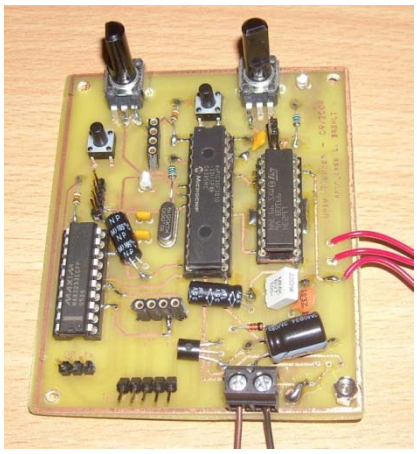
Université Aboubekr BELKAID
TLEMCEN
Faculté de Technologie

جامعة ابو بكر بلقايد
تلمسان
كلية التكنلوجيا

http://www.univ-tlemcen.dz/



Master M1

Commande de machines

Module

µ-processeurs et µ-contrôleurs

Enseignant

Lotfi BAGHLI

EC713 CM **Salle C004**

EC741 TP **Salle : Lab. de microprocesseurs**

http://baghli.com/doc_archi_cmde.php

Version 1.9 - 11/10/2016

Introduction

Ce polycopié est destiné aux étudiants de Master M1 en génie électrique pour l'apprentissage du fonctionnement et de l'utilisation d'un microcontrôleur de type DSC (Digital Signal Controller) dspic. Nous avons choisi comme cible, le dspic 30F3010 de Microchip [1].
Des cartes électroniques ont été spécialement réalisées pour le support de ce cours et des TP sont proposés. Les schémas sont en annexes.
Ce document comporte de larges extraits des datasheets de Microchip, principalement celle du dspic 30F3010 [2] et celui de la famille 30F [3].

Problématique de la commande de machines

Pour commander des machines électriques ou des dispositifs d'électronique de puissance (onduleur, hacheur, convertisseur matriciel), il faut un microcontrôleur qui a des sorties MLI ou PWM, des entrées analogiques (ADC) pour la mesure des courants, des entrées/sorties logiques, une grande capacité de calcul, un système de gestion des interruptions…

Ce microcontrôleur va venir commander un amplificateur (onduleur ou hacheur) afin d'alimenter l'actionneur (moteur) convenablement et entraîner la charge (Figure 1). L'amplificateur puise son énergie d'une source qui peut être une batterie, dans le cas d'un système embarqué, ou un redresseur et un condensateur de filtrage, dans le cas d'un système alimenté par une source fixe triphasée.
La charge peut être réversible en puissance et peut donc renvoyer de l'énergie. Le moteur bascule naturellement en fonctionnement en génératrice mais il faut que l'amplificateur et la source primaire d'énergie soient réversibles afin de faire remonter l'énergie ainsi générée.

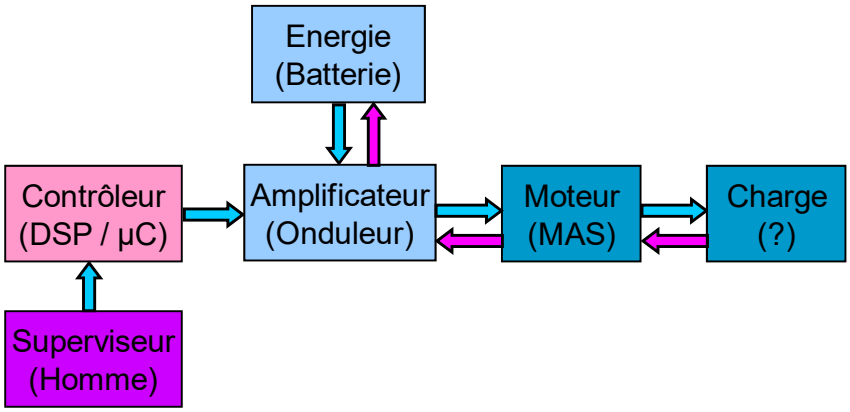
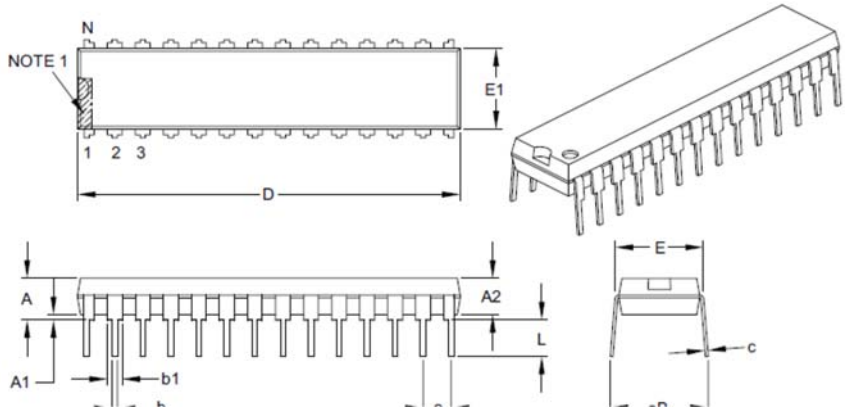


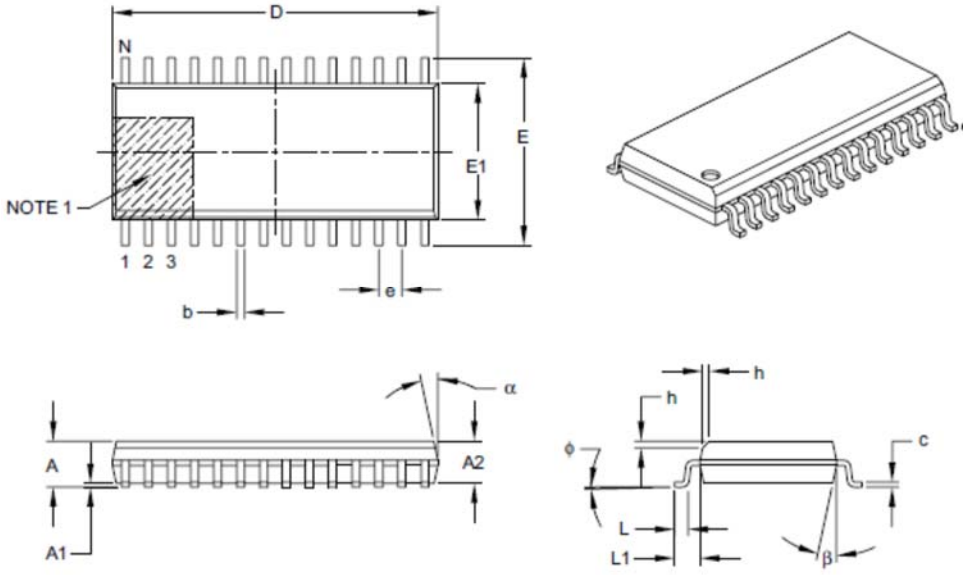
Figure 1 schéma synoptique de la chaîne de conversion d'énergie

Présentation du dspic 30F3010

Nous avons choisi de travailler sur ce dspic pour plusieurs raisons ; C'est un microcontrôleur 16 bits à noyau DSP (appelé donc DSC). Il peut faire des calculs au rythme soutenu de 30 MIPS et dispose de sortie MLI, d'un convertisseur analogique, d'une UART, d'un Watchdog, d'entrées / sorties logiques, de la programmation in situ (sans avoir à le déplacer vers un programmateur)…
Il existe des versions PDIP, cms SOIC et QFN voir page 205 de la datasheet dsPIC30F3010 3010 Datasheet 70141e.pdf



SOIC



Le brochage du dspic est représenté en (Figure 2).

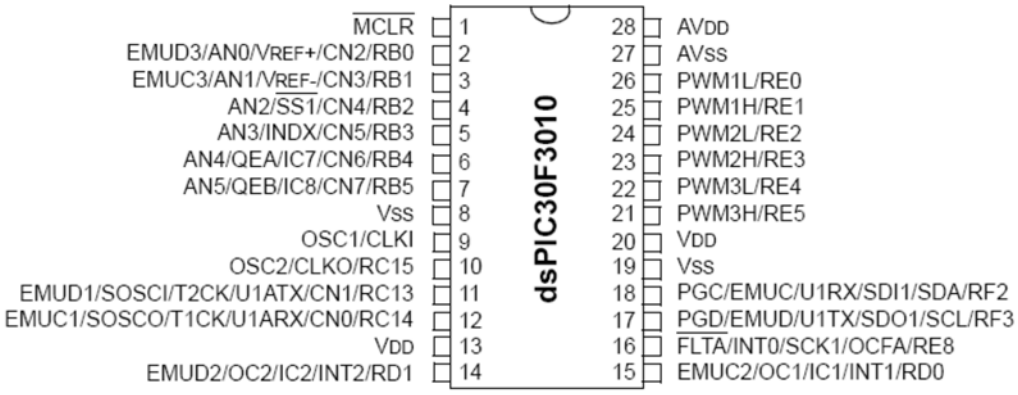


Figure 2 Schéma de brochage du dspic 30F3010

Pin Name	Pin Type	Buffer	Type Description
AN0-AN5 I	I	Analog	Analog input channels. AN0 and AN1 are also used for device programming data and clock inputs,

			respectively.
AVDD	P	P	Positive supply for analog module.
AVSS	P	P	Ground reference for analog module.
CLKI CLKO	I O	ST/CMOS --	External clock source input. Always associated with OSC1 pin function. Oscillator crystal output. Connects to crystal or resonator in Crystal Oscillator mode. Optionally functions as CLKO in RC and EC modes. Always associated with OSC2 pin function.
CN0-CN7	I	ST	Input change notification inputs. Can be software programmed for internal weak pull-ups on all inputs.
EMUD EMUC EMUD1 EMUC1 EMUD2 EMUC2 EMUD3 EMUC3	I/O I/O I/O I/O I/O I/O I/O I/O	ST ST ST ST ST ST ST ST	ICD Primary Communication Channel data input/output pin. ICD Primary Communication Channel clock input/output pin. ICD Secondary Communication Channel data input/output pin. ICD Secondary Communication Channel clock input/output pin. ICD Tertiary Communication Channel data input/output pin. ICD Tertiary Communication Channel clock input/output pin. ICD Quaternary Communication Channel data input/output pin. ICD Quaternary Communication Channel clock input/output pin.
IC1, IC2, IC7, IC8	I	ST	Capture inputs 1, 2, 7 and 8.
INDX QEA QEB	I I I	ST ST ST	Quadrature Encoder Index Pulse input. Quadrature Encoder Phase A input in QEI mode. Auxiliary Timer External Clock/Gate input in Timer mode. Quadrature Encoder Phase A input in QEI mode. Auxiliary Timer External Clock/Gate input in Timer mode.
INT0 INT1 INT2	I I I	ST ST ST	External interrupt 0. External interrupt 1. External interrupt 2.
$\overline{\text{FLTA}}$ PWM1L PWM1H PWM2L PWM2H PWM3L PWM3H	I O O O O O O	ST -- -- -- -- -- --	PWM Fault A input. PWM 1 Low output. PWM 1 High output. PWM 2 Low output. PWM 2 High output. PWM 3 Low output. PWM 3 High output.
$\overline{\text{MCLR}}$	I/P	ST	Master Clear (Reset) input or programming voltage input. This pin is an active low Reset to the device.
OCFA OC1, OC2	I O	ST --	Compare Fault A input (for Compare channels 1, 2, 3 and 4). Compare outputs 1 and 2.
OSC1 OSC2	I I/O	ST/CMOS --	Oscillator crystal input. ST buffer when configured in RC mode; CMOS otherwise. Oscillator crystal output. Connects to crystal or resonator in Crystal Oscillator mode. Optionally functions as CLKO in RC and EC modes.
PGD PGC	I/O I	ST ST	In-Circuit Serial Programming : Data input/output pin. In-Circuit Serial Programming : Clock input pin.
RB0-RB5	I/O	ST	PORTB is a bidirectional I/O port.
RC13-RC15	I/O	ST	PORTC is a bidirectional I/O port.
RD0-RD1	I/O	ST	PORTD is a bidirectional I/O port.
RE0-RE5, RE8	I/O	ST	PORTE is a bidirectional I/O port.
RF2-RF3	I/O	ST	PORTF is a bidirectional I/O port.
SCK1 SDI1 SDO1	I/O I O	ST ST --	Synchronous serial clock input/output for SPI #1. SPI #1 Data In. SPI #1 Data Out.
SCL SDA	I/O I/O	ST ST	Synchronous serial clock input/output for I2C.. Synchronous serial data input/output for I2C.
SOSCO SOSCI	O I	-- ST/CMOS	32 kHz low-power oscillator crystal output. 32 kHz low-power oscillator crystal input. ST buffer when configured in RC mode; CMOS otherwise.
T1CK T2CK	I I	ST ST	Timer1 external clock input. Timer2 external clock input.
U1RX U1TX U1ARX U1ATX	I O I O	ST -- ST --	UART1 Receive. UART1 Transmit. UART1 Alternate Receive. UART1 Alternate Transmit.

V_{DD}	P	--	Positive supply for logic and I/O pins.
V_{SS}	P	--	Ground reference for logic and I/O pins.
V_{REF+}	I	Analog	Analog Voltage Reference (High) input.
V_{REF-}	I	Analog	Analog Voltage Reference (Low) input.

Legend:

CMOS = CMOS compatible input or output
Analog = Analog input
ST = Schmitt Trigger input with CMOS levels
O = Output
I = Input
P = Power

dsPIC30F3010/3011

High Performance, 16-Bit Digital Signal Controllers [2]

High-Performance Modified RISC CPU:

- Modified Harvard Architecture
- C Compiler Optimized Instruction Set Architecture with Flexible Addressing modes
- 83 Base Instructions
- 24-Bit Wide Instructions, 16-Bit Wide Data Path
- 24 Kbytes On-Chip Flash Program Space (8K instruction words)
- 1 Kbyte of On-Chip Data RAM
- 1 Kbyte of Nonvolatile Data EEPROM
- 16 x 16-Bit Working Register Array
- Up to 30 MIPs Operation:
  - DC to 40 MHz external clock input
  - 4 MHz-10 MHz oscillator input with PLL active (4x, 8x, 16x)
- 29 Interrupt Sources
  - 3 external interrupt sources
  - 8 user-selectable priority levels for each interrupt source
  - 4 processor trap sources

DSP Engine Features:

- Dual Data Fetch
- Accumulator Write Back for DSP Operations
- Modulo and Bit-Reversed Addressing modes
- Two, 40-Bit Wide Accumulators with Optional saturation Logic
- 17-Bit x 17-Bit Single-Cycle Hardware Fractional/Integer Multiplier
- All DSP Instructions Single Cycle
- ±16-Bit Single-Cycle Shift

Peripheral Features:

- High-Current Sink/Source I/O Pins: 25 mA/25 mA
- Timer module with Programmable Prescaler:
  - Five 16-bit timers/counters; optionally pair
- 16-bit timers into 32-bit timer modules
- 16-Bit Capture Input Functions
- 16-Bit Compare/PWM Output Functions
- 3-Wire SPI modules (supports 4 Frame modes)
- I2CTM module Supports Multi-Master/Slave mode
- and 7-Bit/10-Bit Addressing
- 2 UART modules with FIFO Buffers

Motor Control PWM Module Features:

- 6 PWM Output Channels
  - Complementary or Independent Output modes
  - Edge and Center-Aligned modes
- 3 Duty Cycle Generators
- Dedicated Time Base
- Programmable Output Polarity
- Dead-Time Control for Complementary mode
- Manual Output Control
- Trigger for A/D Conversions

Quadrature Encoder Interface Module Features:

