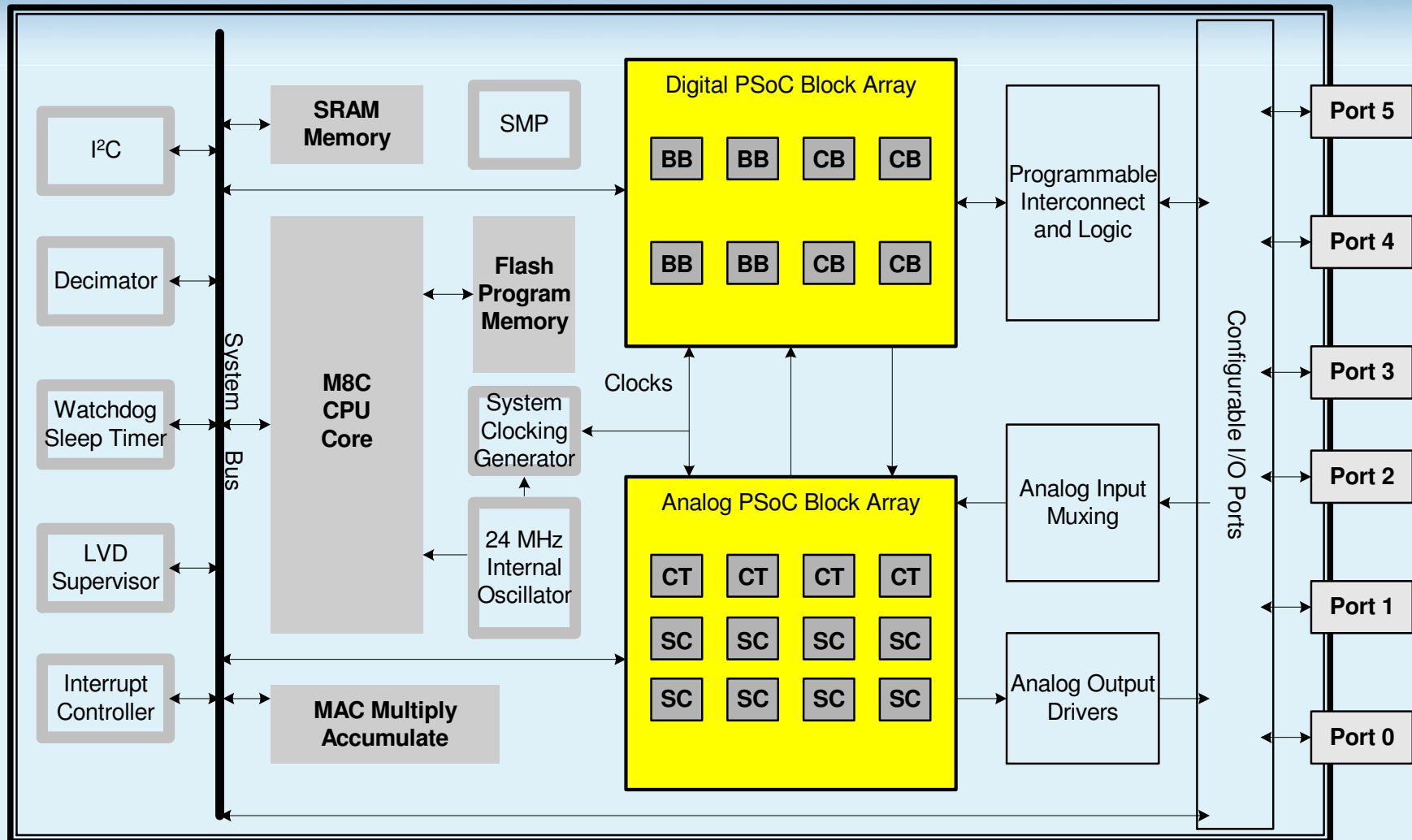
# Les différentes familles PSoC

## Comparatif des trois familles.

	FEATURE	PSoC 1	PSoC 3	PSoC 5
CONFIGURABLE ANALOG/DIGITAL	ADC	1 Delta-Sigma (6- to 14-bit)	1 Delta-Sigma (12- to 20-bit)	1 Delta-Sigma (12- to 20-bit); 2-SAR ADCs (12-bit)
	Sample Rate	Up to 31 ksp/s (8-bit)	192 ksp/s (12-bit)	192 ksp/s (12-bit) Delta-Sigma; 1 Msps (12-bit) SAR ADC
	Reference Voltage Accuracy	±1.53%	Industrial ±0.1%	Industrial ±0.1%
	DACs	Up to 2 (6- to 8-bit)	Up to 4 (12-bit)*	Up to 4 (12-bit)*
	PGA	x1 to x48	x1 to x50	x1 to x50
	LCD Segment Drive	Control/Drive	Control + Drive (736 segments)	Control + Drive (736 segments)
	Integrate Programmable Logic	No	Yes	Yes
	CapSense & Touchscreen	Up to 44 Buttons and 8 Sliders	Up to 62 Buttons and 12 Sliders	Up to 62 Buttons and 12 Sliders
CPU SUBSYSTEM	CPU	M8C	Advanced 8051 (1CPI)	ARM Cortex-M3
	CPU Performance	24 MHz, 4 MIPS	67 MHz, 33 MIPS	80 MHz, 100 DMIPS
	Flash	4 KB to 32 KB	8 KB to 64 KB	32 KB to 256 KB
	SRAM	256B to 2 KB	2 KB to 8 KB	16 KB to 64 KB
	Operating Range	1.7V to 5.25V	0.5V to 5.5V	0.5V to 5.5V
	Power Consumption (Active@6 MHz)	Active: 2 mA, Sleep: 3 µA	Active: 1.2 mA, Sleep: 1 µA, Hibernates: 200 nA	Active: 2 mA, Sleep: 2 µA, Hibernates: 300 nA
	Connectivity Resources	FS USB 2.0, I <sup>2</sup> C, SPI, UART	FS USB 2.0, I <sup>2</sup> C, SPI, UART, CAN, LIN, I <sup>2</sup> S	FS USB 2.0, I <sup>2</sup> C, SPI, UART, CAN, LIN, I <sup>2</sup> S
PROGRAMMABLE INTERCONNECT	Routing & Matrix	Manual Routing, Configurable	Automatic; Any pin anywhere	Automatic; Any pin, anywhere
	Number I/O	Up to 64	Up to 72	Up to 72
TOOLS	Software Development Tools	PSoC Designer and 3rd party compilers	PSoC Creator and 3rd party Compilers/IDEs	PSoC Creator and 3rd party Compilers/IDEs
	In-Circuit Emulation and Debug	Requires ICE Cube and FlexPods (Bond Out)	On-chip JTAG, Debug and Trace; SWD, SWV	On-chip JTAG, Debug and Trace; SWD, SWV



# Les blocs des PSoC

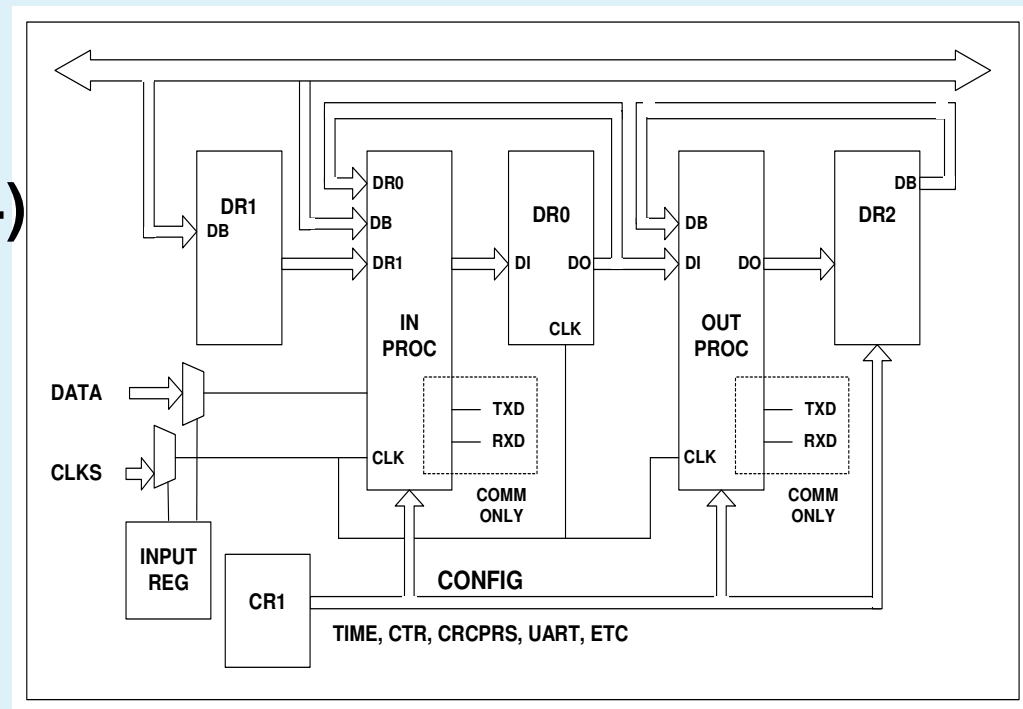


# Huit blocs numériques 8-bits sont disponibles

## De deux types :

- Fonctions de base (4)
- Communications (4)

