FODITIC Université catholique de Louvain Fondation Chimay-Wartoise

Systèmes informatiques 2 - cours 2013

LINGI1113 - Marc Lobelle

FODITIC > LINGI1113 > Projet2

Projet 2: exercice sur machine nue

Informations pratiques

Voici quelques informations pratiques qui vous aideront à réaliser ce mini-projet :

- 17 boîtes contenant des cartes OLIMEX PIC MaxiWeb et leurs accessoires sont disponibles pour cette mission. Chaque carte sera allouée à un ensemble de groupes qui devra se coordonner pour utiliser la carte. Un représentant de chaque ensemble de groupes sera responsable pour la carte. Chaque groupe veillera à ce que chacun de ses membres ait l'occasion de travailler avec le PIC.
- Chaque PIC doit IMPERATIVEMENT être ramené au secrétariat INGI le jour de la remise du rapport du projet
- Le matériel fourni est fragile, vous veillerez en particulier à respecter les consignes données lors de l'emprunt.
 L'environnement de travail pour la programmation des PICS est décrit dans le document suivant : <u>PIC development in C on UNIX howto.</u>
- Certains fichiers vous sont fournis notamment un Makefile que vous devrez adapter au programme que vous voulez compiler).

Le but de ce mini-projet est de vous familariser avec la programmation en C sur une machine dite "nue", c'est-à-dire sans réel système d'exploitation. Pour cela vous programmerez un réveil matin sur une carte OLIMEX PIC MaxiWeb. En particulier, vous devrez utiliser vous mêmes les interruptions du "timer" pour déterminer l'heure. Cette carte inclut un microcontrôleur de la firme Microchip, et vos programmes y ont un accès direct à la mémoire et aux périphériques.

Le réveil matin aura les fonctionnalités suivantes :

- l'horloge affichera l'heure suivant le format hh:mm:ss. L'affichage sera donc mis à jour au moins une fois par seconde.
 les heures seront comptées de 00 à 23, l'affichage de l'heure passera donc de 23:59.59 à 00:00.00
- 3. Lors de la mise sous tension du PIC, l'horloge pourra être mise à l'heure et on pourra programmer l'heure de réveil. La sonnerie est remplacée par le clignotement d'une led chaque seconde pendant 30 secondes
- 4. En cours de fonctionnement il sera possible de changer l'heure de sonnerie sans perturber l'horloge; il sera aussi possible de remettre l'hologe à l'heure

Le rapport

Chaque (sous) groupe d'étudiant qui aura réalisé ce mini-projet (obligatoire, rappelons le) remettra dans l'outil "Travaux" du site un dossier compressé (.tar.gz ou zip) contenant:

- 1. Le code source du ou des programmes qui tournent sur le PIC
- 2. Le makefile permettant de reconstituer le programme exécutable (un conseil: ne faites pas de modification de dernière minute avant de remettre votre résultat, aussi bénigne soit-elle, sans tout retester complètement: il peut suffire d'un rien pour qu'un programme qui marchait ne marche plus)
- - 1. Le mode d'emploi de votre programme lorsqu'il est installé sur le PIC (documentation pour l'utilisateur)
 - 2. Les instructions décrivant comment compiler, installer sur le PIC et tester votre programme (documentation pour l'installateur)
 - 3. Tout ce qui est nécessaire à un programmeur qui devrait adapter votre programme (documentation pour le programmeur), c'est-à dire, par exemple:
 - Quelle est la fonction du programme (spécification)
 - Quels sont les choix structurels du programme (p. ex. il fonctionne par interruptions et pourquoi)
 - Quelle méthode avez-vous choisie pour mesurer des délais au moyen des timers du PIC et pourquoi
 - Quels sont les autres décisions d'implémentation que vous avez prises et pourquoi vous avez fait ces choix là plutôt que d'autres (même si la raison est que c'est la première idée qui vous est passée par la tête!)
 - quels sont les détails techniques du PIC qu'il faut avoir en tête pour comprendre le programme

Le rapport doit être clair pertinent et complet, ce qui n'implique pas qu'il soit long: il vaut mieux être clair en peu de mots que noircir du papier avec du blabla.

L'environnement hardware

Le système sur lequel vous allez travailler est constitué principalement d'un microcontrôleur (la grande puce carrée au dessus du LCD dans la figure 1), d'un écran LCD, de deux boutons (à gauche), d'un port Ethernet (en haut à droite), d'un port série (à droite dans l'image) et d'un petit bouton "reset" rose (juste en dessous du port ethernet). Le système fonctionne avec une alimentation de 9V (le connecteur est dans le coin inférieur gauche, près de la led rouge). La carte s'adapte à la polarité, mais il vaut mieux ne pas toucher à ces fils

Figure 1: Carte avec un microcontrôleur 18F97J60 de Microchip



Le microcontrôleur PIC18F97J60

Un microcontrôleur concentre en une seule puce toutes les fonctionnalités d'un petit ordinateur. Il comporte une unité d'exécution (le "processeur"), une mémoire RAM, FLASH et EEPROM, des périphériques tels que des timers (horloges) ainsi qu'un ou plusieurs ports pouvant être utilisés, par exemple, comme ports série ou parallèle.