- Phase A, Phase B and Index Pulse Input
- 16-Bit Up/Down Position Counter
- Count Direction Status
- Position Measurement (x2 and x4) mode
- Programmable Digital Noise Filters on Inputs
- Alternate 16-Bit Timer/Counter mode
- Interrupt on Position Counter Rollover/Underflow

Analog Features:

- 10-Bit Analog-to-Digital Converter (ADC) with 4 S/H Inputs:
  - 1 Msps conversion rate
  - 9 input channels
  - Conversion available during Sleep and Idle

- Programmable Brown-out Reset

Special Microcontroller Features:

- Enhanced Flash Program Memory:10,000 erase/write cycle (min.) for
- industrial temperature range, 100K (typical)
- Data EEPROM Memory: 100,000 erase/write cycle (min.) for industrial temperature range, 1M (typical)
- Self-Reprogrammable under Software Control
- Power-on Reset (POR), Power-up Timer (PWRT) and Oscillator Start-up Timer (OST)
- Flexible Watchdog Timer (WDT) with On-Chip Low-Power RC Oscillator for Reliable Operation
- Fail-Safe Clock Monitor Operation Detects Clock
- Failure and Switches to On-Chip Low-Power RC Oscillator
- Programmable Code Protection
- In-Circuit Serial Programming(ICSP.)
- Selectable Power Management modes: Sleep, Idle and Alternate Clock modes

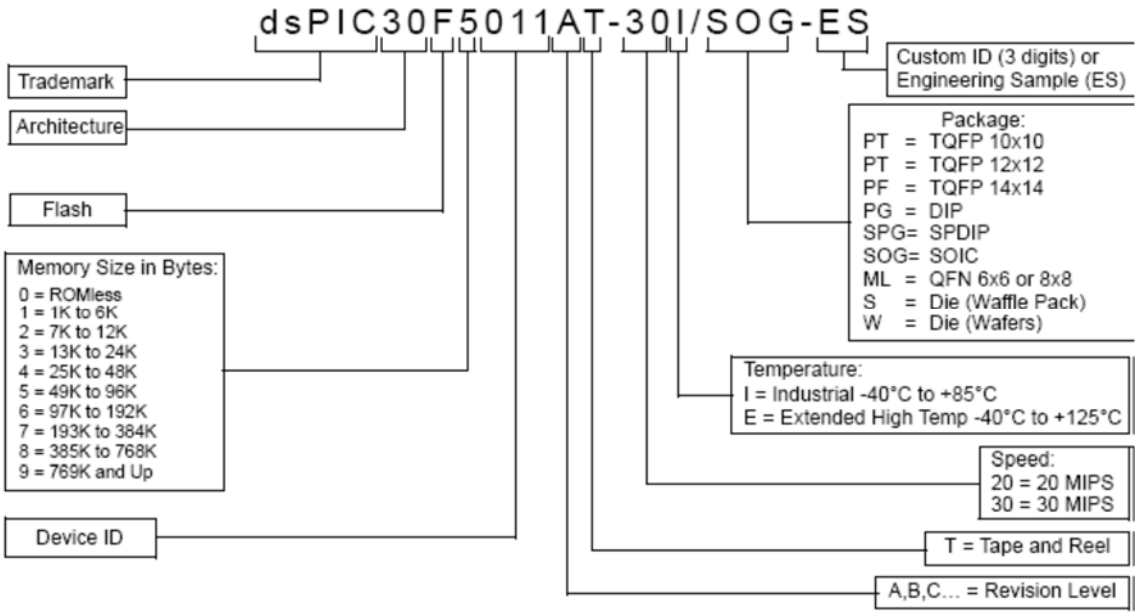
CMOS Technology:

- Low-Power, High-Speed Flash Technology
- Wide Operating Voltage Range (2.5V to 5.5V)
- Industrial and Extended Temperature Ranges
- Low Power Consumption

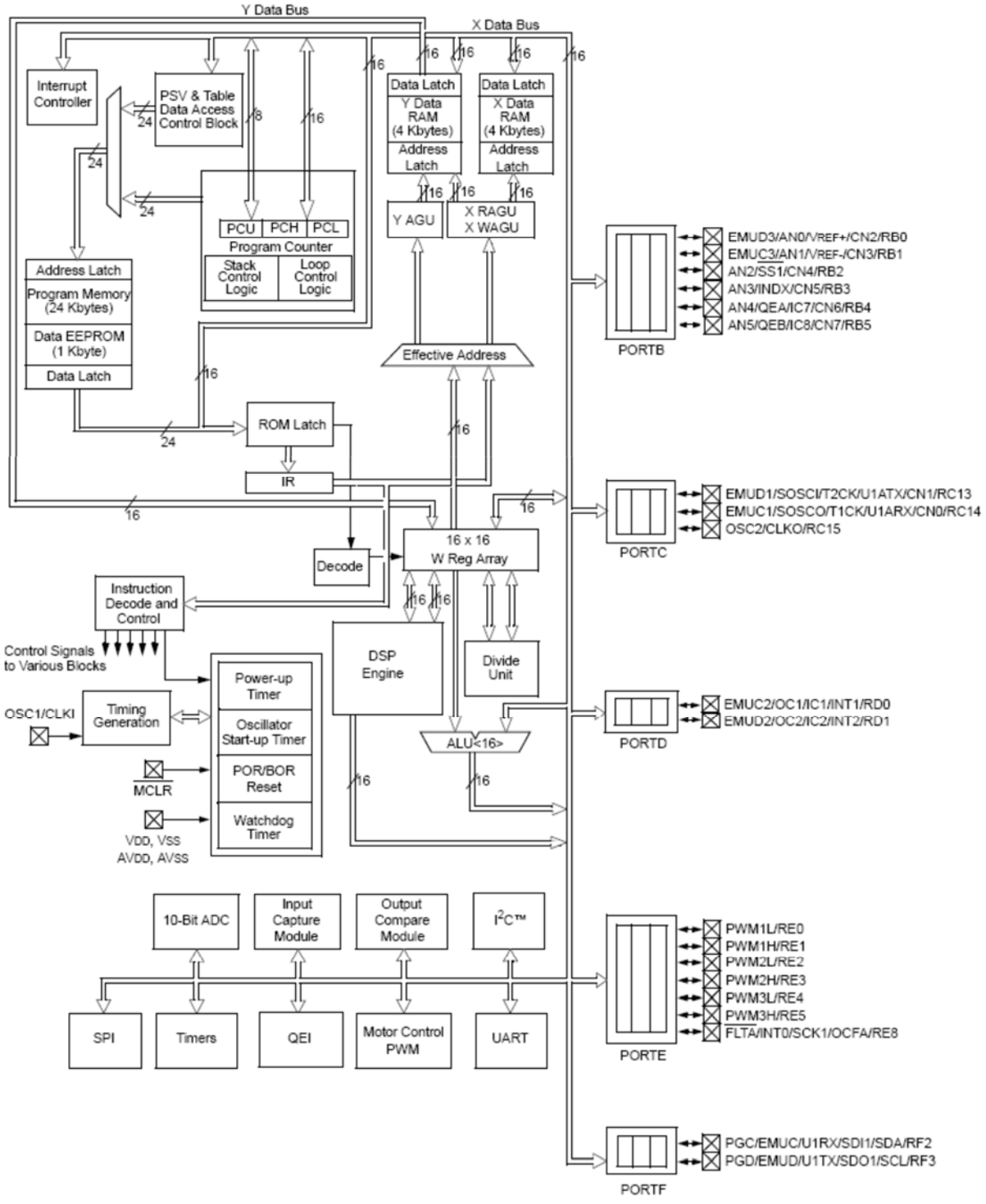
Il existe 3 familles [4] : General Purpose Family, Motor Control and Power Conversion Family, Sensor Family.

Motor Control and Power Conversion Family:

Device	Pins	Program Memory		SRAM Bytes	EEPROM Bytes	Timer 16-bit	Input Capture	Output Compare/ Std. PWM	Motor Control PWM	A/D 10-bit 1 Msps	Quad Enc.	UART	SPI™	I^2C™	CAN	I/O Pins (Max.)(1)	Packages (2)
		Bytes	Instructions														
dsPIC30F2010	28	12K	4K	512	1024	3	4	2	6 ch	6 ch	1	1	1	1	—	20	SOG, PG, ML
dsPIC30F3010	28	24K	8K	1024	1024	5	4	2	6 ch	6 ch	1	1	1	1	—	20	SOG, PG
dsPIC30F4012	28	48K	16K	2048	1024	5	4	2	6 ch	6 ch	1	1	1	1	1	20	SOG, PG
dsPIC30F3011	40/44	24K	8K	1024	1024	5	4	4	6 ch	9 ch	1	2	1	1	—	30	PG, PT
dsPIC30F4011	40/44	48K	16K	2048	1024	5	4	4	6 ch	9 ch	1	2	1	1	1	30	PG, PT
dsPIC30F5015	64	66K	22K	2048	1024	5	4	4	8 ch	16 ch	1	1	2	1	1	52	PT
dsPIC30F6015	64	144K	48K	8192	4096	5	8	8	8 ch	16 ch	1	2	2	1	2	52	PT
dsPIC30F6010(3) dsPIC30F6010A	80	144K	48K	8192	4096	5	8	8	8 ch	16 ch	1	2	2	1	2	68	PF, PT



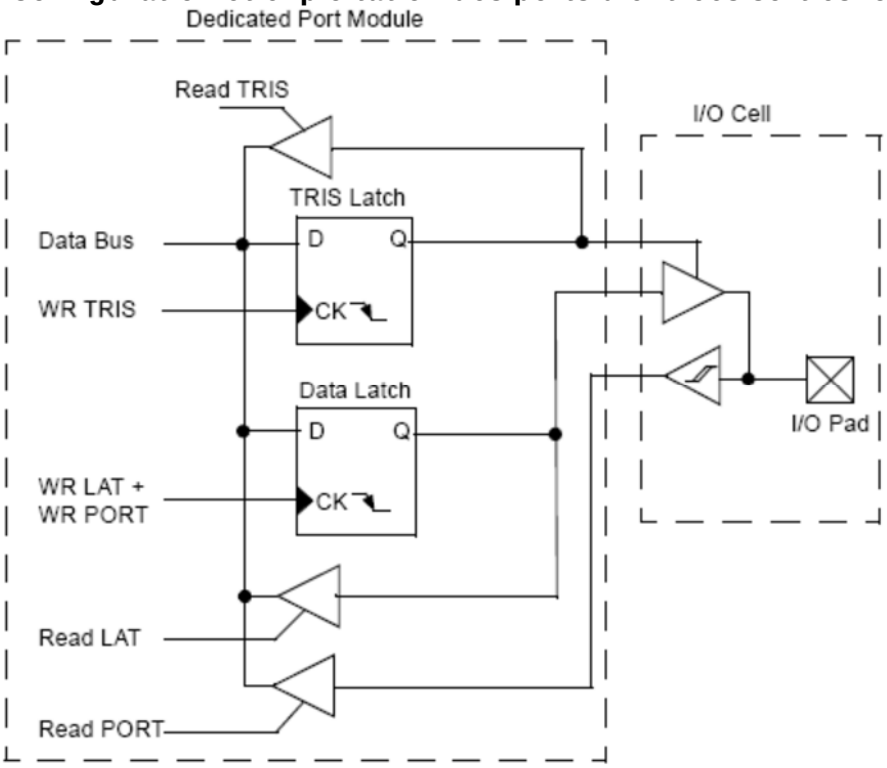
dsPIC30F3010 BLOCK DIAGRAM



Entrées/Sorties

Configuration des registres TRISx. Lecture et écriture des registres PORTx et LATx. Allumer et eteindre des LED, Entrée d'un signal logique à l'aide d'un Bouton Poussoir (BP).
Utilisation des I/O pour observer les tâches en temps réel dans le programme.

I / O : Configuration et exploitation des ports d'entrées sorties logiques



Le dsPIC comporte 20 pins d'entrées sorties partagées. Il faut donc choisir celles qui vont rester sur les fonctions primaires (ADC, SPI, PWM...) et celles qui seront en fonctionnement I/O logique.
Ce dsPIC comporte plusieurs ports (Port **B**, **C**, **D**, **E**, et **F**). Pour chacun, 3 registres sont associés (TRIS**x**, PRT**x**, LAT**x**).

- TRIS**x** : Chaque bit indique la direction du port correspondant : 1 pour input, 0 pour output
- PORT**x** : Accède aux données sur le port : Une lecture donne l'état de la pin d'entrée/sortie, une écriture impose cet état sur le latch.
- LAT**x** : Même fonctionnement que PORTX mais à la lecture, elle donne la valeur qui se trouve sur le port latch et non pas sur la pin elle-même.

Exemple d'utilisation pour allumer une diode LED :

```
#define LEDBlue PORTEbits.RE0      // donne un nom explicite au bit n°0 du port E
TRISE = 0xFFFE;  // définit la direction des ports : RE0 Output LED, RE1 à RE8 input

PORTEbits.RE0=1;
// ligne équivalente à
LEDBlue=1;        // met le bit à 1 donc VDD (5V ou 3V) sur la pin RE0
```

Configurer d'abord le registre **TRISx** pour choisir quel port sera en sortie (**O**utput) **0** et lequel sera en entrée (**I**nput) **1**. Chaque bit du registre correspond à la voie du port en quesition.
Ne pas oublier, si on a des entrées analogiques, de les spécifier à l'aide du registre **ADPCFG** : choix des pins en E/S logique ou en analogique (1 D**I**g**I**tal, 0 Anal**O**g).
Par défaut, au rester du dspic, la configuration est ADPCFG = 0xFFFF, c'est-à-dire que toutes les pins sont affectés aux ports logiques (I/O) et elle sont en entrées (TRISx=0xFFFF).

Exemple de lecture du port d'entrée touche:

```
#define ToucheRUN PORTEbits.RE1    // donne un nom explicite au bit n°1
du port E
#define LEDBlue PORTEbits.RE0      // donne un nom explicite au bit n°0
du port E
TRISE = 0xFFFE;  // définit la direction des ports : RE0 Output LED, RE1
à RE8 input
if (ToucheRUN)  LEDBlue=1;        // Allume la LED si la touche est
appuyée
// ligne équivalente à
if (PORTEbits.RE1)  PORTEbits.RE0=1;        // Allume la LED si la touche
est appuyée
```

Attention cependant si l'on écrit 2 fois de suite sur des ports voisins exemple RD0 et RD1.
Si on le fait de cette manière :

```
PORTDbits.RD0 = 1;
PORTDbits.RD1 = 0;
```

et en fonction de l'état initial des ports, on peut avoir des résultats très bizarres dus au Latch... voir doc.
Donc, il vaut mieux utiliser le Latch pour écrire sur le port (Output) :

```
LATDbits.LATD0 = 1;
LATDbits.LATD1 = 0;
```

Pour la lecture du ports (Input), il vaut mieux utiliser

```
Etatreel = PORTDbits.RD2
```

car on aura l'état réel de la broche du dspic.

Changements :
Depuis quelques temps maintenant, Microchip, par sa nouvelle version de compilateur C et des bibliothèques associées, a rendu plus facile l'écriture et l'accès aux ports logiques.
Une correspondance a été établie, par exemple au lieu d'écrire :

```
PORTEbits.RE2
```

Il suffit d'écrire :

```
_RE2
```

et à la place de :

```
LATEbits.LATE2
```

Il suffit d'écrire :

```
_LATE2
```

Bien sûr, nous utiliserons toujours les `#define`, ce qui donne :

```
#define ToucheRUN _RE1  // donne un nom explicite au bit n°1 du port E
#define LEDBlue _LATE0   // donne un nom explicite au bit n°0 du port E
```

```
TRISE = 0xFFFE;  // définit la direction des ports : RE0 Output LED, RE1 à RE8 input
if (ToucheRUN)  LEDBlue=1;        // Allume la LED si la touche est appuyée
```

En résumé :
Tout ce qui est **Sortie** (LED, commande...) `_LATxx` : (`_LATB2` , `_LATE0` ,..).
Tout ce qui est **Entrée** (bouton poussoir, sonde Hall, capteur tout ou rien,...) `_Rxx` : (`_RB3` , `_RE0` ,...).