- Programmable au niveau fonctionnel, pas au niveau “porte”

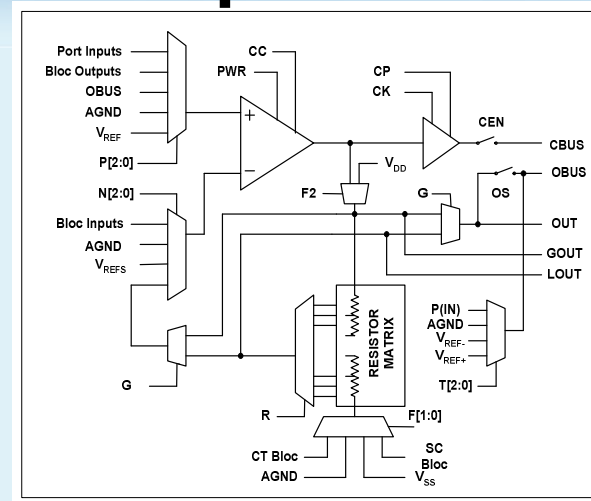


# Douze blocs analogiques sont disponibles

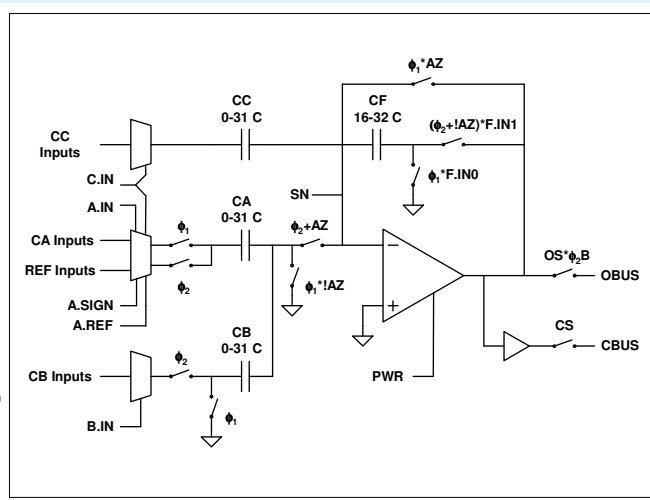
## Trois Types:

- **Linéaires (4)**
- **Capacité commutée C (4)**
- **Capacité commutée D (4)**

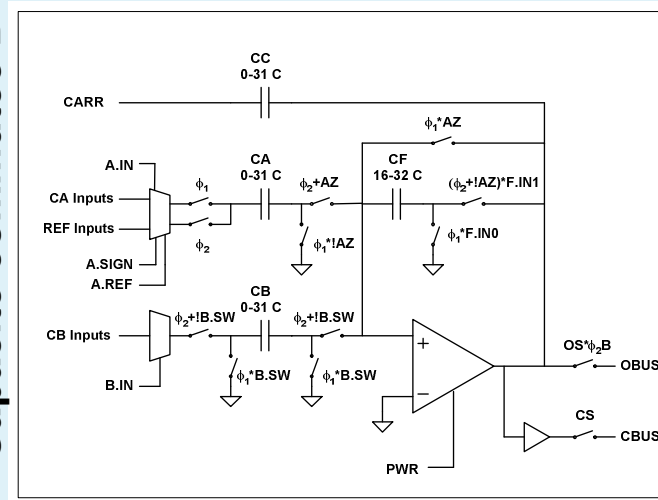
## Temps Continu



## Capacité commutée C



## Capacité commutée D



# Modules Utilisateur

**Blocs analogiques et numériques pré configurés et pré caractérisés.**

**Périphériques standard de micro contrôleurs :**

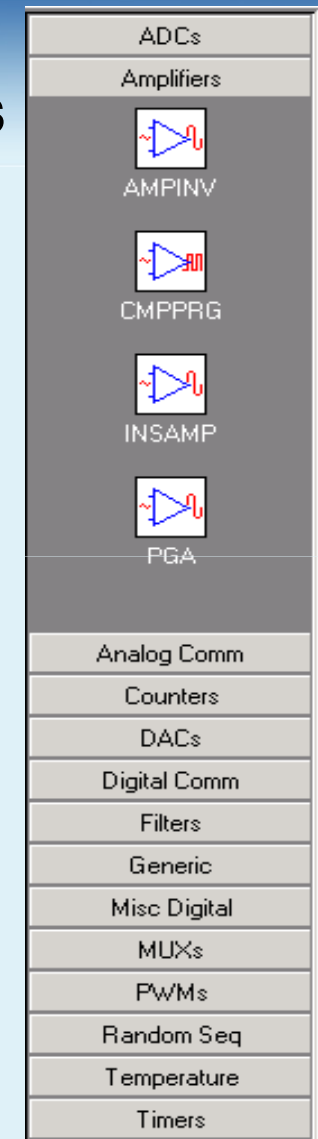
- **Timer- Counters – PWM's**
- **UART – SPI**
- **A/D –DAC's – SAR**

**Les registres de configuration sont gérés automatiquement.**

**Sélectionnables et intégrable dans le projet par double clic.**

**Les modules utilisateurs incluent**

- **Les ressources logicielles correspondantes (APIs)**
- **Les routines d'interruptions (ISRs)**
- **Les documentations (UM Data Sheets)**





# Modules utilisateur numériques

**Timer 8, 16, 24, 32-bits**

**Compteur 8, 16, 24, 32-bits**

**PWM 8, 16-bits**

**Générateur de temps morts 8, 16-bits**

**Générateur Pseudo Aléatoire Pseudo Random  
Source (PRS)**

**Generateur de CRC (Cyclic Redundancy Check)**

**I<sup>2</sup>C Maître**

**I<sup>2</sup>C Esclave**

**SPI Maître**

**SPI Esclave**

**UART Full Duplex**

**Emetteur et récepteur Infra rouge (IrDA)**



Connecting From Last Mile to First Mile™

# Modules utilisateur Analogiques

## Convertisseurs Analogiques/Numériques

- 8-bits à Approximations Successives
- 8-bits Sigma Delta
- 11-bits Sigma Delta
- 12-bits Incremental
- 7-13 bits Incremental Variable
- Deux entrées 7-13 bits Incremental Variable
- Trois entrées 7-13 bits Incremental Variable

## Convertisseurs Numériques/Analogiques

- 6, 8, and 9-bits
- Multiplication 6 et 8 bits

## Filtres

- Filtre Passe Bas 2<sup>ème</sup> ordre
- Filtre Passe Bande 2<sup>ème</sup> ordre

## Amplificateurs

- Amplificateur à Gain Programmable
- Amplificateur d'Instrumentation
- Amplificateur Inverseur

## Comparateur à seuil Programmable

## Décodeur DTMF

**I<sup>2</sup>C Maître**

**EEPROM**

**LCD – Interface pour contrôleur Hitachi  
HD44780**

**Pilotage de LED**

**SD/MMC – API pour lecteur de carte SD/MMC**

# Environnement de Développement Intégré



- **Device Editor**  
(Editeur du composant)
- **Application Editor**  
(Editeur d'application)
- **Compilateur C**
- **Assembleur**
- **Librairies**
- **Debugger**



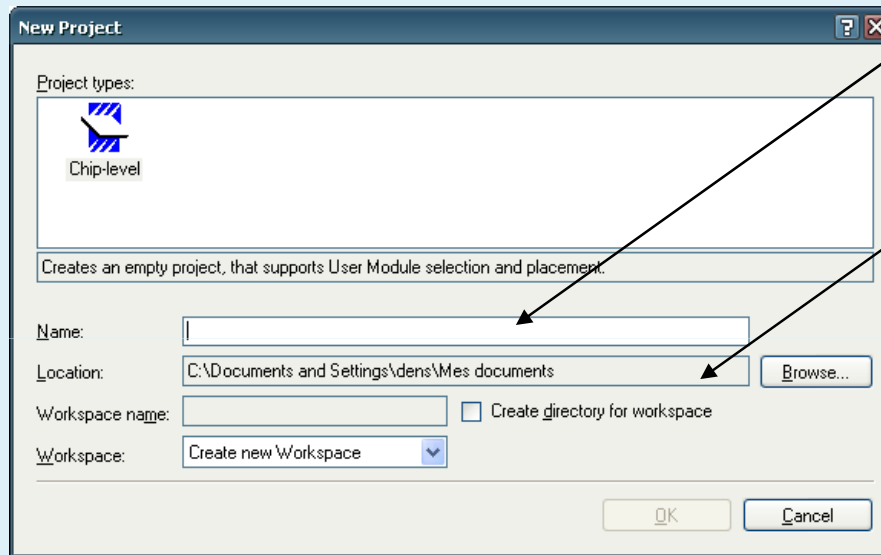
**Pour créer son applications, les étapes suivantes sont à réaliser :**

- Créer un projet
- Sélectionner le ou les user module (UM)
- Configurer le ou les user module (UM)
- Générer l'application
- Coder
- Compiler
- Programmer

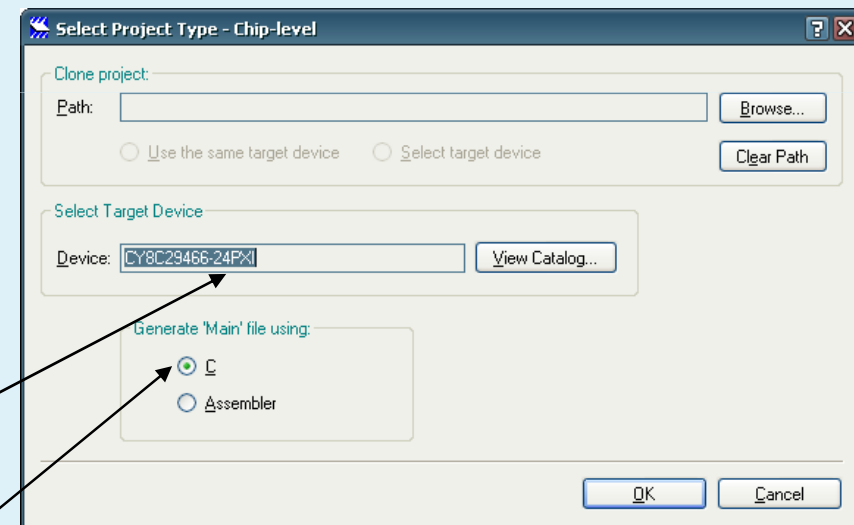
# PSoC Designer

## Création d'un projet

Choisir le nom du projet,  
l'environnement crée un répertoire  
dans le répertoire de base.



Répertoire de base.



Choisir le bon  
composant

Choisir Projet  
en C



Connecting From Last Mile to First Mile™

# PSoC Designer

## Device Editor – Choix et placement des modules utilisateur

hifi\_interface\_v2 - PSoC Designer 5.1

File Edit View Project Interconnect Build Debug Program Tools Window Help

Global Resources - hifi\_interface\_v2

Resource	Value
CPU_Clock	24_MHz (SysClk/1)
32K_Select	Internal
PLL_Mode	Disable
Sleep_Timer	512_Hz
VC1= SysClk/N	10
VC2= VC1/N	1
VC3 Source	SysClk/1
VC3 Divider	1
SysClk Source	Internal 24_MHz
SysClk*2 Disable	No
Analog Power	SC On/Ref High
Ref Mux	(Vdd/2)+/(Vdd/2)
AGndBypass	Disable
Op-Amp Bias	Low
Δ Buff_Power	Low

CPU\_Clock  
Selects the CPU clock speed, from 93.75 KHz to 24 MHz. Derived from the SysClk. This setting affects the Power on Reset level in order to prevent the CP...

Parameters - Controleur\_tension\_signal

Name	Value
User Module	SAR6
Version	1.5
SignalSource	ACB03
InterruptAPI	Enable
IntDispatchMode	ActiveStatus

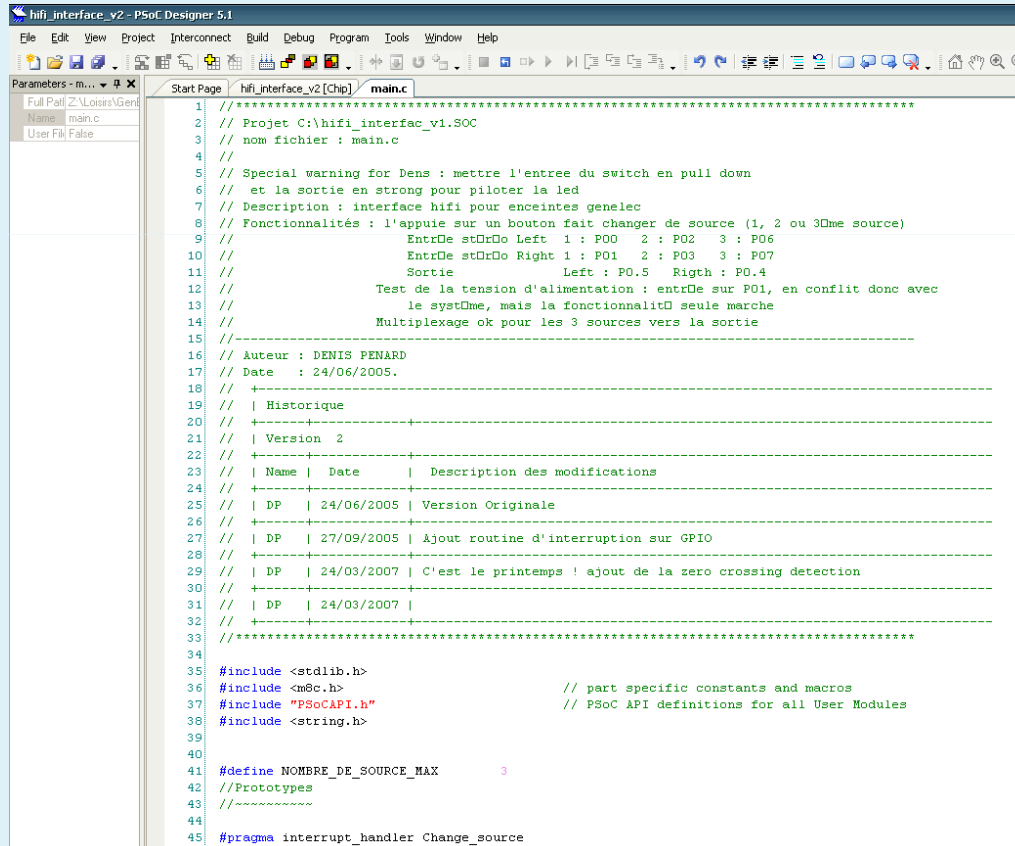
Name  
Indicates the name used to identify this User Module instance

Pinout - hifi\_interface\_v2

Pin	Configuration
P0[0]	AnalogColumn_InputMUX_1, AnalogInput, High Z Analog, Disat
P0[1]	Port_0_1, AnalogInput, High Z Analog, DisableInt, 0
P0[2]	Port_0_2, StdCPU, High Z Analog, DisableInt, 0
P0[3]	AnalogColumn_InputMUX_0, AnalogInput, High Z Analog, Disat
P0[4]	AnalogOutBuf_2, AnalogOutBuf_2, High Z Analog, DisableInt, 0
P0[5]	AnalogOutBuf_1, AnalogOutBuf_1, High Z Analog, DisableInt, 0