Timer

Les timers ou compteurs matériels permettent d'avoir des tâches qui s'exécutent de manière synchrone, toutes les périodes.
Dans le cadre de la commande de machines et de la régulation, il très important d'avoir une exécution de certaines fonctions à une période d'échantillonnage précise.
Cela se fait généralement dans une routine de service de l'interruption (ISR).
Pour configurer un timer, on doit connaître la période à laquelle on doit le programmer et la fréquence interne du dspic FCY qui dépend de la configuration matérielle (Quartz externe, FRC,...) et logicielle (PLL, Prescaler).
On calcule ensuite la valeur à mettre dans le registre PR1.
Si la période est trop grande par rapport aux 16 bits de PR1, on peut soit mettre en cascade 2 timers, soit programmer un compteur logiciel dans l'ISR `_T1Interrupt` qui sera incrémentée à chaque fois que le timer matériel aura compté PR1.
On lance le timer avec le bit 15 de T1CON.

FIGURE 9-1: 16-BIT TIMER1 MODULE BLOCK DIAGRAM (TYPE A TIMER)

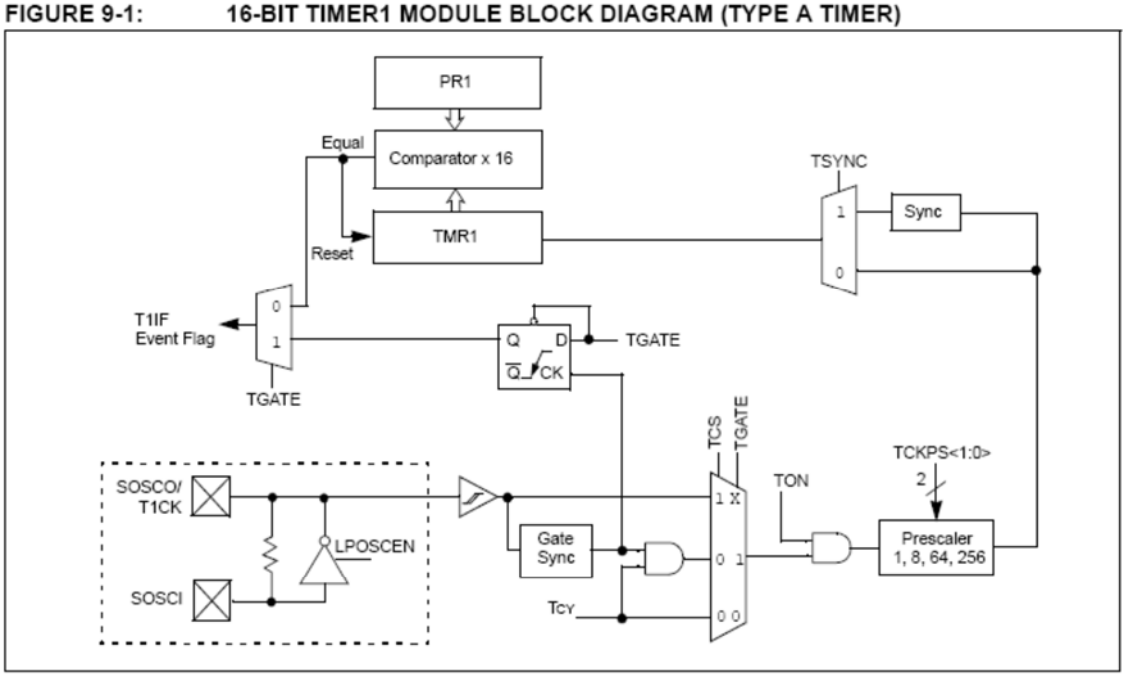


TABLE 9-1: TIMER1 REGISTER MAP(1)

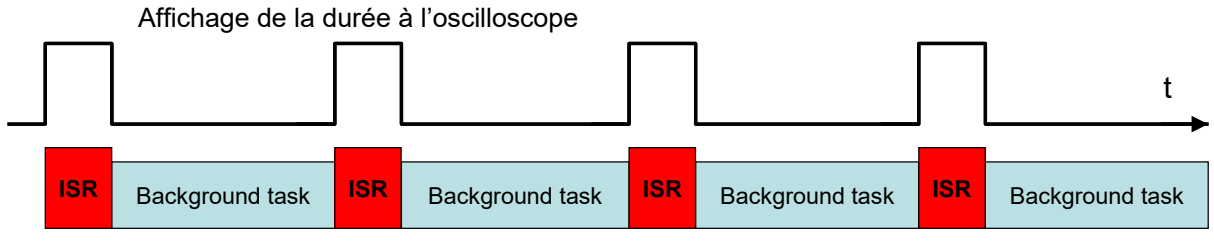
SFR Name	Addr.	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
TMR1	0100	Timer1 Register															
PR1	0102	Period Register 1															
T1CON	0104	TON	—	TSIDL	—	—	—	—	—	—	TGATE	TCKPS1	TCKPS0	—	TSYNC	TCS	—

Legend: u = uninitialized bit; — = unimplemented bit, read as '0'
Note 1: Refer to "*dsPIC30F Family Reference Manual*" (DS70046) for descriptions of register bit fields.

Interruption

Routine de Service de l'interruption
Interrupt Service Routine (ISR)

Configuration du Timer1 pour générer un comptage d'une période de 100 us et une interruption.
Observation des signaux sur oscilloscope.
Notion de IF, IEC, IPC, Routine de Service de l'Interruption (ISR).
Tâches synchrones. Notion de temps de calcul et de dépassement.



Tâche peu gourmande en temps CPU

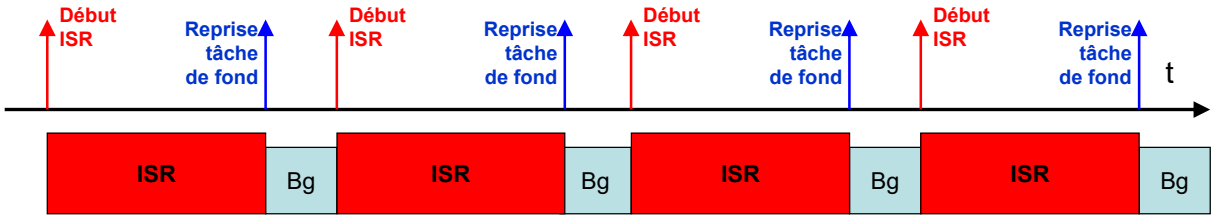


TABLE 5-2: INTERRUPT CONTROLLER REGISTER MAP(1)

SFR Name	ADR	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
INTCON1	0080	NSTDIS	—	—	—	—	OVATE	OVBTE	COVTE	—	—	—	MATHERR	ADDRERR	STKERR	OSCFAIL	—
INTCON2	0082	ALTIVT	DISI	—	—	—	—	—	—	—	—	—	—	—	INT2EP	INT1EP	INT0EP
IFS0	0084	CNIF	MI2CIF	SI2CIF	NVMIF	ADIF	U1TXIF	U1RXIF	SPI1IF	T3IF	T2IF	OC2IF	IC2IF	T1IF	OC1IF	IC1IF	INT0IF
IFS1	0086	—	—	—	—	—	—	U2TXIF	U2RXIF	INT2IF	T5IF	T4IF	OC4IF	OC3IF	IC8IF	IC7IF	INT1IF
IFS2	0088	—	—	—	—	FLTAIF	—	—	QEIIF	PWMIF	—	—	—	—	—	—	—
IEC0	008C	CNIE	MI2CIE	SI2CIE	NVMIE	ADIE	U1TXIE	U1RXIE	SPI1IE	T3IE	T2IE	OC2IE	IC2IE	T1IE	OC1IE	IC1IE	INT0IE
IEC1	008E	—	—	—	—	—	—	U2TXIE	U2RXIE	INT2IE	T5IE	T4IE	OC4IE	OC3IE	IC8IE	IC7IE	INT1IE
IEC2	0090	—	—	—	—	FLTAIE	—	—	QEIIE	PWMIE	—	—	—	—	—	—	—
IPC0	0094	—	T1IP<2:0>			—	OC1IP<2:0>			—	IC1IP<2:0>			—	INT0IP<2:0>		
IPC1	0096	—	T31P<2:0>			—	T2IP<2:0>			—	OC2IP<2:0>			—	IC2IP<2:0>		
IPC2	0098	—	ADIP<2:0>			—	U1TXIP<2:0>			—	U1RXIP<2:0>			—	SPI1IP<2:0>		
IPC3	009A	—	CNIP<2:0>			—	MI2CIP<2:0>			—	SI2CIP<2:0>			—	NVMIP<2:0>		
IPC4	009C	—	OC3IP<2:0>			—	IC8IP<2:0>			—	IC7IP<2:0>			—	INT1IP<2:0>		
IPC5	009E	—	INT2IP<2:0>			—	T5IP<2:0>			—	T4IP<2:0>			—	OC4IP<2:0>		
IPC6	00A0	—	—	—	—	—	—	—	—	—	U2TXIP<2:0>			—	U2RXIP<2:0>		
IPC7	00A2	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
IPC8	00A4	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
IPC9	00A6	—	PWMIP<2:0>			—	—	—	—	—	INT41IP<2:0>			—	INT3IP<2:0>		
IPC10	00A8	—	FLTAIP<2:0>			—	—	—	—	—	—	—	—	—	QEIIP<2:0>		
IPC11	00AA	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—

Legend: — = unimplemented, read as '0'
Note 1: Refer to "*dsPIC30F Family Reference Manual*" (DS70046) for descriptions of register bit fields.

TABLE 5-1: INTERRUPT VECTOR TABLE

INT Number	Vector Number	Interrupt Source
Highest Natural Order Priority		
0	8	INT0 – External Interrupt 0
1	9	IC1 – Input Capture 1
2	10	OC1 – Output Compare 1
3	11	T1 – Timer 1
4	12	IC2 – Input Capture 2
5	13	OC2 – Output Compare 2
6	14	T2 – Timer 2
7	15	T3 – Timer 3
8	16	SPI #1
9	17	U1RX – UART1 Receiver
10	18	U1TX – UART1 Transmitter
11	19	ADC – ADC Convert Done
12	20	NVM – NVM Write Complete
13	21	SI2C – I^2C™ Slave Interrupt
14	22	MI2C – I^2C Master Interrupt
15	23	Input Change Interrupt
16	24	INT1 – External Interrupt 1
17	25	IC7 – Input Capture 7
18	26	IC8 – Input Capture 8
19	27	OC3 – Output Compare 3*
20	28	OC4 – Output Compare 4*
21	29	T4 – Timer 4
22	30	T5 – Timer 5
23	31	INT2 – External Interrupt 2
24	32	U2RX – UART2 Receiver*
25	33	U2TX – UART2 Transmitter*
39	47	PWM – PWM Period Match
40	48	QEI – QEI Interrupt
41	49	Reserved
42	50	Reserved
43	51	$\overline{\text{FLTA}}$ – PWM Fault A
44	52	Reserved
45-53	53-61	Reserved
Lowest Natural Order Priority		

Exemple de code configuration du Timer1 + ISR

```
//-----------------------------------------------------------------------------
void initTimer(void)
{
// Timer1 pour l ISR des 200 us
   T1CON = 0;       // ensure Timer 1 is in reset state, no prescale
   TMR1  = 0;// RAZ Timer1
   _T1IF = 0;// reset Timer 1 interrupt flag
   _T1IP = 4;// set Timer1 interrupt priority level to 4
   _T1IE = 1;// enable Timer 1 interrupt
   PR1 = T1Period;               // set Timer 1 period register
   T1CONbits.TON = 1;     // enable Timer 1 and start the count
}

//-----------------------------------------------------------------------------
// Timer1 interrupt, ISR toutes les 200 us
//-----------------------------------------------------------------------
void __attribute__((interrupt, auto_psv)) _T1Interrupt( void )
{
   InfoLED = 1;
   _T1IF = 0;
   // do some work
   // ...
   // ...
   InfoLED = 0;
}
```

ADC

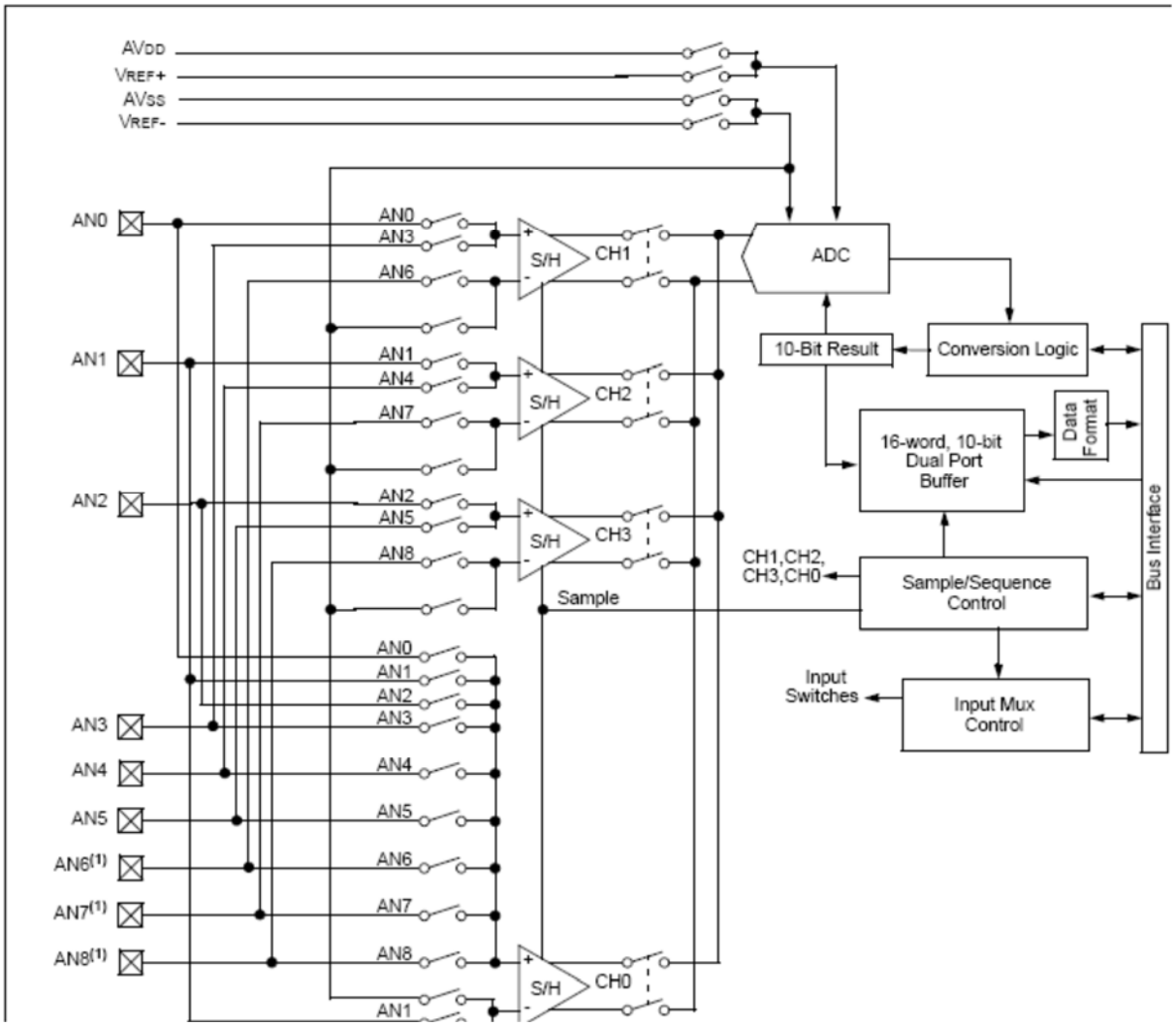
Le dspic 30F3010 comporte un seul ADC et 4 échantillonneurs bloqueurs (S/H). Ces derniers peuvent être lancés simultanément ou successivement.
Auparavant, l'ADC doit être configuré :