- Sélectionner les blocs
- Configurer les blocs
- Placer les blocs
- Définir et configurer les horloges
- Définir et configurer les broches

## Générer l'application.

```

1 //*****
2 // Projet C:\hifi_interfac_v1.SOC
3 // nom fichier : main.c
4 //
5 // Special warning for Dens : mettre l'entree du switch en pull down
6 // et la sortie en strong pour piloter la led
7 // Description : interface hifi pour enceintes genelec
8 // Fonctionnalités : l'appuie sur un bouton fait changer de source (1, 2 ou 3me source)
9 //
10 // EntrDe stDrDo Left 1 : P00 2 : P02 3 : P06
11 // EntrDe stDrDo Right 1 : P01 2 : P03 3 : P07
12 // Sortie Left : P0.5 Righ : P0.4
13 // Test de la tension d'alimentation : entrDe sur P01, en conflit donc avec
14 // le systDme, mais la fonctionnalitD seule marche
15 // Multiplexage ok pour les 3 sources vers la sortie
16 //-----
17 // Auteur : DENIS PENARD
18 // Date : 24/06/2005.
19 //
20 // | Historique
21 // |-----+
22 // | Version 2
23 // |-----+
24 // | Name | Date | Description des modifications
25 // |-----+
26 // | DP | 24/06/2005 | Version Originale
27 // |-----+
28 // | DP | 27/09/2005 | Ajout routine d'interruption sur GPIO
29 // |-----+
30 // | DP | 24/03/2007 | C'est le printemps ! ajout de la zero crossing detection
31 // |-----+
32 // | DP | 24/03/2007 |
33 // |-----+
34 //*****
35 #include <stdlib.h>
36 #include <tm8c.h> // part specific constants and macros
37 #include "PSoCAPI.h" // PSoC API definitions for all User Modules
38 #include <string.h>
39
40
41 #define NOMBRE_DE_SOURCE_MAX 3
42 //Prototypes
43 //-----
44
45 #pragma interrupt_handler Change_source

```

## Coder

- En C
- En assembleur
- Les 2



Compiler l'application

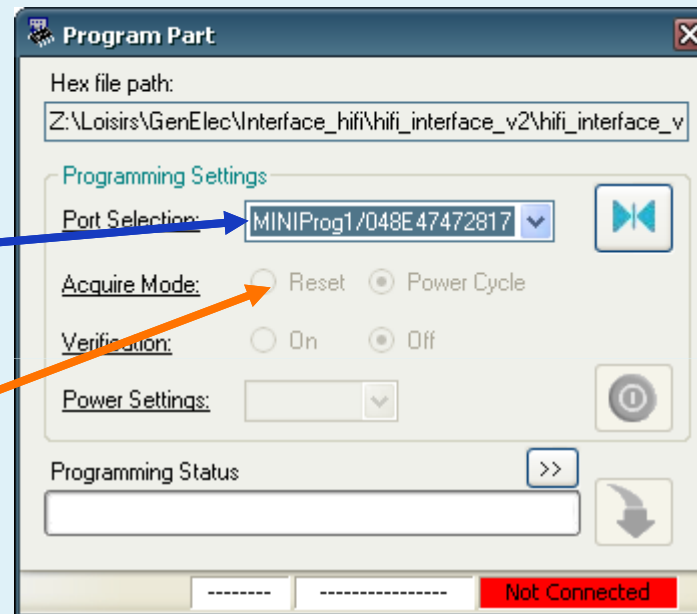


Programmer



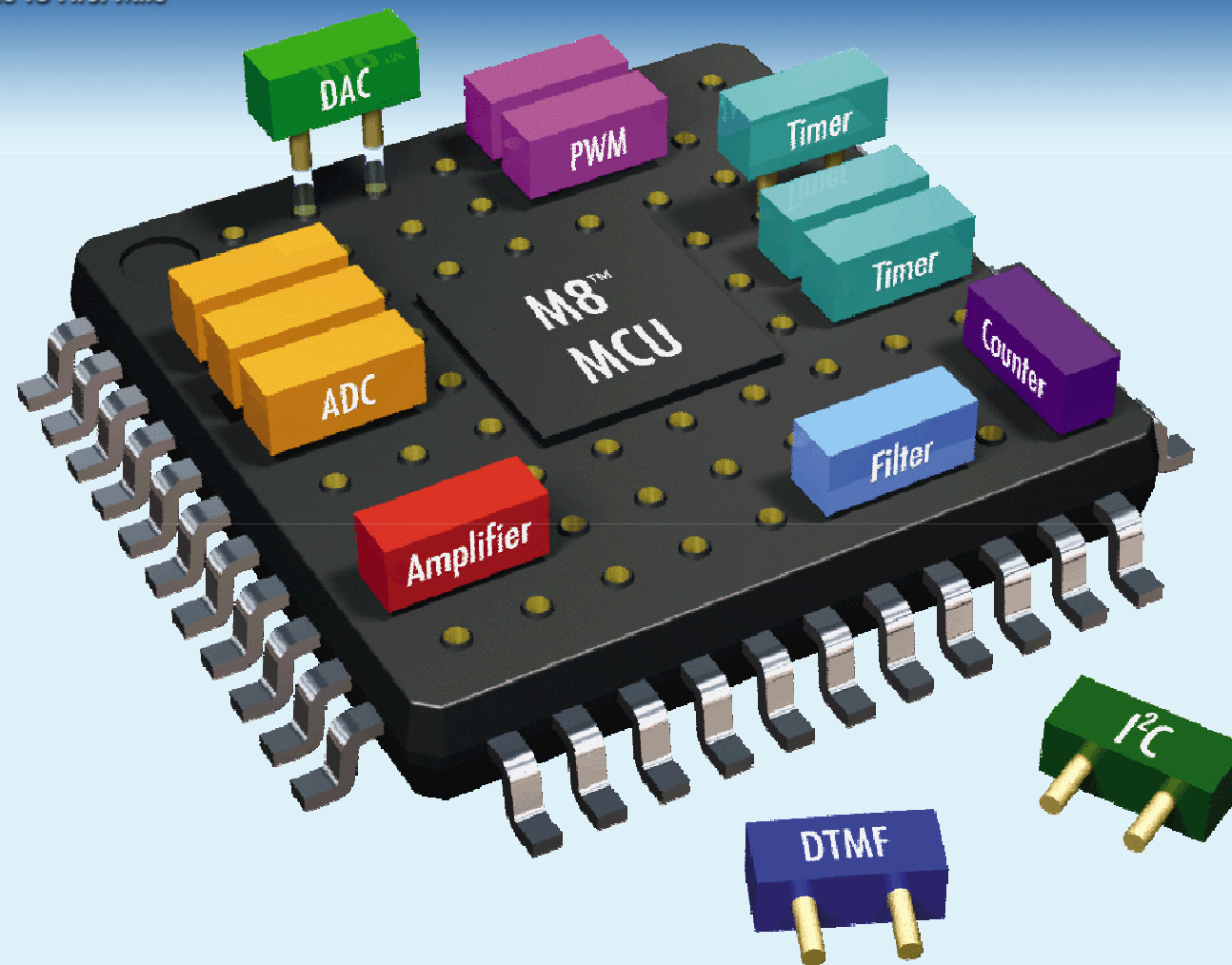
Sélectionner  
Mini prog

Sélectionner  
Mode RESET





Connecting From Last Mile to First Mile™



# Documents ressources :

- Paramétrage ressources globales**
- Mise en œuvre sortie logique**
- Mise en œuvre entrée logique**
- Mise en œuvre LCD**
- Mise en place interruption timer**
- Mise en œuvre liaison série (UART)**
- Mise en œuvre interruption externe**
- Mise en œuvre DAC**
- Mise en œuvre ADC**
- Mise en œuvre multiplexeur avec ADC**
- Mise en œuvre I2C**

# Paramétrage ressources globales

Global Resources	Value
CPU_Clock	3_MHz (SysClk/8)
32K_Select	Internal
PLL_Mode	Disable
Sleep_Timer	512_Hz
VC1= SysClk/N	16
VC2= VC1/N	15
VC3 Source	VC2
VC3 Divider	100
SysClk Source	Internal 24_MHz
SysClk*2 Disable	No
Analog Power	SC On/Ref Low
Ref Mux	(Vdd/2)+/-BandGap
AGndBypass	Disable
Op-Amp Bias	Low
A_Buff_Power	High
SwitchModePump	OFF
Trip Voltage [LVD (SMP)]	4.81V (5.00V)
LVDThrottleBack	Disable
Supply Voltage	5.0V
Watchdog Enable	Disable

Définition des  
horloges  
Internes

Configuration de l'alimentation  
des blocs analogiques :

Analog Power SC On/Ref High

Dynamique de tension des blocs  
Analogiques : Sélectionner

(Vdd/2)+/-(Vdd/2)

Mettre à High pour alimenter les  
blocs analogiques



# Mise en œuvre d'une sortie logique

## Paramétrage des pattes

Sélectionner le Drive Mode

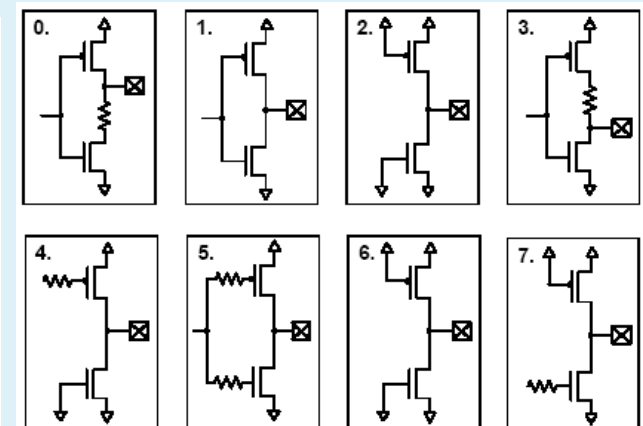
Name	Port	Select	Drive	Interrupt
Port_0_0	P0[0]	StdCPU	Strong	DisableInt
Port_0_1	P0[1]	StdCPU	High Z Analog	DisableInt
Port_0_2	P0[2]	StdCPU	High Z Analog	DisableInt
Port_0_3	P0[3]	StdCPU	High Z Analog	DisableInt
Port_0_4	P0[4]	StdCPU	High Z Analog	DisableInt
Port_0_5	P0[5]	StdCPU	High Z Analog	DisableInt
Port_0_6	P0[6]	StdCPU	High Z Analog	DisableInt
Port_0_7	P0[7]	StdCPU	High Z Analog	DisableInt
Port_1_0	P1[0]	StdCPU	High Z Analog	DisableInt

Output logique : STRONG

Input logique : High Z, ou  
pull up ou pull down

Analogique Input ou output :  
High Z analog

Drive Modes				Diagram Number		
DM2	DM1	DM0	Drive Mode		Data = 0	Data = 1
0	0	0	Resistive Pull Down	0	Resistive	Strong
0	0	1	Strong Drive	1	Strong	Strong
0	1	0	High Impedance	2	Hi-Z	Hi-Z
0	1	1	Resistive Pull Up	3	Strong	Resistive
1	0	0	Open Drain, Drives High	4	Hi-Z	Strong (Slow)
1	0	1	Slow Strong Drive	5	Strong (Slow)	Strong (Slow)
1	1	0	High Impedance Analog	6	Hi-Z	Hi-Z
1	1	1	Open Drain, Drives Low	7	Strong (Slow)	Hi-Z



# Mise en œuvre d'une sortie logique

## Codage

PRT?DR = 0x??

Port considéré

Valeur

Pour travailler sur un bit, l'utilisation de masque est obligatoire :

```
PRT0DR = PRT0DR | 0x01; //set pin 0 of 0 port
```

```
PRT0DR = PRT0DR & 0xFE; //reset pin 0
```

Dans le masque, ne mettre à 1 que les broches en sorties

# Mise en œuvre d'une entrée logique

## Paramétrage des pattes

Name	Port	Select	Drive	Interrupt
Port_0_0	P0[0]	StdCPU	Strong	DisableInt
Port_0_1	P0[1]	StdCPU	High Z Analog	DisableInt
Port_0_2	P0[2]	StdCPU	High Z Analog	DisableInt
Port_0_3	P0[3]	StdCPU	High Z Analog	DisableInt
Port_0_4	P0[4]	StdCPU	High Z Analog	DisableInt
Port_0_5	P0[5]	StdCPU	High Z Analog	DisableInt
Port_0_6	P0[6]	StdCPU	High Z Analog	DisableInt
Port_0_7	P0[7]	StdCPU	High Z Analog	DisableInt
Port_1_0	P1[0]	StdCPU	High Z Analog	DisableInt

Sélectionner le Drive Mode :  
High Z, pull up ou pull down

## Code

**Variable = (PRT0DR & 0x??)**



Masque pour sélectionner le bit à tester

# Mise en œuvre d'un écran LCD

## Sélection de l'UM LCD



## Configuration du user module

User Module Parameters	Value
LCDPort	Port_2
BarGraph	Disable

# Mise en œuvre d'un écran LCD

## Utilisation du LCD (API en langage C)

Exemple :

```
char theStr[] = "PSoC LCD"; // Define RAM string
LCD_Start(); // Initialize LCD
LCD_Position(0,5); // Place LCD cursor at row 0, col 5.
LCD_PrString("PsoC LCD"); // Print "PSoC LCD" on the LCD
LCD_PrCString (theStr); // Print a constant "ROM" string
```

## API d'affichage:

```
extern void LCD_Start(void);
extern void LCD_Init(void);
extern void LCD_Control(BYTE bData);
extern void LCD_WriteData(BYTE bData);
extern void LCD_PrString(char * sRamString);
extern void LCD_PrCString(const char * sRomString);
extern void LCD_Position(BYTE bRow, BYTE bCol);
extern void LCD_PrHexByte(BYTE bValue);
extern void LCD_PrHexInt(INT iValue);
```



# Mise en place d'une interruption timer

## Différentes étapes

- Insertion et placement d'un UM Timer
- Sélection du mode d'interruption
- Mise en place de la réponse à l'interruption timer
- Mise en place du code à exécuter

## Insertion et placement d'un UM Timer



## Sélection du mode d'interruption : Définition des paramètres de l'UM

Timer16	
User Module Parameters	Value
Clock	VC3
Capture	Low
TerminalCountOut	None
CompareOut	None
Period	1000
CompareValue	0
CompareType	Less Than Or Equal
InterruptType	Terminal Count
ClockSync	Sync to SysClk
TC_PulseWidth	Full Clock
InvertCapture	Normal

Utilisation de l'horloge VC3

$\text{Periode Interrupt} = \text{Period} * \text{Période}_{\text{VC3}}$

Déclenchement d'une interruption sur fin de comptage

# Mise en place d'une interruption timer

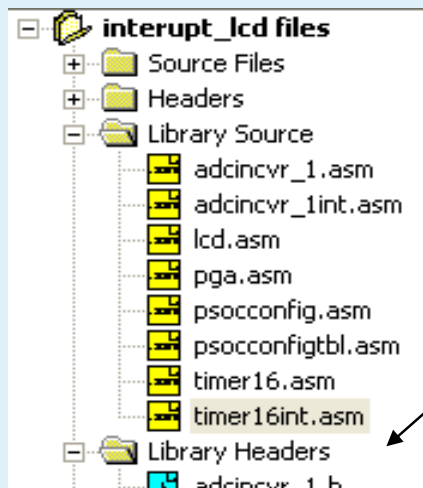
## F Calcul de la période d'interruption

Global Resources	Value
32K_Select	Internal
PLL_Mode	Disable
Sleep_Timer	512_Hz
VC1= SysClk/N	16
VC2= VC1/N	15
VC3 Source	VC2
VC3 Divider	100
SysClk Source	Internal 24_MHz
SysClk*2 Disable	No

$$VC3 = VC2 / 100 = \text{SysClk} / 16 / 15 / 100 = 1000\text{Hz}$$

$$\text{Periode Interrupt} = \text{TVC3} * \text{Period} = \frac{1}{1000} * 1000 = 1\text{s}$$

## Mise en place de la réponse à l'interruption timer



Après avoir généré l'application, ouvrir le source assembleur correspondant à la réponse à une interruption du timer :