- Les entrées doivent être configurées en AN à l'aide du registre ADPCFG.
- Toute configuration doit être faite avec l'ADC arrêté : (bit 15 de ADCON1) _ADON = 0;
- ADCON1 = 0x0000;
- Le mode d'opération doit être spécifié : SOC manuel ou automatique, SOS manuel ou automatique (ADCON1 bit 2 **ASAM**), la manières dont les données sont sortis (ADCON1 bit 9-8 **FORM**) la source de lancement de l'ADC (ADCON1 bit 7-5 **SSRC).**
- Les canaux qui seront convertis (ADCON2 bit 9-8 **CHPS**), tous les combiens de séquence échantillonnage/conversion, une interruption ADC sera enclenchée (ADCON2 bit 5-2 **SMPI**), par défaut à chaque fois.
- Les S/H doivent être connectés aux entrées ANx à l'aide du registre ADCHS.
- Le TAD doit être choisi ADCON3 = 0x0080;
- L'ADC est rendu opérationnel (bit 15 de ADCON1) _ADON = 1;

Si on spécifie une interruption, on doit aussi configurer le _ADIF, _ADIE, éventullement la priorité, et ajouter la routine de service de l'interruption (ISR) spécifique. Elle doit lire les buffers de l'ADC.
Exemple de lecture de 2 buffers 0 et 1 :

```
void __attribute__((interrupt, auto_psv)) _ADCInterrupt ()
{
   InfoLED=1;
   _ADIF = 0;
   a = ADCBUF0<<2;  // 10 bits * 4 pour avoir 4096, Q12 : 4.12 pu
   b = ADCBUF1<<2;
   InfoLED=0;
}
```

FIGURE 19-1: 10-BIT HIGH-SPEED ADC FUNCTIONAL BLOCK DIAGRAM



PWM

Motor Control PWM

- Page 95 [2] et page 333 [3]
- Génère 6 signaux PWM synchronisés
- 6 = 3 phases x (Lo et Hi)
- Commande de MAS, MS, BLDC, MRV
- PWM centrée ou non
- Dead-time
- Output Override
- Evènements (SOC,…)

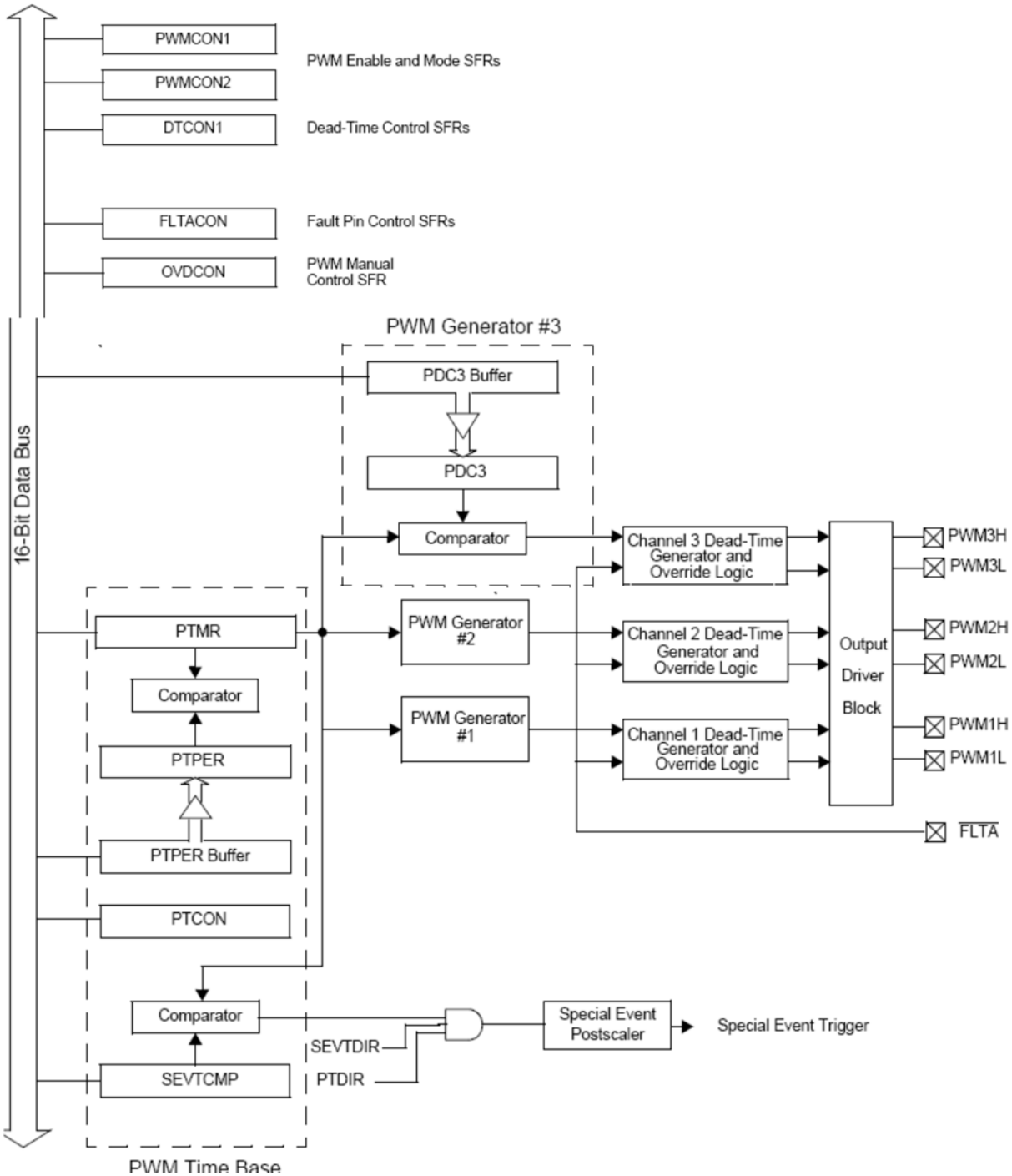


Figure 15-7: Edge-Aligned PWM

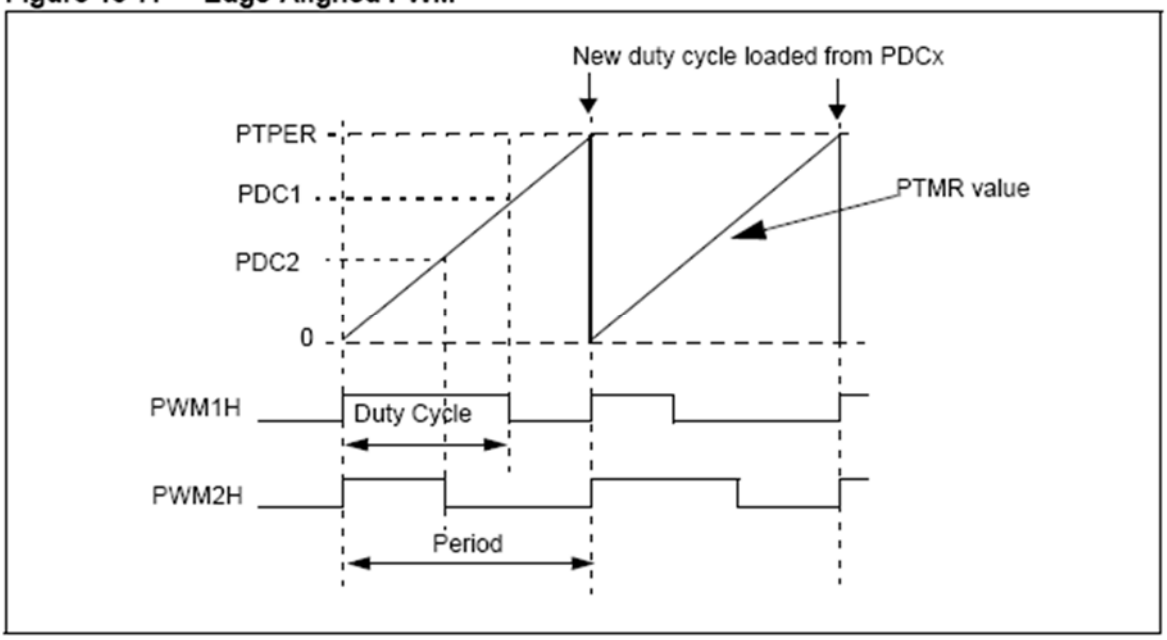


Figure 15-9: Center Aligned PWM

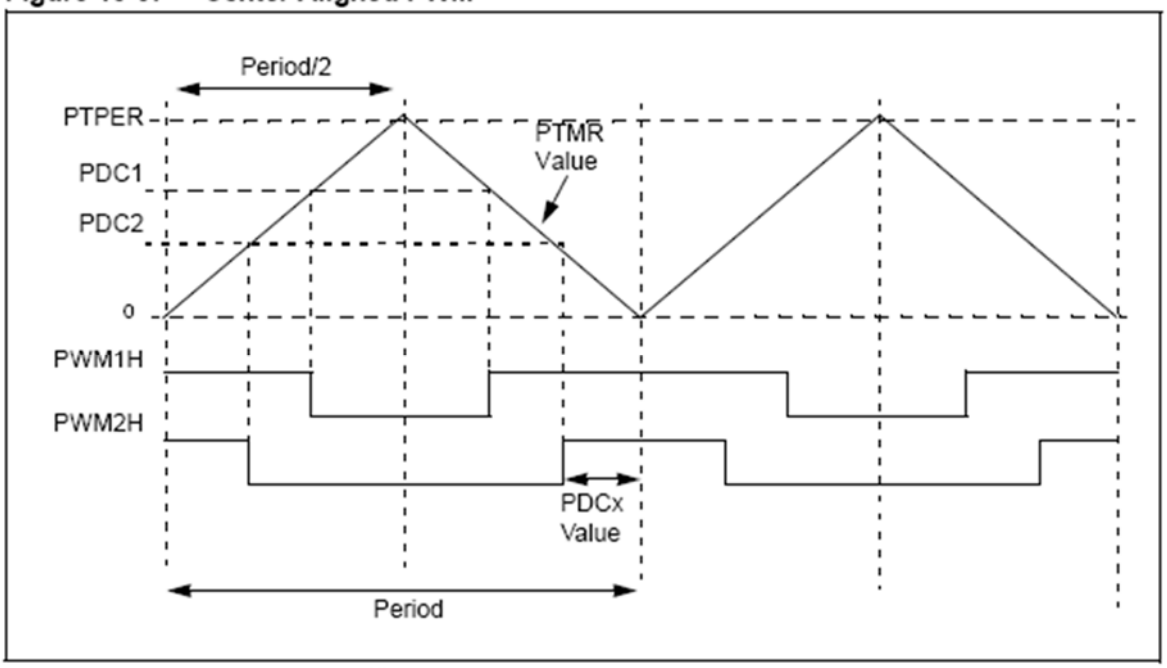


TABLE 15-1: PWM REGISTER MAP(1)

SFR Name	Addr.	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
PTCON	01C0	PTEN	—	PTSIDL	—	—	—	—	—	PTOPS<3:0>				PTCKPS<1:0>		PTMOD<1:0>	
PTMR	01C2	PTDIR	PWM Timer Count Value														
PTPER	01C4	—	PWM Time Base Period Register														
SEVTCMP	01C6	SEVTDIR	PWM Special Event Compare Register														
PWMCON1	01C8	—	—	—	—	—	PTMOD3	PTMOD2	PTMOD1	—	PEN3H	PEN2H	PEN1H	—	PEN3L	PEN2L	PEN1L
PWMCON2	01CA	—	—	—	—	SEVOPS<3:0>				—	—	—	—	—	—	OSYNC	UDIS
DTCON1	01CC	—	—	—	—	—	—	—	—	DTAPS<1:0>		Dead-Time A Value					
FLTACON	01D0	—	—	FAOV3H	FAOV3L	FAOV2H	FAOV2L	FAOV1H	FAOV1L	FLTAM	—	—	—	—	FAEN3	FAEN2	FAEN1
OVDCON	01D4	—	—	POVD3H	POVD3L	POVD2H	POVD2L	POVD1H	POVD1L	—	—	POUT3H	POUT3L	POUT2H	POUT2L	POUT1H	POUT1L
PDC1	01D6	PWM Duty Cycle 1 Register															
PDC2	01D8	PWM Duty Cycle 2 Register															
PDC3	01DA	PWM Duty Cycle 3 Register															

Legend: — = unimplemented bit, read as '0'

Note 1: Refer to "*dsPIC30F Family Reference Manual*" (DS70046) for descriptions of register bit fields.

Différence entre une PWM classique (de timer) et celle dédiée au contrôle moteur :

- Possibilité d'ajouter des temps morts entre la commande des transistors haut et bas afin d'éviter un court circuit du bras d'onduleur
- Entrée de défaut $\overline{\mathrm{FLTA}}$ qui permet de couper instantanément la commande de manière matérielle.
- Possibilité de la prise de contrôle manuel des commandes PWM (OVDCON)

Registres de configuration

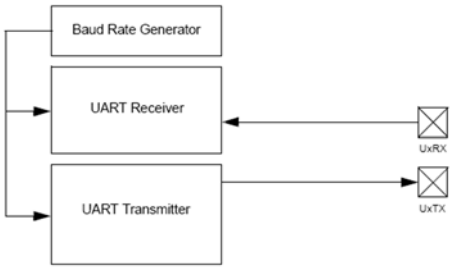
PWMCON1

Exemple en Annexes

UART

Description

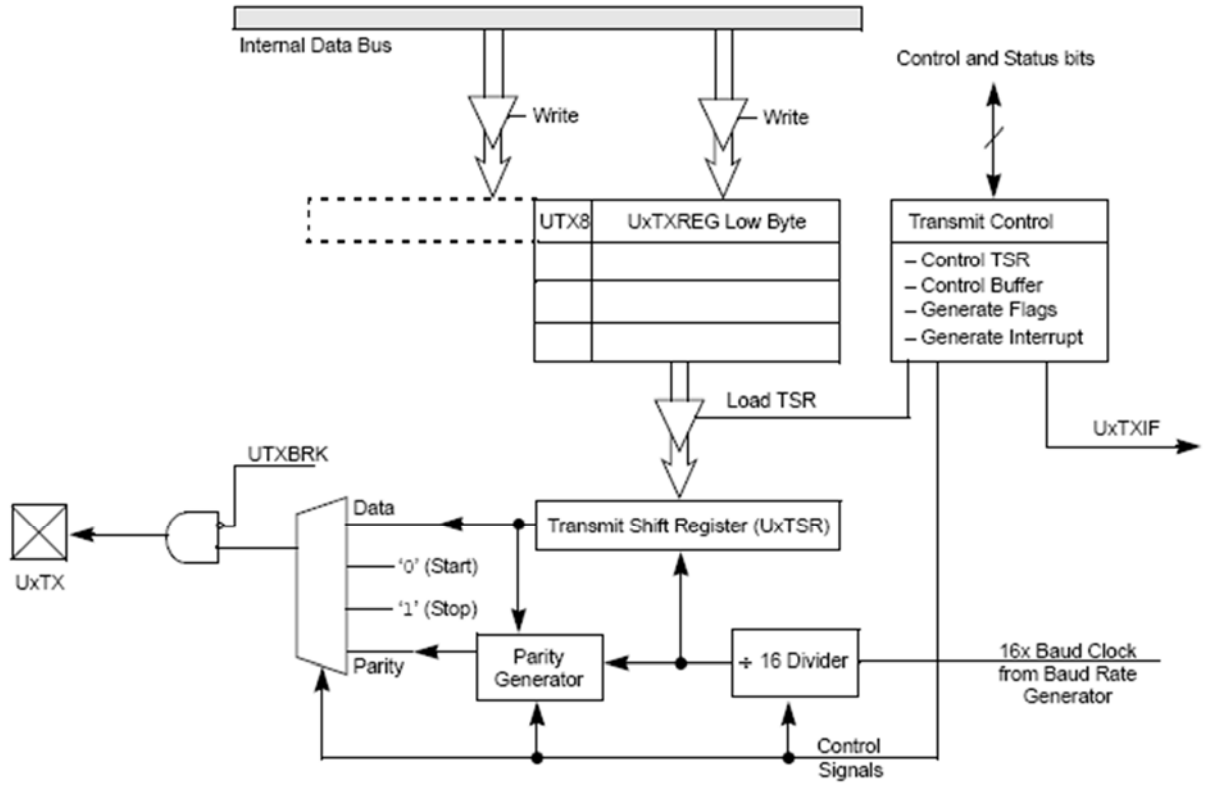
L'UART est un système asynchrone full-duplex qui peut communiquer avec des périphériques, comme des PC, des interfaces RS-232 et RS-485.