# Mise en place d'une interruption timer

```

_Timer16_ISR:

;@PSoC_UserCode_BODY@ (Do not change this line.)
;-----
; Insert your custom code below this banner
;-----
; NOTE: interrupt service routines must preserve
; the values of the A and X CPU registers.

    ljmp _affiche

;-----
; Insert your custom code above this banner
;-----
;@PSoC_UserCode_END@ (Do not change this line.)

    reti
    
```

- Rechercher dans le source assembleur (timer16int.asm) le code correspondant à la réponse à une interruption
- Insérer un LongJump vers la procédure en C contenant le code à réaliser : `ljmp _nom_procedure`

## Mise en place du code à exécuter (dans main.c)

```

#pragma interrupt_handler affiche
void affiche(void)
{
    // placer le code à exécuter
}

void main()
{
    Timer16_EnableInt();
    Timer16_Start();

    M8C_EnableGInt;
}
    
```

Indique au compilateur que `affiche` est une procédure de réponse à une interruption

Autorise les interruptions timer

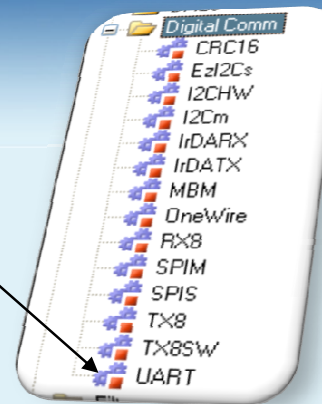
Démarre le timer

Autorise les interruptions au niveau du cœur du micro

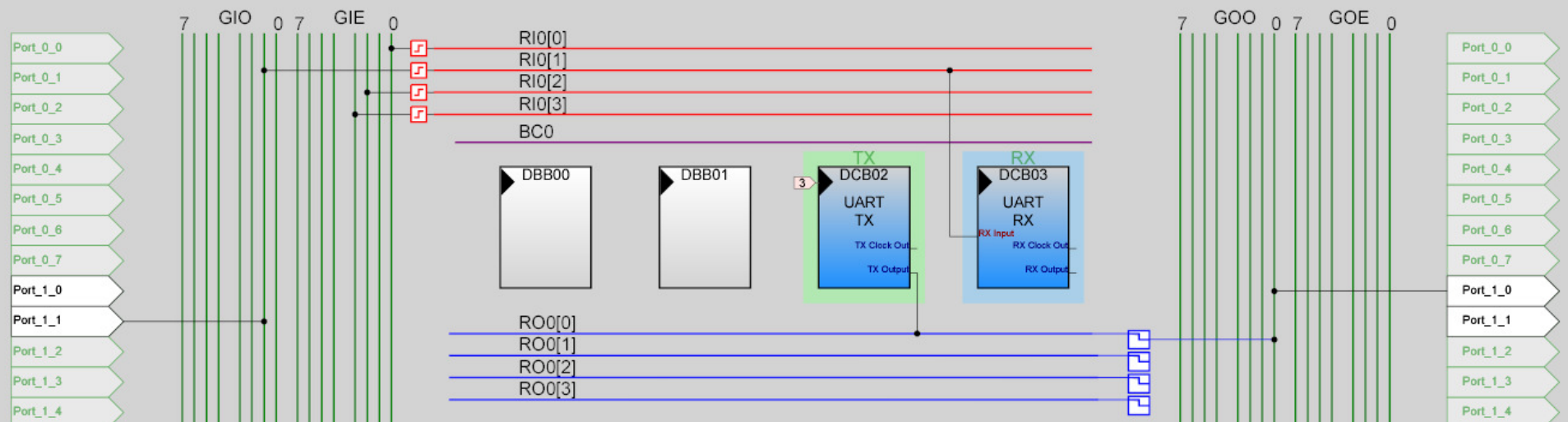
# Mise en œuvre de la liaison série

Sélectionner le bloc UART

Placer et router UART



Relier RX à une broche (P11 dans l'exemple, configurée en (High Z) par défaut)  
Relier TX à une broche (P10 dans l'exemple, configurée en (Strong) par défaut)



# Mise en œuvre de la liaison série

## Configuration du user module

Parameters - UART_1	
Name	UART_1
User Module	UART
Version	5.3
Clock	VC3
RX Input	Row_0_Input_1
TX Output	Row_0_Output_0
TX Interrupt Mode	TXComplete
ClockSync	Sync to SysClk
RxCmdBuffer	Enable
RxBufferSize	16
CommandTerminator	13
Param_Delimiter	32
IgnoreCharsBelow	32
Enable_BackSpace	Disable
RX Output	None
RX Clock Out	None
TX Clock Out	None
InterruptAPI	Enable
IntDispatchMode	ActiveStatus
InvertRX Input	Normal

Relier à une horloge dont la fréquence est 8x le débit de la liaison

VC 3 = 76 800 HZ pour 9 600 Bauds

Travail en API haut niveau

Le caractère ASCII 13 est interprété comme une fin de commande

Le caractère ASCII 32 est interprété comme délimiteur d'argument dans la commande considérée

Relier physiquement sur le kit les broches P11 à RX et P10 à TX

## Configurer le terminal

Liaison : 9600,8,1, none; contrôle de flux : aucun

Propriétés / Paramètres / Configuration ASCII : Valider 'reproduire localement les caractères entrés.



# Mise en œuvre de la liaison série

## Codage

```
#include <m8c.h>
#include "PSoCAPI.h"

void main()
{
    char * strPtr;                // défini un pointer qui poitera sur les commandes UART

    UART_CmdReset();              // Initialisation du registre de commande / buffer

    UART_IntCntl(UART_ENABLE_RX_INT); // Enable RX interrupts

    UART_Start(UART_PARITY_NONE);  // Enable UART

    MSC_EnableGInt ;              // Turn on interrupts

    UART_CPutString("\r\n UART CACHAN \r\n");

    while(1)
    {
        if(UART_bCmdCheck())       // Si commande recue
        {
            if(strPtr = UART_szGetParam()) // si commande présente
            {
                UART_CPutString("Found valid command\r\nCommand =>");
                UART_PutString(strPtr); // Print out command
                UART_CPutString("<\r\nParameters:\r\n");
                while(strPtr = UART_szGetParam()) // si argument present
                { // loop on each parameter
                    UART_CPutString("    <");
                    UART_PutString(strPtr); // Print each parameter
                    UART_CPutString(">\r\n");
                }
                UART_CmdReset(); // Reset command buffer
            }
        }
    }
}
```

# Mise en place d'une interruption externe

## Différentes étapes

- Configurer la broche (Drive et mode d'interruption)
- Mise en place de la réponse à l'interruption GPIO
- Mise en place du code à exécuter

## Configurer la patte (Drive et mode d'interruption)

Name	Port	Select	Drive	Interrupt
Port_0_0	P0[0]	StdCPU	Pull Down	RisingEdge

Sur le kit,  
l'interrupteur est  
connecté au Vcc

Interruption sur front  
montant

# Mise en place d'une interruption externe

## Mise en place de la réponse à l'interruption GPIO

```
PSoC_GPIO_ISR:

;@PSoC_UserCode_BODY@ (Do not change this line.
;-----
; Insert your custom code below this banner
;-----
    ljmp _lafonctionappeleesuiteainterruption
;-----
; Insert your custom code above this banner
;-----
;@PSoC_UserCode_END@ (Do not change this line.)

reti
```

- Rechercher dans le source assembleur (PSoCGPIoint.asm) le code correspondant à la réponse à une interruption
- Insérer un LongJump vers la procédure en C contenant le code à réaliser : `ljmp _nom_procedure`

# Mise en place d'une interruption externe

## Mise en place du code à exécuter

```
#pragma interrupt_handler lafonctionappeleesuiteainterruption
void lafonctionappeleesuiteainterruption()
{
    BYTE PTO;
    PTO = PRTODR;
    if ((PTO && 0x01) == 0x01)
    {
        //traitement
    }
}

void main(void)
{
    MSC_EnableIntMask (INT_MSK0, INT_MSK0_GPIO); //autorise les interruptions du port GPIO
    MSC_EnableGInt; //autorise les interruptions globales
}
```

# Mise en œuvre d'un convertisseur numérique-analogique

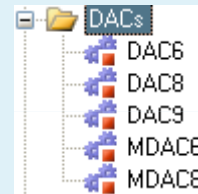
## Différentes étapes

- Insertion et placement d'un UM DAC
- Routage de la sortie et de l'horloge
- Sélection des paramètres UM
- Codage