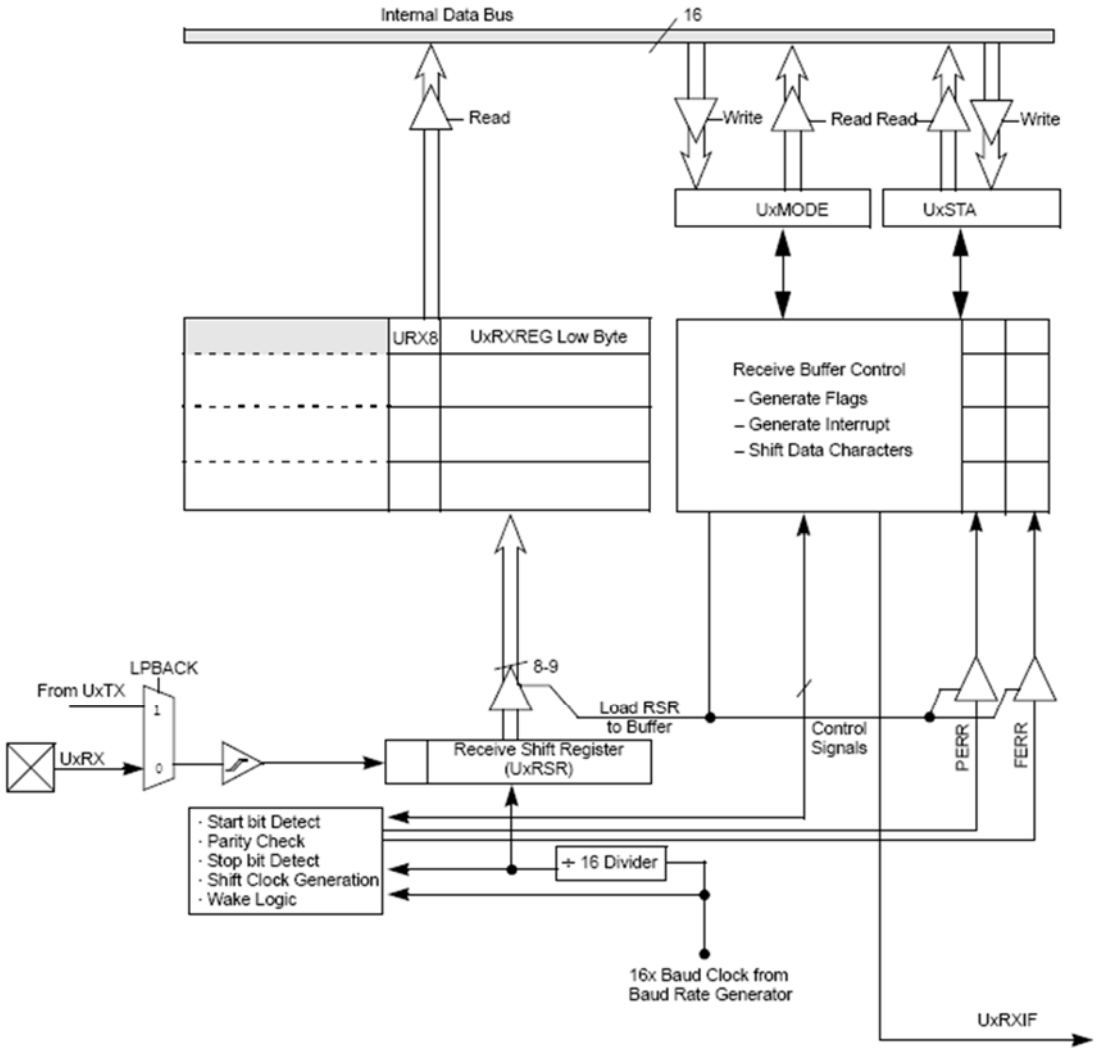
Features (caractéristiques)

- Full-duplex 8- or 9-bit data transmission through the UxTX and UxRX pins
- Even, Odd or No Parity options (for 8-bit data)
- One or two Stop bits
- Fully integrated Baud Rate Generator with 16-bit prescaler
- Baud rates ranging from 29 bps to 1.875 Mbps at FCY = 30 MHz
- 4-deep First-In-First-Out (FIFO) transmit data buffer
- 4-deep FIFO receive data buffer
- Parity, Framing and Buffer Overrun error detection
- Support for 9-bit mode with Address Detect (9th bit = 1)
- Transmit and Receive Interrupts
- Loopback mode for diagnostic support

UART TRANSMITTER BLOCK DIAGRAM



UART RECEIVER BLOCK DIAGRAM



- **Features (caractéristiques)**
  - Baud Rate = FCY / (16(BRG+1))
  - UxMODE : x =1 seule UART pour le 30F3010

TABLE 18-1: UART1 REGISTER MAP(1)

SFR Name	Addr.	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
U1MODE	020C	UARTEN	—	USIDL	—	—	ALTIO	—	—	WAKE	LPBACK	ABAUD	—	—	PDSEL1	PDSEL0	STSEL
U1STA	020E	UTXISEL	—	—	—	UTXBRK	UTXEN	UTXBF	TRMT	URXISEL1	URXISEL0	ADDEN	RIDLE	PERR	FERR	OERR	URXDA
U1TXREG	0210	—	—	—	—	—	—	—	UTX8	Transmit Register							
U1RXREG	0212	—	—	—	—	—	—	—	URX8	Receive Register							
U1BRG	0214	Baud Rate Generator Prescaler															

Legend: u = uninitialized bit; — = unimplemented bit, read as '0'

Note 1: Refer to "*dsPIC30F Family Reference Manual*" (DS70046) for descriptions of register bit fields.

```
void InitUART()
{
// Initialize the UART1 for BAUD
        U1MODE = 0x8400; // enable + alternate pins
//      U1MODE = 0x8000; // enable + normal pins
        U1STA = 0x0000;
        U1BRG = ((FCY/16)/BAUD) - 1;      // set baud to BAUD (9600 ou 19200 ou…)
        RXPtr = &InData[0];               // point to first char in receive buffer
        Flags.CheckRX = 0;                        // clear rx and tx flags
        IFS0bits.U1RXIF = 0;     // clear interrupt flag
        IEC0bits.U1RXIE = 1;                      // enable RX interrupt
        Flags.SendTX = 0;
        Flags.SendData = 0;               // clear flag
        SeqComm=SeqCommMax;
        U1STAbits.UTXEN = 1;           // Initiate transmission
}
//-----------------------------------------------------------------------
void __attribute__((interrupt, auto_psv)) _U1TXInterrupt(void)
{       IFS0bits.U1TXIF = 0;     }       // clear interrupt flag
```

```
//---------------------------------------------------------------------
void __attribute__((interrupt, auto_psv)) _U1RXInterrupt(void)
{       IFS0bits.U1RXIF = 0;      // clear interrupt flag
        *RXPtr = U1RXREG;
//      U1TXREG = *RXPtr; debug reflect any char received on the RX to the TX
        if (*RXPtr == CR   || *RXPtr == LF || RXPtr>=(&InData[0] +current_inputindexLimit))
                {
                Flags.CheckRX = 1;
                RXPtr = &InData[0];
                }
        else RXPtr++;
}
void SendMsg()
{
while (*TXPtr)
        {
        while (U1STAbits.UTXBF);
        U1TXREG = *TXPtr++;
        }
}
```

➢ Déclaration

```
// RS232 ----------------------------------------
unsigned char *TXPtr;
unsigned char *RXPtr;
#define CR       0x0D
#define LF       0x0A
//#define BAUD 19200
#define BAUD 57600        // for BT
unsigned char InData[] = {"00000000000000000000000000000000"};
//unsigned char current_inputindex;
#define current_inputindexLimit 31
unsigned char OutData[]  = {"$RBref=0000 Bmes=0000 Vlim=0000 Vref=0000 :R\r"};
#define OffsetBref 7                    // offset in OutData : position de la val de Bref
#define OffsetBmes 17                   // offset in OutData : position de la val de Bmes
#define OffsetVlim 27                   // offset in OutData : position de la val de Vlim
#define OffsetVref 37                   // offset in OutData : position de la val de Vref
#define Stateoffs  43                   // offset in OutData : position de la val de Running
int SeqComm;              // ttes les 0.5 s
#define SeqCommMax 500            // ttes les 0.0625 s
```

➢ Déclaration de la structure des variables de contrôles booléennes

```
struct  {
                unsigned Running : 1;
        unsigned CheckRX : 1;
        unsigned SendTX  : 1;
        unsigned SendData        : 1;
        unsigned unused  : 12;
        } Flags;                  // NOTE : Flags n'a rien à voir avec les interruptions
```

➢ Utilisation : Envoi

```
        TXPtr = &OutData[0];
        SendMsg();
```

➢ Utilisation : Réception

```
//---------------------------------------------------------------------
void ReceiveData()
{
        TXPtr = &InData[0];
        SendinputMsg();
        Flags.CheckRX=0;
        if (InData[0] != 'C')            return; // sinon, traite la commande
        switch (InData[1])
                {
                case 'E' :       if (InData[2] == '1')    Flags.Running=1;
                                 else     Flags.Running=0;
                           EnableL6234=Flags.Running; // enable or not L6234 principal
                           RunningLED = Flags.Running; // set or clear running LED
                           UpdateBufferRunningStatus(Flags.Running);
                           break;
                case 'M' :       if (InData[2] == '1')    { Flags.UARTBref=1;      Bref=BrefUart; }
                                 else     { Flags.UARTBref=0; Bref=ADCBUF0; }
                case 'F' :       if (InData[2] == '1')    WriteEEPROM();
                                 else     ReadEEPROM();
                case 'R' :       BrefUart =ConvHexaToInt( &InData[2]);
                           break;
                case 'P' :       lreg.Field.Kp=ConvHexaToDouble( &InData[2]);
                           break;
                case 'I' :       lreg.Field.Ki=ConvHexaToDouble( &InData[2]);
                           break;
                case 'D' :       lreg.Field.Kd=ConvHexaToDouble( &InData[2]);
                           break;
                }
```

```
}
```

Thèmes non abordés

- **dspic 30F3010**
  - SPI : Serial Peripheral Interface : Liaison série synchrone (avec Clk) rapide
  - I2C : liaison série
  - Bus CAN : liaison série utilisée en industrie (automobile et autres), voir le dspic 30F4012
  - Modes d'économie d'énergie (IDLE et SLEEP)
  - EEPROM : pour garder des informations d'utilisateur (comme les paramètres de réglage d'un PID…) après coupure de l'alimentation
  - QEI : Quadrature Encoder Interface, pour brancher un codeur incrémental de mesure de l'incrément de position et de la vitesse
- **Autres systèmes**
  - DSP : Systèmes numériques de traitement spécialisés, cartes dSPACE. Les principes restent les mêmes que les µC mais ils sont plus puissants
  - API : Automates Programmables, ne conviennent pas pour la commande rapprochée car ils ne sont pas assez rapides mais ils servent d'interfaces entre les commandes et les superviseurs

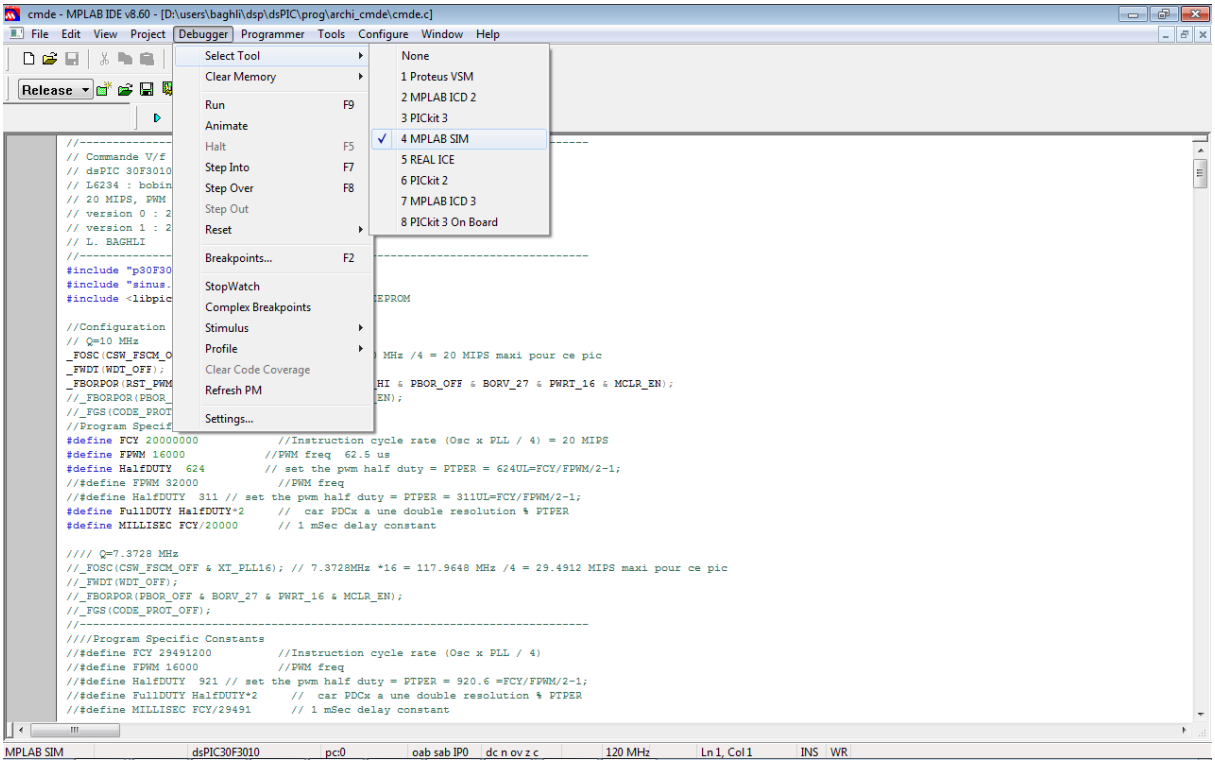
Plan des Travaux Pratiques (EE812)

Voir document séparé.

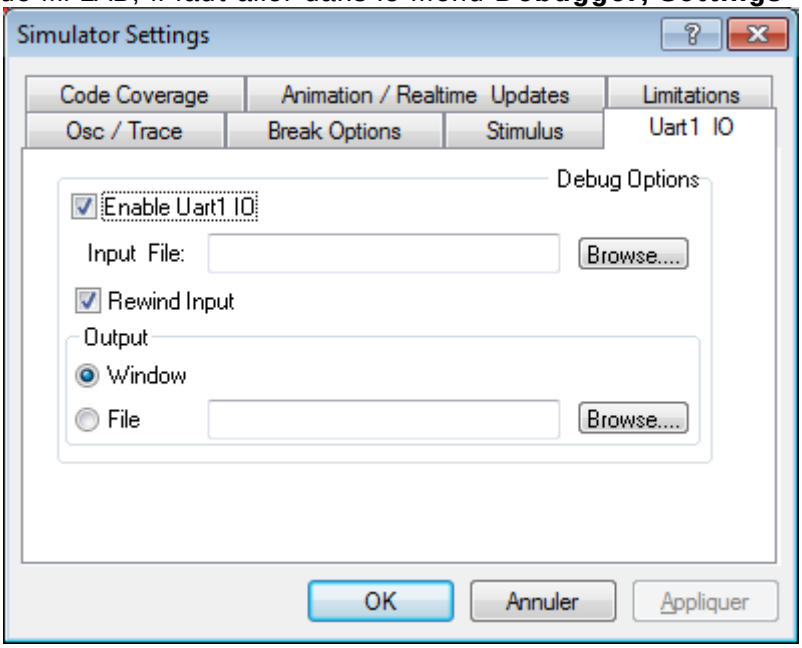
Annexes

Initiation à MPLAB

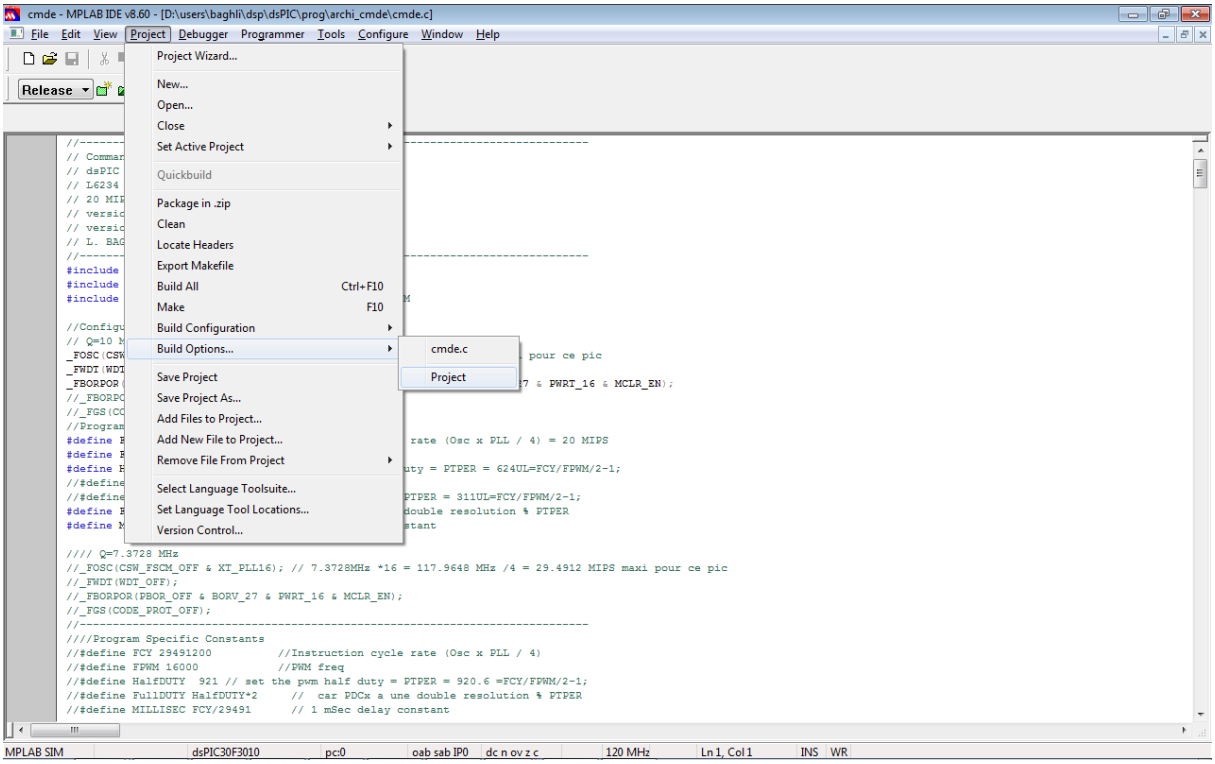
Afin de déboguer un programme, en mode de simulation, sans dspic ni outil de programmation pickit2, il suffit de simuler la presence du dspic par MPLAB.
Aller dans le menu **Debugger, Select Tool,** choisir **MPLAB SIM**

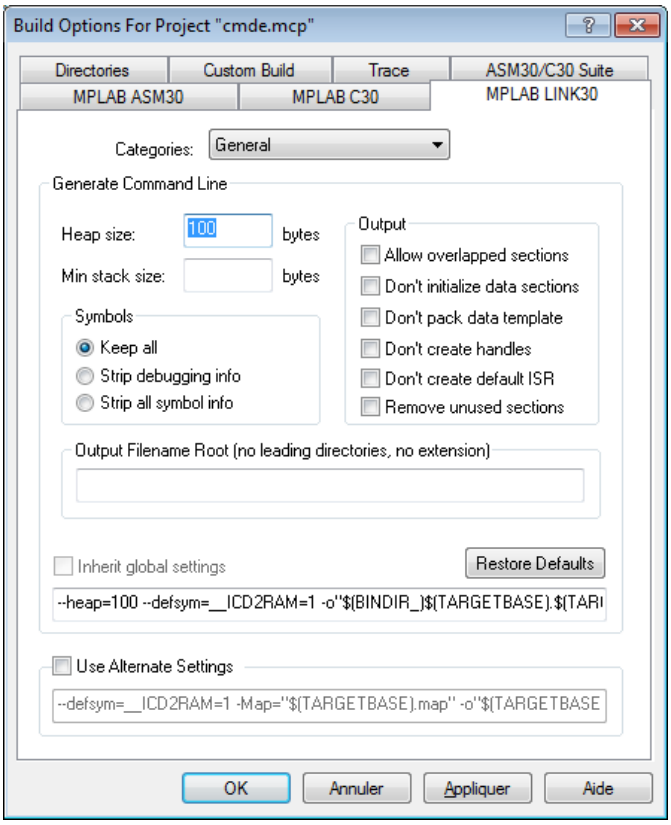


Pour pouvoir rediriger les sorties de printf sur l'UART et visualiser sur la fenêtre Output de MPLAB, il faut aller dans le menu **Debugger, Settings**



Il faut aussi augmenter la taille de la pile, qui par défaut est nulle, pour que la fonction printf puisse correctement fonctionner.
Aller dans le menu **Project, Build option, Project**

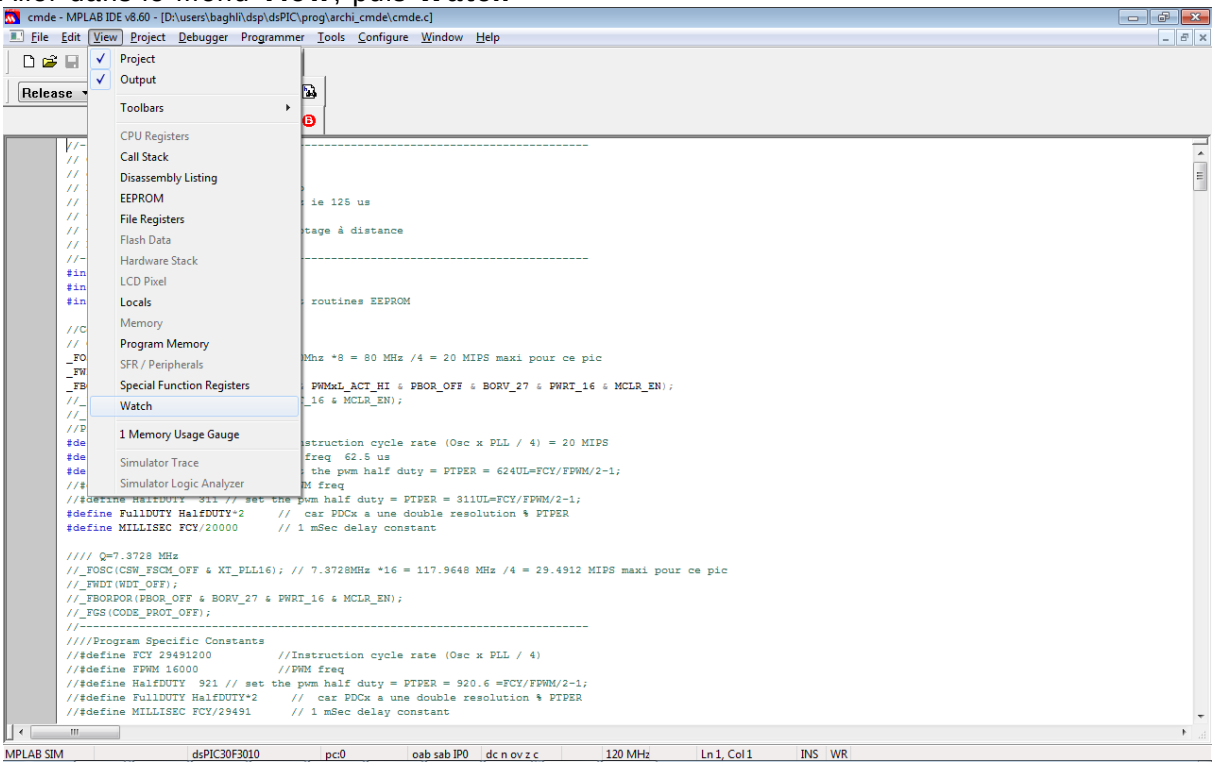


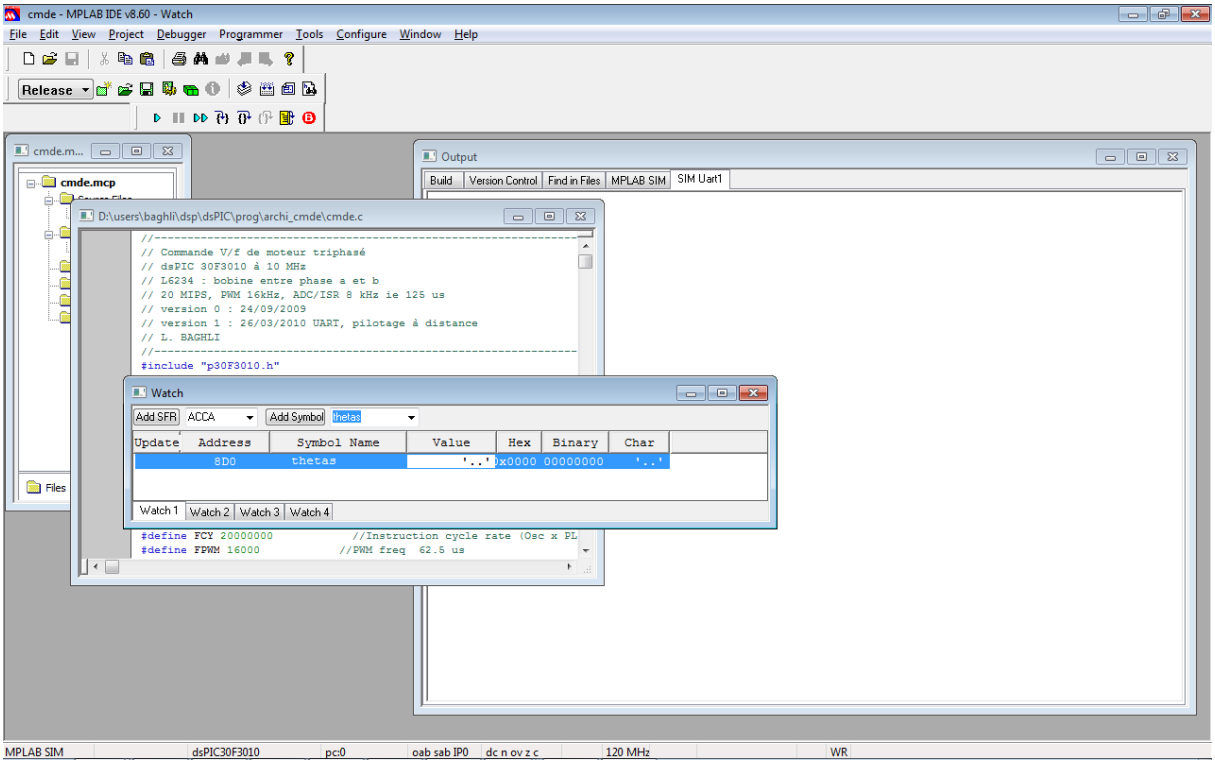


Dans l'onglet **MPLAB LINK30**, mettre la pile à 100 (Heap à 100)

Puis ajouter une fenêtre permettant de regarder le contenu des variables du programme pendant l'exécution.

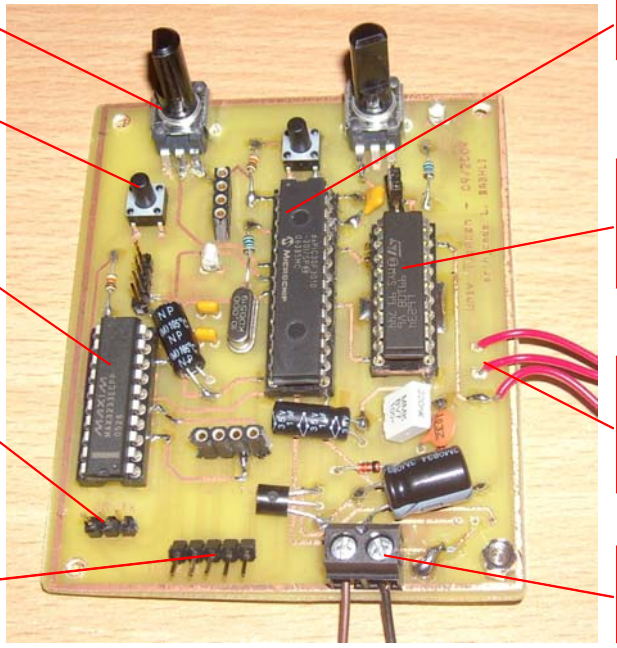
Aller dans le menu **View**, puis **Watch**





À côté de "Add Symbol" choisir la variable à observer, puis faire "Add Symbol". Positionner la fenêtre de manière à ne pas gêner la vision de la fenêtre du code source pendant le débogage du programme.