## Insertion et placement d'un UM DAC

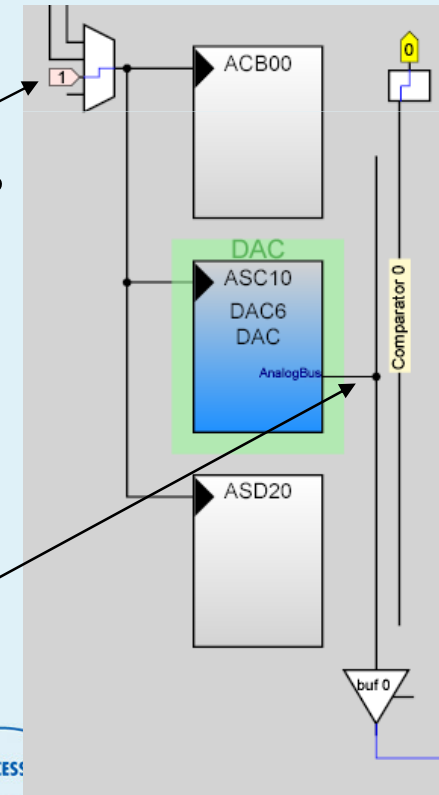
## Routage de la sortie et de l'horloge

$f_{\text{clock}}$ , Analog Column Clock <sup>1</sup>			
Low Power	0.128 to 0.5	--	MHz
Med Power	0.128 to 2.0	--	MHz
High Power	0.128 to 3.2	--	MHz



Configurer VC1 dans les  
Global Ressources

Connexion de la sortie à l'Analog Out Bus





# Mise en œuvre d'un convertisseur numérique-analogique

## Sélection des paramètres UM

### Global Ressources :

Analog Power	SC On/Ref High
Ref Mux	(Vdd/2)+/(Vdd/2)
AGndBypass	Disable
Op-Amp Bias	Low
A_Buff_Power	High

User Module Parameters	Value
AnalogBus	AnalogOutBus_0
ClockPhase	Normal
DataFormat	OffsetBinary

Choisir le format !

## Codage

```
#include <m8c.h>           // part specific constants and macros
#include "PSoCAPI.h"       // PSoC API definitions for all User Modules

void main()
{
    // Insert your main routine code here.
    DAC6_Start(DAC6_HIGHPOWER);
    DAC6_WriteBlind(5);
}
```

# Mise en œuvre d'un convertisseur analogique-numérique

## Différentes étapes

- Insertion et placement d'un UM ADC
- Routage de l'entrée
- Sélection des paramètres UM
- Codage

## Insertion et placement d'un UM ADC

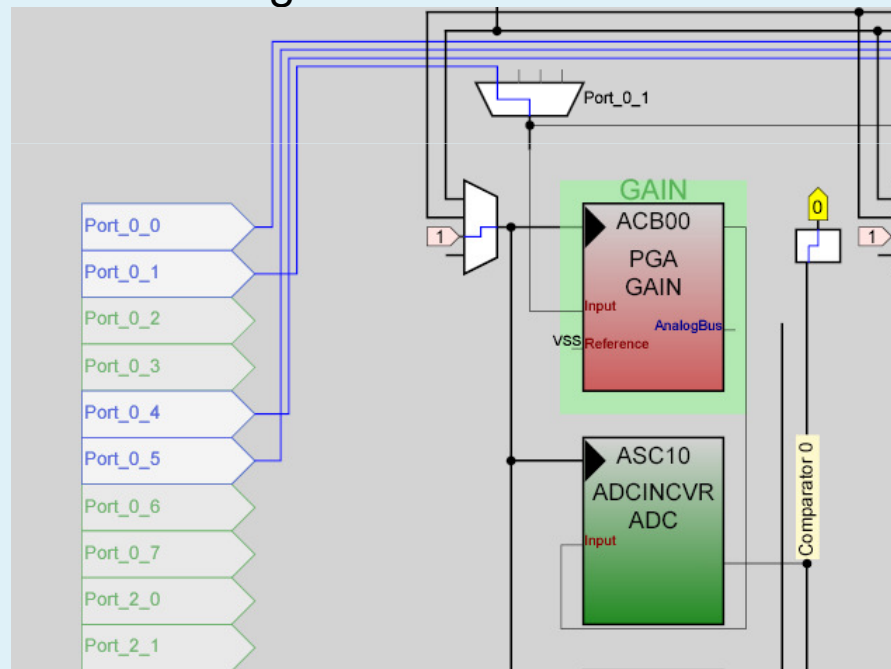
Sélectionner par exemple un ADC incrémental 7-13 bits :  
Utiliser le placement par défaut.



# Mise en œuvre d'un convertisseur analogique-numérique

## Routage de l'entrée

Le plus souvent, il est impossible de connecter directement l'entrée du convertisseur sur une broche. Il est alors nécessaire de mettre en œuvre un PGA de gain 1 :



PGA	
User Module Parameters	Value
Gain	1.000
Input	AnalogColumn_InputMUX_0
Reference	VSS
AnalogBus	Disable

Dans cet exemple, l'entrée du convertisseur est reliée à la sortie du PGA. L'entrée du PGA est reliée à P10 via le `AnalogColumn_InputMUX_0`

# Mise en œuvre d'un convertisseur analogique-numérique

## Sélection des paramètres UM

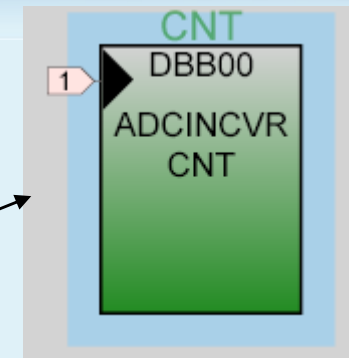
ADCINCVR	
User Module Parameters	Value
Input	ACB00
ClockPhase	Norm
Clock	?
ADCResolution	8 Bit
CalcTime	1
DataFormat	Unsigned

### ADCINCVR DC and AC 5.0V Electrical Characteristics (Data Sheet)

Data Clock		0.125 to 2.67	MHz	Input to digital blocks and analog column clock
------------	--	---------------	-----	---

VC1= SysClk/N 10

VC1 = 2.4 M Hz



Dans notre exemple, sélection de VC1 comme horloge, avec VC1=2.4 MHz

### Global Ressources :

Analog Power	SC On/Ref High
Ref Mux	(Vdd/2)+/(Vdd/2)
AGndBypass	Disable
Op-Amp Bias	Low
A_Buff_Power	High

# Mise en œuvre d'un convertisseur analogique-numérique

## Codage

```
#include <m8c.h>           // part specific constants and macros
#include "PSoC_API.h"      // PSoC API definitions for all User Modules
#include <string.h>
#include <stdlib.h>

void main()
{
    // Insert your main routine code here.
    char theStr[] = "IUT Cachan"; // Define RAM string
    BYTE iData;

    LCD_Start();              // Initialize LCD
    LCD_Position(0,5);        // Place LCD cursor at row 0, col 5.
    LCD_PrString(theStr);     // Print "PSoC LCD" on the LCD

    PGA_Start(PGA_HIGHPOWER);

    M8C_EnableGInt;          // Enable global interrupts

    ADCINCVR_Start(ADCINCVR_HIGHPOWER); // Turn on Analog section
    ADCINCVR_GetSamples(0);    // Start ADC to read continuously
    for (;;)
    {
        while(ADCINCVR_fIsDataAvailable() == 0); // Wait for data to be ready.
        iData = ADCINCVR_iGetData();             // Get Data
        ADCINCVR_ClearFlag();                    // Clear data ready flag

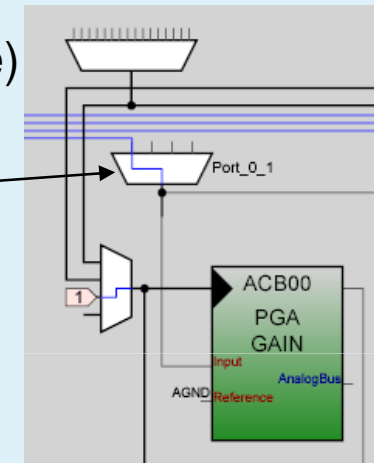
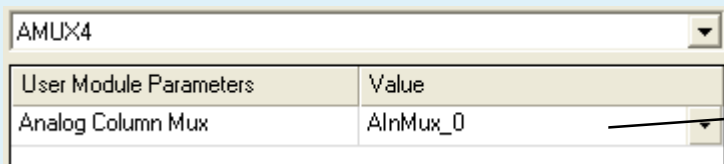
        LCD_Position(1,10);                      // Place LCD cursor at row 0, col 10.
        itoa(theStr,iData,10);                   // copie dans theStr le contenu de iData en base 10
        strcat(theStr," ");                     // ajoute des blancs dans theStr
        LCD_PrString(theStr);                   // affiche
    }
}
```