Présentation de la carte électronique



Timer

Register 12-1: TxCON: Type A Time Base Register

Upper Byte:							
R/W-0	U-0	R/W-0	U-0	U-0	U-0	U-0	U-0
TON	—	TSIDL	—	—	—	—	—
bit 15							bit 8

Lower Byte:							
U-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0	R/W-0	U-0
—	TGATE	TCKPS<1:0>		—	TSYNC	TCS	—
bit 7							bit 0

bit 15 **TON:** Timer On Control bit
- 1 = Starts the timer
- 0 = Stops the timer

bit 14 **Unimplemented:** Read as '0'

bit 13 **TSIDL:** Stop in Idle Mode bit
- 1 = Discontinue timer operation when device enters Idle mode
- 0 = Continue timer operation in Idle mode

bit 12-7 **Unimplemented:** Read as '0'

bit 6 **TGATE:** Timer Gated Time Accumulation Enable bit
- 1 = Gated time accumulation enabled
- 0 = Gated time accumulation disabled

(TCS must be set to '0' when TGATE = 1. Reads as '0' if TCS = 1)

bit 5-4 **TCKPS<1:0>:** Timer Input Clock Prescale Select bits
- 11 = 1:256 prescale value
- 10 = 1:64 prescale value
- 01 = 1:8 prescale value
- 00 = 1:1 prescale value

bit 3 **Unimplemented:** Read as '0'

bit 2 **TSYNC:** Timer External Clock Input Synchronization Select bit
- When TCS = 1:
- 1 = Synchronize external clock input
- 0 = Do not synchronize external clock input
- When TCS = 0:
- This bit is ignored. Read as '0'. Timer1 uses the internal clock when TCS = 0.

bit 1 **TCS:** Timer Clock Source Select bit
- 1 = External clock from pin TxCK
- 0 = Internal clock (FOSC/4)

bit 0 **Unimplemented:** Read as '0'

Legend:

R = Readable bit W = Writable bit U = Unimplemented bit, read as '0'

-n = Value at POR '1' = Bit is set '0' = Bit is cleared x = Bit is unknown

ADC

Configuration de l'ADC.
Lancement d'une conversion immédiate.
Observation des signaux sur oscilloscope et des résultats dans les registres du Buffer de l'ADC.
Configuration pour une conversion synchronisé sur le Timer1
Configuration pour une conversion synchronisé sur la PWM (en TP7).
ADC EOC ISR.
ADPCFG : choix des pins en E/S ou en analogique.

TABLE 19-2: ADC REGISTER MAP(1)

SFR Name	Addr.	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ADCBUF0	0280	ADC Data Buffer 0															
ADCBUF1	0282	ADC Data Buffer 1															
ADCBUF2	0284	ADC Data Buffer 2															
ADCBUF3	0286	ADC Data Buffer 3															
ADCBUF4	0288	ADC Data Buffer 4															
ADCBUF5	028A	ADC Data Buffer 5															
ADCBUF6	028C	ADC Data Buffer 6															
ADCBUF7	028E	ADC Data Buffer 7															
ADCBUF8	0290	ADC Data Buffer 8															
ADCBUF9	0292	ADC Data Buffer 9															
ADCBUFA	0294	ADC Data Buffer 10															
ADCBUFB	0296	ADC Data Buffer 11															
ADCBUFC	0298	ADC Data Buffer 12															
ADCBUFD	029A	ADC Data Buffer 13															
ADCBUFE	029C	ADC Data Buffer 14															
ADCBUFF	029E	ADC Data Buffer 15															
ADCON1	02A0	ADON	—	ADSIDL	—	—	—	FORM<1:0>		SSRC<2:0>			—	SIMSAM	ASAM	SAMP	DONE
ADCON2	02A2	VCFG<2:0>			—	—	CSCNA	CHPS<1:0>		BUFS	—	SMPI<3:0>				BUFM	ALTS
ADCON3	02A4	—	—	—	SAMC<4:0>					ADRC	—	ADCS<5:0>					
ADCHS	02A6	CH123NB<1:0>		CH123SB	CH0NB	CH0SB<3:0>				CH123NA<1:0>		CH123SA	CH0NA	CH0SA<3:0>			
ADPCFG	02A8	—	—	—	—	—	—	—	PCFG8(2)	PCFG7(2)	PCFG6(2)	PCFG5	PCFG4	PCFG3	PCFG2	PCFG1	PCFG0
ADCSSL	02AA	—	—	—	—	—	—	—	CSSL8(2)	CSSL7(2)	CSSL6(2)	CSSL5	CSSL4	CSSL3	CSSL2	CSSL1	CSSL0

Legend: u = uninitialized bit; — = unimplemented bit, read as '0'
Note 1: Refer to "*dsPIC30F Family Reference Manual*" (DS70046) for descriptions of register bit fields.
2: These bits are not available on dsPIC30F3010 devices.

Registres :

Register 17-1: ADCON1: A/D Control Register 1

Upper Byte:

R/W-0	U-0	R/W-0	U-0	U-0	U-0	R/W-0	R/W-0
ADON	—	ADSIDL	—	—	—	FORM<1:0>	
bit 15							bit 8

Lower Byte:

R/W-0	R/W-0	R/W-0	U-0	R/W-0	R/W-0	R/W-0 HC, HS	R/C-0 HC, HS
SSRC<2:0>			—	SIMSAM	ASAM	SAMP	DONE
bit 7							bit 0

bit 15 **ADON:** A/D Operating Mode bit
1 = A/D converter module is operating
0 = A/D converter is off
bit 14 **Unimplemented:** Read as '0'
bit 13 **ADSIDL:** Stop in Idle Mode bit
1 = Discontinue module operation when device enters Idle mode
0 = Continue module operation in Idle mode
bit 12-10 **Unimplemented:** Read as '0'
bit 9-8 **FORM<1:0>:** Data Output Format bits
11 = Signed Fractional (DOUT = sddd dddd dd00 0000)
10 = Fractional (DOUT = dddd dddd dd00 0000)
01 = Signed Integer (DOUT = ssss sssd dddd dddd)
00 = Integer (DOUT = 0000 00dd dddd dddd)
bit 7-5 **SSRC<2:0>:** Conversion Trigger Source Select bits

111 = Internal counter ends sampling and starts conversion (auto convert)
110 = Reserved
101 = Reserved
100 = Reserved
011 = Motor Control PWM interval ends sampling and starts conversion
010 = GP Timer3 compare ends sampling and starts conversion
001 = Active transition on INT0 pin ends sampling and starts conversion
000 = Clearing SAMP bit ends sampling and starts conversion

bit 4 **Unimplemented:** Read as '0'

bit 3 **SIMSAM:** Simultaneous Sample Select bit (only applicable when CHPS = 01 or 1x)
1 = Samples CH0, CH1, CH2, CH3 simultaneously (when CHPS = 1x)
or
Samples CH0 and CH1 simultaneously (when CHPS = 01)
0 = Samples multiple channels individually in sequence

bit 2 **ASAM:** A/D Sample Auto-Start bit
1 = Sampling begins immediately after last conversion completes. SAMP bit is auto set
0 = Sampling begins when SAMP bit set

bit 1 **SAMP:** A/D Sample Enable bit
1 = At least one A/D sample/hold amplifier is sampling
0 = A/D sample/hold amplifiers are holding
When ASAM = 0, writing '1' to this bit will start sampling
When SSRC = 000, writing '0' to this bit will end sampling and start conversion

bit 0 **DONE:** A/D Conversion Status bit (Rev. B silicon or later)
1 = A/D conversion is done
0 = A/D conversion is NOT done
Cleared by software or start of a new conversion
Clearing this bit will not effect any operation in progress

Register 17-2: ADCON2: A/D Control Register 2

Upper Byte:

R/W-0	R/W-0	R/W-0	U-0	U-0	R/W-0	R/W-0	R/W-0
VCFG<2:0>			reserved	—	CSCNA	CHPS<1:0>	
bit 15							bit 8

Lower Byte:

R-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
BUFS	—	SMPI<3:0>				BUFM	ALTS
bit 7							bit 0

bit 15-13 **VCFG<2:0>:** Voltage Reference Configuration bits

bit 12 **Reserved:** User should write '0' to this location

bit 11 **Unimplemented:** Read as '0'

bit 10 **CSCNA:** Scan Input Selections for CH0+ S/H Input for MUX A Input Multiplexer Setting bit
1 = Scan inputs
0 = Do not scan inputs

bit 9-8 **CHPS<1:0>:** Selects Channels Utilized bits
1x = Converts CH0, CH1, CH2 and CH3
01 = Converts CH0 and CH1
00 = Converts CH0
When SIMSAM bit (ADCON1<3>) = 0 multiple channels sampled sequentially
When SMSAM bit (ADCON1<3>) = 1 multiple channels sampled as in CHPS<1:0>

bit 7 **BUFS:** Buffer Fill Status bit
Only valid when BUFM = 1 (ADRES split into 2 x 8-word buffers).
1 = A/D is currently filling buffer 0x8-0xF, user should access data in 0x0-0x7
0 = A/D is currently filling buffer 0x0-0x7, user should access data in 0x8-0xF

bit 6 **Unimplemented:** Read as '0'

bit 5-2 **SMPI<3:0>:** Sample/Convert Sequences Per Interrupt Selection bits
1111 = Interrupts at the completion of conversion for each 16th sample/convert sequence
1110 = Interrupts at the completion of conversion for each 15th sample/convert sequence
.....
0001 = Interrupts at the completion of conversion for each 2nd sample/convert sequence
0000 = Interrupts at the completion of conversion for each sample/convert sequence

bit 1 **BUFM:** Buffer Mode Select bit
1 = Buffer configured as two 8-word buffers ADCBUF(15...8), ADCBUF(7...0)
0 = Buffer configured as one 16-word buffer ADCBUF(15...0.)

bit 0 **ALTS:** Alternate Input Sample Mode Select bit

1 = Uses MUX A input multiplexer settings for first sample, then alternate between MUX B and MUX A input
multiplexer settings for all subsequent samples
0 = Always use MUX A input multiplexer settings

Exemple de code (du cours) configuration de l'ADC, SOC manuel, SOS automatique

```
void InitADC10()
{
   // config ADC en lancement immédiat
   ADCON1 = 0x0004; // 4 ch simultanés, ADC Off for configuring
                    // ASAM bit = 1 implies sampling starts immediately after last
                    // conversion is done
   ADCON2 = 0x0200; // simulataneous sample 4 channels
                    // ADC INTerrupt à chaque EOC=100 us
   ADCHS  = 0x0022; // AN2/RB2 Ch0, AN3/RB3 Ch1, AN4/RB4 Ch2 NC, AN5/RB5 Ch3 NC
   ADCON3 = 0x0080; // Tad = internal RC (4uS)
   _ADON = 1;       // turn ADC ON
}
//---------------------------------------------------------------------------
//Main routine
int main()
{
   setup_ports();
   InitADC10();
   RunningLED=1;                                    // LED run = On
   RunningLED=0;                                    // LED run = Off
   while(1)
      {
      DelayNmSec(100);
      ADCON1bits.SAMP = 0;              // start conversion
      while(!ADCON1bits.DONE) {} // attend la fin
      ValAdc=ADCBUF0;
      }// end of while (1)
}      // end of main
//---------------------------------------------------------------------------
```

Exemple de code (du cours) configuration de l'ADC, SOC manuel, SOS manuel

```
void InitADC10()
{
   // config ADC en lancement immédiat
   ADCON1 = 0x0000; // 4 ch simultanés, ADC Off for configuring
             // ASAM bit = 0 implies SAMP=0 ends sampling and start converting
   ADCON2 = 0x0200; // simulataneous sample 4 channels
                    // ADC INTerrupt à chaque EOC=100 us
   ADCHS  = 0x0022; // AN2/RB2 Ch0, AN3/RB3 Ch1, AN4/RB4 Ch2 NC, AN5/RB5 Ch3 NC
   ADCON3 = 0x0080; // Tad = internal RC (4uS)
   _ADON = 1;       // turn ADC ON
}
//---------------------------------------------------------------------------
//Main routine
int main()
{
   setup_ports();
   InitADC10();
   RunningLED=1;                                    // LED run = On
   RunningLED=0;                                    // LED run = Off
   while(1)
      {
```

```
      ADCON1bits.SAMP =1;              // start sampling
      DelayNmSec(100);
      ADCON1bits.SAMP = 0;             // start conversion
      while(!ADCON1bits.DONE) {} // attend la fin
      ValAdc=ADCBUF0;
      }// end of while (1)
}      // end of main
```

PWM

Registres :

Register 15-1: PTCON: PWM Time Base Control Register

Upper Byte:							
R/W-0	U-0	R/W-0	U-0	U-0	U-0	U-0	U-0
PTEN	—	PTSIDL	—	—	—	—	—
bit 15							bit 8

Lower Byte:							
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
PTOPS<3:0>				PTCKPS<1:0>		PTMOD<1:0>	
bit 7							bit 0

bit 15 **PTEN:** PWM Time Base Timer Enable bit
- 1 = PWM time base is ON
- 0 = PWM time base is OFF

bit 14 **Unimplemented:** Read as '0'

bit 13 **PTSIDL:** PWM Time Base Stop in Idle Mode bit
- 1 = PWM time base halts in CPU Idle mode
- 0 = PWM time base runs in CPU Idle mode

bit 12-8 **Unimplemented:** Read as '0'

bit 7-4 **PTOPS<3:0>:** PWM Time Base Output Postscale Select bits
- 1111 = 1:16 Postscale
- •
- •
- 0001 = 1:2 Postscale
- 0000 = 1:1 Postscale

bit 3-2 **PTCKPS<1:0>:** PWM Time Base Input Clock Prescale Select bits
- 11 = PWM time base input clock period is 64 TCY (1:64 prescale)
- 10 = PWM time base input clock period is 16 TCY (1:16 prescale)
- 01 = PWM time base input clock period is 4 TCY (1:4 prescale)
- 00 = PWM time base input clock period is TCY (1:1 prescale)

bit 1-0 **PTMOD<1:0>:** PWM Time Base Mode Select bits
- 11 = PWM time base operates in a continuous up/down mode with interrupts for double PWM updates
- 10 = PWM time base operates in a continuous up/down counting mode
- 01 = PWM time base operates in single event mode
- 00 = PWM time base operates in a free running mode

Register 15-3: PTPER: PWM Time Base Period Register

Upper Byte:							
U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	PTPER <14:8>						
bit 15							bit 8

Lower Byte:							
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
PTPER <7:0>							
bit 7							bit 0

bit 15 **Unimplemented:** Read as '0'

bit 14-0 **PTPER<14:0>:** PWM Time Base Period Value bits

Register 15-2: PTMR: PWM Time Base Register

Upper Byte:							
R-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
PTDIR	PTMR <14:8>						
bit 15							bit 8

Lower Byte:							
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
PTMR <7:0>							
bit 7							bit 0

bit 15 **PTDIR:** PWM Time Base Count Direction Status bit (Read Only)
1 = PWM time base is counting down
0 = PWM time base is counting up
bit 14-0 **PTMR <14:0>:** PWM Timebase Register Count Value

Register 15-4: SEVTCMP: Special Event Compare Register

Upper Byte:							
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
SEVTDIR	SEVTCMP <14:8>						
bit 15							bit 8

Lower Byte:							
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
SEVTCMP <7:0>							
bit 7							bit 0

bit 15 **SEVTDIR:** Special Event Trigger Time Base Direction bit**(1)**
1 = A special event trigger will occur when the PWM time base is counting downwards.
0 = A special event trigger will occur when the PWM time base is counting upwards.
bit 14-0 **SEVTCMP <14:0>:** Special Event Compare Value bit**(2)**
Note 1: SEVTDIR is compared with PTDIR (PTMR<15>) to generate the special event trigger.
2: SEVTCMP<14:0> is compared with PTMR<14:0> to generate the special event trigger.

Register 15-5: PWMCON1: PWM Control Register 1

Upper Byte:							
U-0	U-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	—	PMOD4	PMOD3	PMOD2	PMOD1
bit 15							bit 8

Lower Byte:							
R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
PEN4H	PEN3H	PEN2H	PEN1H	PEN4L	PEN3L	PEN2L	PEN1L
bit 7							bit 0

bit 15-12 **Unimplemented:** Read as '0'
bit 11-8 **PMOD4:PMOD1:** PWM I/O Pair Mode bits
1 = PWM I/O pin pair is in the independent output mode
0 = PWM I/O pin pair is in the complementary output mode
bit 7-4 **PEN4H-PEN1H:** PWMxH I/O Enable bits**(1)**
1 = PWMxH pin is enabled for PWM output
0 = PWMxH pin disabled. I/O pin becomes general purpose I/O
bit 3-0 **PEN4L-PEN1L:** PWMxL I/O Enable bits**(1)**
1 = PWMxL pin is enabled for PWM output
0 = PWMxL pin disabled. I/O pin becomes general purpose I/O
Note 1: Reset condition of the PENxH and PENxL bits depend on the value of the PWM/PIN device configuration bit in the FBORPOR Device Configuration Register.

Register 15-11: OVDCON: Override Control Register

Upper Byte:							
R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
POVD4H	POVD4L	POVD3H	POVD3L	POVD2H	POVD2L	POVD1H	POVD1L
bit 15							bit 8

Lower Byte:							
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
POUT4H	POUT4L	POUT3H	POUT3L	POUT2H	POUT2L	POUT1H	POUT1L
bit 7							bit 0

bit 15-8 **POVD4H-POVD1L:** PWM Output Override bits
1 = Output on PWMxx I/O pin is controlled by the PWM generator
0 = Output on PWMxx I/O pin is controlled by the value in the corresponding POUTxx bit
bit 7-0 **POUT4H-POUT1L:** PWM Manual Output bits
1 = PWMxx I/O pin is driven ACTIVE when the corresponding POVDxx bit is cleared
0 = PWMxx I/O pin is driven INACTIVE when the corresponding POVDxx bit is cleared

Register 15-12: PDC1: PWM Duty Cycle Register 1

Upper Byte:							
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
PWM Duty Cycle #1 bits 15-8							
bit 15							bit 8

Lower Byte:							
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
PWM Duty Cycle #1 bits 7-0							
bit 7							bit 0

bit 15-0 **PDC1<15:0>:** PWM Duty Cycle #1 Value bits

Register 15-16: FBORPOR: BOR AND POR Device Configuration Register

Upper Byte:							
U-0	U-0	U-0	U-0	U-0	U-0	U-0	U-0
—	—	—	—	—	—	—	—
bit 23							bit 16

Middle Byte:							
U-0	U-0	U-0	U-0	U-0	R/P	R/P	R/P
—	—	—	—	—	PWMPIN	HPOL	LPOL
bit 15							bit 8

Lower Byte:							
R/P	U-0	R/P	R/P	U-0	U-0	R/P	R/P
BOREN	—	BORV<1:0>		—	—	FPWRT<1:0>	
bit 7							bit 0

bit 10 **PWMPIN:** MPWM Drivers Initialization bit
1 = Pin state at reset controlled by I/O Port (PWMCON1<7:0> = 0x00)
0 = Pin state at reset controlled by module (PWMCON1<7:0> = 0xFF)
bit 9 **HPOL:** MCPWM High Side Drivers (PWMxH) Polarity bit
1 = Output signal on PWMxH pins has active high polarity
0 = Output signal on PWMxH pins has active low polarity
bit 8 **LPOL:** MCPWM Low Side Drivers (PWMxL) Polarity bit
1 = Output signal on PWMxL pins has active high polarity
0 = Output signal on PWMxL pins has active low polarity

Attention : Ce registre est configuré par les bits de configurations
```
_FBORPOR(RST_PWMPIN & PWMxH_ACT_LO & PWMxL_ACT_HI & PBOR_OFF & BORV_27 & PWRT_16 & MCLR_EN);
```
en début du programme.

Exemple d'utilisation de la PWM
Routine d'initialisation :

```
void InitMCPWM()
{
   PTPER = HalfDUTY; // set the pwm period register, ne pas oublier la double
précision
// OVDCON = 0xFFFF;  // Cmde MLI, no effect of OVDCON
   OVDCON = 0x00FF;  // allow control using OVD register (active low : PWM pins ie 0)
// PWMCON1 = 0x0700; // disable PWMs
   PWMCON1= 0x0770;  // enable PWM outputs (mais pas les LOW)
   PDC1=HalfDUTY; PDC2=HalfDUTY; PDC3=HalfDUTY;     // init rapport cyclique 50%
   EnableL6234=0;  // disable L298
   PWMCON2 = 0x0000; // 1 PWM values
   PTCON = 0x8002; // start PWM symetrique
}
//-----------------------------------------------------------------------
```

Instruction de blocage ou de validation :

```
OVDCON = 0xFFFF;  // Cmde MLI, no effect of OVDCON

OVDCON = 0x00FF;  // allow control using OVD register (active low : PWM pins ie 0)
```

Exemple d'ADC SOC + PWM :
Cet exemple montre l'utilisation de l'ADC et sa synchronisation avec la PWM.
Le lancement de conversion (SOC) se fait sur "Period Match" à l'aide du registre SEVTCMP.
L'interruption de l'ADC est programmée afin que l'ISR soit appelé en fin de conversion (EOC)

```
//-----------------------------------------------------------------------------
// Setup the ADC registers for :
// 4 channels conversion
// PWM trigger starts conversion
// AD interrupt is set and buffer is read in the interrupt
//-----------------------------------------------------------------------------
void InitADC10()
{
   ADCON1 = 0x006F;  // PWM starts conversion, 4 ch simultanés, ADC Off for
configuring
   ADCON2 = 0x0200;  // simulataneous sample 4 channels, ADC INTerrupt à chaque
EOC=100 us
   ADCHS= 0x0022;    // AN2/RB2 Ch0 Bref, AN3/RB3 Ch1 Bmes, AN4/RB4 Ch2 NC,
AN5/RB5 Ch3 NC
   ADCON3 = 0x0080;  // Tad = internal RC (4uS)
   _ADIF = 0;        // Adc int flag Off
   _ADIE = 1;        // Adc int On
   _ADON = 1;     // turn ADC ON
}
//-----------------------------------------------------------------------------
//  InitMCPWM, intializes the PWM as follows:
//  FPWM = 16 khz voir en haut
//  Independant PWMs
//  Set ADC to be triggered by PWM special trigger
//-----------------------------------------------------------------------------
void InitMCPWM()
{
   PTPER = HalfDUTY; // set the pwm period register, ne pas oublier la double
précision
   OVDCON = 0x00FF;  // allow control using OVD register (active low pour PWM pins
ie 0)
// PWMCON1 = 0x0700; // disable PWMs
```

```
   PWMCON1= 0x0770;  // enable PWM outputs (mais pas les LOW)
   PDC1=HalfDUTY; PDC2=HalfDUTY; PDC3=HalfDUTY;     // init sans rien, apres une
regul ça change
   EnableL6234=0;    // disable L298
   SEVTCMP = PTPER;  // set ADC to trigeer at ...
   PWMCON2 = 0x0000; // 1 PWM values
   PTCON = 0x8002;   // start PWM symetrique
}
//-----------------------------------------------------------------------
// The ADC interrupt reads the demand pot value.
// tt est synchrone % à cette int de ADC int =2*PWMperiod=2*62.5 us=125 us
//-----------------------------------------------------------------------
void __attribute__((interrupt, auto_psv)) _ADCInterrupt ()
{
   InfoLED=1;
   _ADIF = 0;
   k_V_f=ADCBUF0<<2; // 4.12 pu 4096=1.0=6V
   fs=ADCBUF1<<2;         // 4.12 pu 4096=1.0=20 Hz
   InfoLED=0;
}
//-------------------------------------------------------------------------------
//Main routine
int main()
{
   setup_ports();
   InitADC10();
   InitMCPWM();
   RunningLED=1;                // LED run = On
   RunningLED=0;                // LED run = Off
   while(1)
      {
      } // end of while (1)
} // end of main
```

Glossaire

DSP	Disgital Signal Processor
Dspic	
MLI	Modulation de Largeur d'Impulsion
PWM	Pulse Width Modulation
DSC	Digital Signal Controller
SPI	Serial Peripheral Interface
UART	Universal Asynchronous Receiver Transmitter
I2C	Inter Integrated Circuit
CAN	Controller Area Network
DAC	Digital Analog Converter
ADC	Analog to Digital Converter
CAN	Convertisseur Analogique-Numérique
CNA	Convertisseur Numérique-Analogique
MIPS	Million d'instructions par seconde

Bibliographie

[1] dsPIC30F3010, Microchip, caractéristiques et documents disponibles en ligne sur : http://www.microchip.com/wwwproducts/Devices.aspx?dDocName=en010335

[2] dsPIC30F3010, datasheet documents disponible en ligne sur : http://ww1.microchip.com/downloads/en/DeviceDoc/70141F.pdf

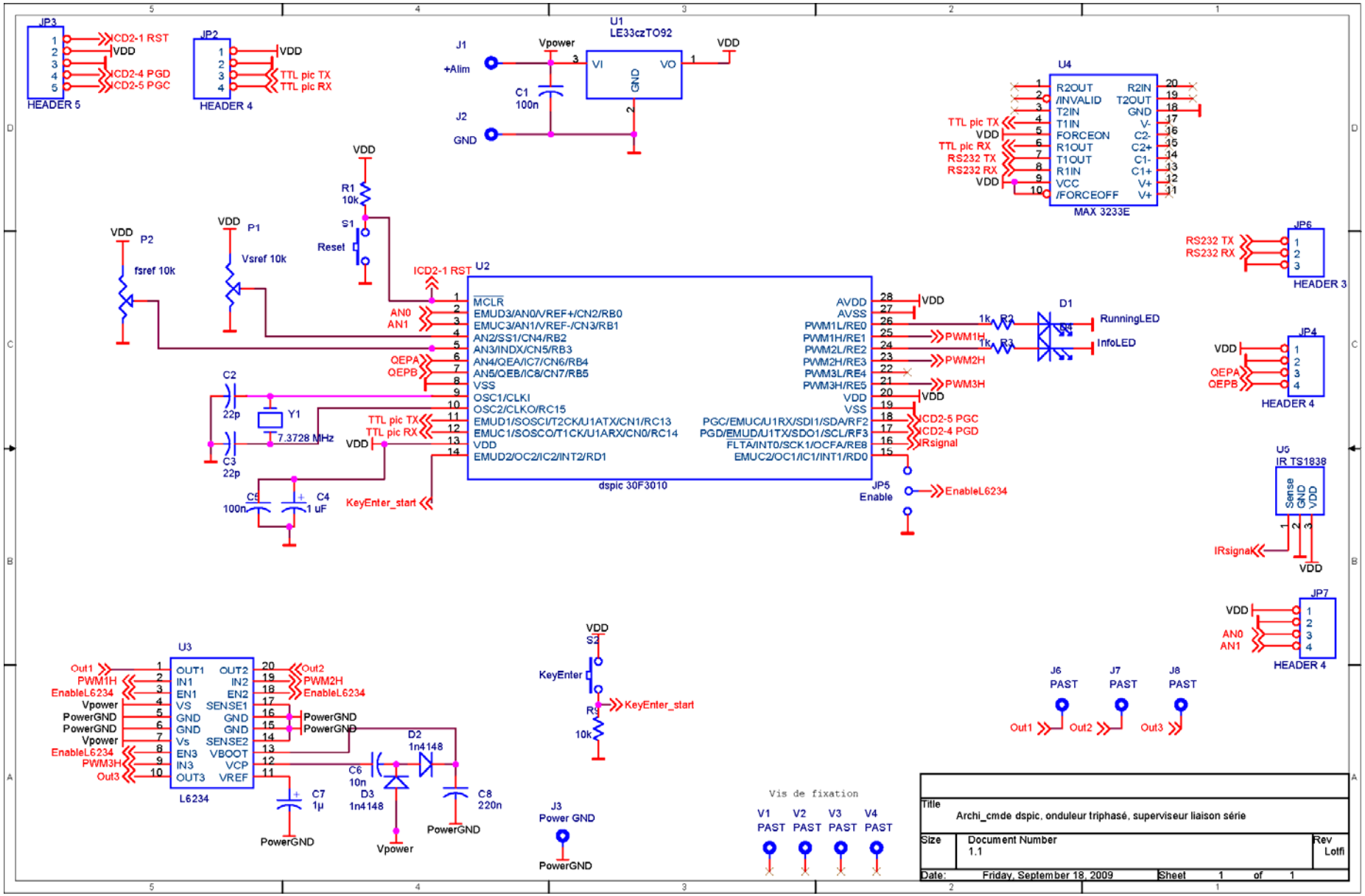
[3] dsPIC30F family reference manual, disponible en ligne sur : http://ww1.microchip.com/downloads/en/DeviceDoc/70046E.pdf

[4] dsPIC30F Family Overview, disponible en ligne sur : http://ww1.microchip.com/downloads/en/DeviceDoc/70043F.pdf

[1]

Zotero References

[1] dsPIC30F3010/3011 Data Sheet (11/12/2010) http://ww1.microchip.com/downloads/en/DeviceDoc/70141F.pdf. Microchip.


JP3
HEADER 5
ICD2-1 RST
VDD
ICD2-4 PGD
ICD2-5 PGC
JP2
HEADER 4
VDD
TTL pic TX
TTL pic RX
U1
LE33czTO92
J1
+Alim
J2
GND
Vpower
C1
100n
VI
GND
VO
VDD
U4
MAX 3233E
R2OUT
/INVALID
T2IN
T1IN
FORCEON
R1OUT
T1OUT
R1IN
VCC
/FORCEOFF
R2IN
T2OUT
GND
V-
C2-
C2+
C1-
C1+
V+
V+
RS232 TX
RS232 RX
JP6
HEADER 3
R1
10k
S1
Reset
P1
Vsref 10k
P2
fsref 10k
ICD2-1 RST
U2
MCLR
AN0
AN1
EMUD3/AN0/VREF+/CN2/RB0
EMUC3/AN1/VREF-/CN3/RB1
AN2/SS1/CN4/RB2
AN3/INDX/CN5/RB3
QEPA
QEPB
AN4/QEA/IC7/CN6/RB4
AN5/QEB/IC8/CN7/RB5
VSS
OSC1/CLKI
OSC2/CLKO/RC15
TTL pic TX
TTL pic RX
EMUD1/SOSCI/T2CK/U1ATX/CN1/RIC13
EMUC1/SOSCO/T1CK/U1ARX/CN0/RC14
VDD
EMUD2/OC2/IC2/INT2/RD1
AVDD
AVSS
PWM1L/RE0
PWM1H/RE1
PWM2L/RE2
PWM2H/RE3
PWM3L/RE4
PWM3H/RE5
VDD
VSS
PGC/EMUC/U1RX/SDI1/SDA/RF2
PGD/EMUD/U1TX/SDO1/SCL/RF3
FLTA/INT0/SCK1/OCFA/RE8
EMUC2/OC1/IC1/INT1/RD0
dspic 30F3010
D1
RunningLED
InfoLED
1k R2
1k R3
PWM1H
PWM2H
PWM3H
JP4
HEADER 4
QEPA
QEPB
C2
22p
Y1
7.3728 MHz
C3
22p
C5
100n
C4
1 uF
KeyEnter_start
JP5
Enable
EnableL6234
U5
IR TS1838
Sense
GND
VDD
IRsignal
JP7
HEADER 4
AN0
AN1
U3
L6234
Out1
PWM1H
EnableL6234
Vpower
PowerGND
PowerGND
Vpower
EnableL6234
PWM3H
Out3
OUT1
IN1
EN1
VS
GND
GND
Vs
EN3
IN3
OUT3
OUT2
IN2
EN2
SENSE1
GND
GND
SENSE2
VBOOT
VCP
VREF
Out2
PWM2H
EnableL6234
PowerGND
PowerGND
D2
1n4148
D3
1n4148
C6
10n
C7
1µ
C8
220n
PowerGND
Vpower
PowerGND
S2
KeyEnter
R5
10k
KeyEnter_start
J3
Power GND
PowerGND
Vis de fixation
V1 PAST
V2 PAST
V3 PAST
V4 PAST
J6 PAST
J7 PAST
J8 PAST
Out1
Out2
Out3
Title
Archi_cmde dspic, onduleur triphasé, superviseur liaison série
Size
Document Number
1.1
Rev
Lotfi
Date:
Friday, September 18, 2009
Sheet 1 of 1