# Utilisation d'un multiplexeur

Sélectionner un bloc AMUX4



Paramétrer le multiplexeur considéré (assignation d'une colonne)



Pour connaître l'état d'un multiplexeur, utiliser le registre AMX\_IN

## 20.2.1 AMX\_IN Register

Add.	Name	Cols.	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Access
0,60h	AMX_IN	4	ACI3[1:0]		ACI2[1:0]		ACI1[1:0]		ACI0[1:0]		RW : 00
		2					ACI1[1:0]		ACI0[1:0]		

Pour affecter l'état d'un multiplexeur, utiliser la procédure

```
AMUX4_InputSelect(AMUX4_PORT0_3);
```

Paramètre défini dans AMUX4.h



# Utilisation du bus I2C

Insertion et placement d'un UM I2CHW (Utilisation du bloc Hardware I2C)

Sélection des paramètres UM

Codage

## Insertion et placement d'un UM I2CHW



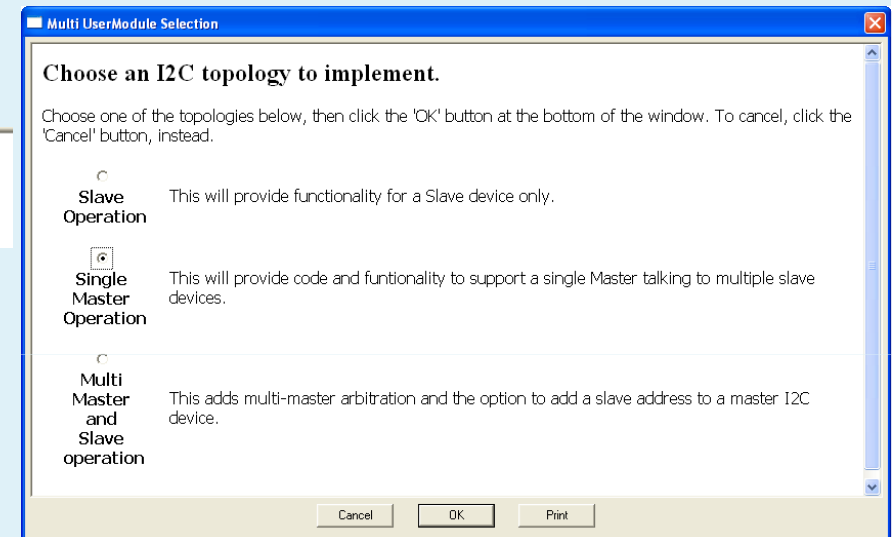
Choisir le type de topologie (I2CMaster),  
puis placer l'UM

## Sélection des paramètres UM

User Module Parameters	Value
Read_Buffer_Types	RAM ONLY
CPU_Clk_speed_(CY8C27xA)	NOT CY8C27xA
I2C Clock	100K Standard
I2C Pin	P[1]5-P[1]7

SDA

SCL

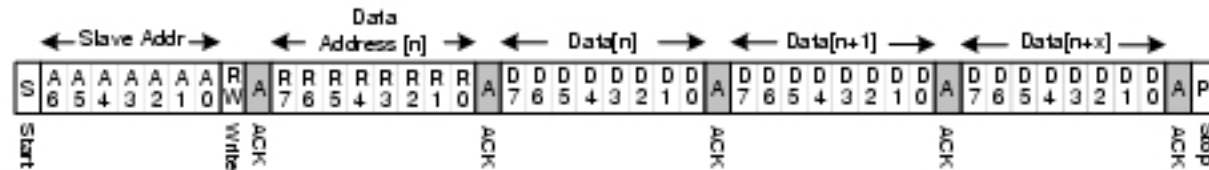


A spécifier pour cause de bug sur la  
série 27xA

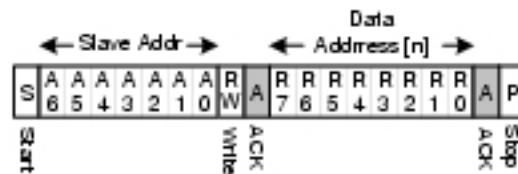
# Utilisation du bus I2C

## Cycle lecture / écriture

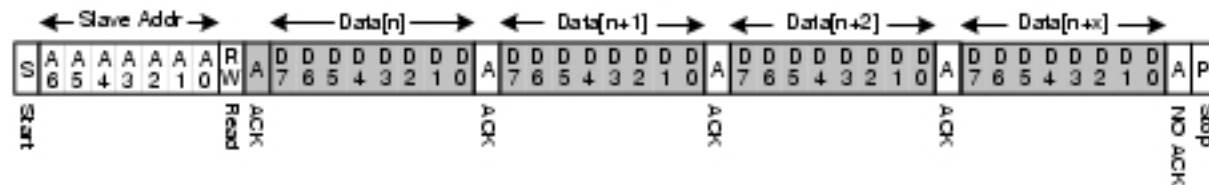
### Write x Bytes to I2C Slave



### Set Slave Data Pointer



### Read x Bytes from I2C Slave



Master  
Slave

# Utilisation du bus I2C

## Codage

### Initialisation

```
/* Start the master */
I2CHW_Start();
I2CHW_EnableMstr();

/* Enable the global and local interrupts */
MSC_EnableGInt;
I2CHW_EnableInt();
```

### Écriture d'une trame

```
I2CHW_WriteBytes(0x00, txBufferInitAdress, 2, I2CHW_CompleteXfer);
```

Adresse de  
L'esclave  
(0x00 pour broadcast)

Longueur de la  
trame

Trame : voir cycle écriture [le pointeur de registre est incrémenté, puis non réinitialisé en fin d'écriture]

### Vérification de la bonne écriture d'une trame

```
while (!(I2CHW_ReadI2CStatus() & I2CHW_WR_COMPLETE));
```

```
I2CHW_ClrWrStatus();
```

Constant	Value	Description
I2CHW_RD_NOERR	01h	Data read by the master, normal ISR exit
I2CHW_RD_OVERFLOW	02h	More data bytes were read by the master than were available
I2CHW_RD_INCOMPLETE	04h	A read was initiated but has not yet been completed
I2CHW_READFLASH	08h	The next read will be from a Flash location
I2CHW_WR_NOERR	10h	Data was written successfully by the master
I2CHW_WR_OVERFLOW	20h	The master wrote too many bytes for the write buffer
I2CHW_WR_COMPLETE	40h	A master write was completed by a new address or stop
I2CHW_ISR_ACTIVE	80h	The I <sup>2</sup> C ISR has not yet exited and is active

# Utilisation du bus I2C

## Codage

### Lecture d'une trame

Écriture du pointeur d'adresse (écriture d'une trame ne comportant d'un octet)

```
do
{
    txBuffer[0] = 0x03;
    I2CHW_bWriteBytes(SLAVE_ADDRESS, txBuffer, 1, I2CHW_CompleteXfer);
}
while (!(I2CHW_bReadI2CStatus() & I2CHW_WR_COMPLETE));
I2CHW_ClrWrStatus();
```

### Lecture de la trame de donnée

Adresse de  
L'esclave

```
#define SLAVE_ADDRESS 0x70 // adresse divisé par 2 !!!! 0x70 pour 0xE0
```

```
I2CHW_fReadBytes(SLAVE_ADDRESS, rxBuffer, 1, I2CHW_CompleteXfer);
while (!(I2CHW_bReadI2CStatus() & I2CHW_RD_COMPLETE));
I2CHW_ClrRdStatus();
```

Attente de la fin d'écriture

Trame : voir cycle de lecture [le pointeur de registre est incrémenté, puis réinitialisé en fin de lecture]