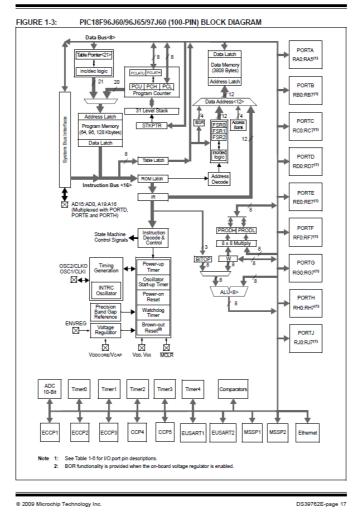
Le microcontrôleur que nous allons utiliser pour ce projet est le 18F87J60 de la firme Microchip. Un diagramme bloc de ce microcontrôleur est illustré par la figure 2. Ce microcontrôleur présente une architecture Harvard (mémoires distinctes pour programme et données). Les données sont placées dans une mémoire de type RAM, d'une capacité de 3808 octets. La mémoire de programme est constituée de mots de 16 bits, est de type FLASH (non volatile) et a une capacité de 2¹⁶ = 64 K mots (donc 128 Koctets). Ces ressources sont donc précieuses, en comparaison de celles des ordinateurs classiques. Le 18F97J60 possède encore 10 ports (A à J) et 5 temporisateurs ("timers" on y reviendra).

9/03/2013 11:33 1 sur 9

La machine dispose d'une petite pile (stack) interne utilisée pour les appels de procédure, mais elle n'a que 31 niveaux. Il ne faudra donc pas dépasser ce niveau d'imbrication d'appels, sous peine d'obtenir un comportement imprévisible.

Figure 2: Diagramme bloc du PIC18F97J60

PIC18F97J60 FAMILY

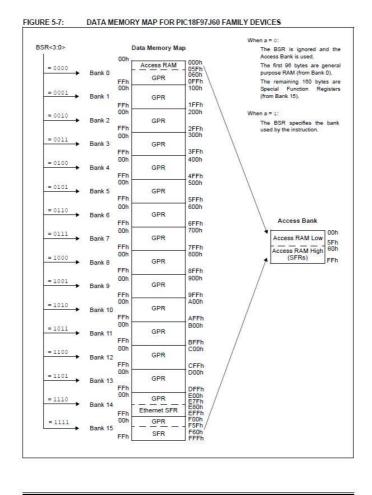


La mémoire de données

La mémoire de données apparaît comme une série de registres. Elle est divisée en 16 bancs de 256 registres. Il y a deux types de registres, les GPR (general purpose registres: de la ram) et des SFR (special function registers: des registres spécialisée internes du processeur, tels que des registres d'index, l'accumulateur (WREG) etc., ainsi que des SFR qui sont les ports d'entrées/sorties des périphériques intégrés sur la puce.

Il y a 160 registres spécialisés dans le haut du banc 15 (les plus fréquement utilisée et 128 dans le haut du banc 14. Outre les 16 vrais bancs de registre, il existe un banc virtuel qui contient les 96 GPR du bas du banc 0 et les 160 SFR du haut du banc 15. Chaque instruction contient un bit 'a'; s'il est 0, on utilise l'access bank, sinon, on utilise le vrai banc dont le numéro est dans le BSR (bank select register); on doit donc très souvent mettre à jour ce BSR, mais 'est le compilateur qui s'en charge. La figure 3 détaille l'ensemble des registres et la manière de choisir les bancs au moyen du registre spécialisé BSR et du bit a.

Figure 3: Les registres



DS39762E-page 81 © 2009 Microchip Technology Inc. Figure 3: Les registres SFR

TABLE 5-3: SPECIAL FUNCTION REGISTER MAP FOR PIC18F97J60 FAMILY DEVICES

TABLE 5-4: ETHERNET SFR MAP FOR PIC18F97J60 FAMILY DEVICES

Address	Name	Address	Name	Address	Name	Address	Name	Address	Name	Address	Name	Address	Name	Address	Name	Address	Name
FFFh	TOSU	FDFh	INDF2 ⁽¹⁾	FBFh	CCPR1H	F9Fh	IPR1	F7Fh	SPBRGH1	EFFh	(1)	EDFh	_(1)	EBFh	(1)	E9Fh	_(1)
FFEh	TOSH	FDEh	POSTINC2(1)	FBEh	CCPR1L	F9Eh	PIR1	F7Eh	BAUDCON1	EFEh	ECON2	EDEh	_(1)	EBEh	_(1)	E9Eh	_(1)
FFDh	TOSL	FDDh	POSTDEC2 ⁽¹⁾	FBDh	CCP1CON	F9Dh	PIE1	F7Dh	SPBRGH2	EFDh	ESTAT	EDDh	_(1)	EBDh	_(1)	E9Dh	_(1)
FFCh	STKPTR	FDCh	PREINC2 ⁽¹⁾	FBCh	CCPR2H	F9Ch	MEMCON ⁽⁴⁾	F7Ch	BAUDCON2	EFCh	_(1)	EDCh	_(1)	EBCh	_(1)	E9Ch	_(1)
FFBh	PCLATU	FDBh	PLUSW2 ⁽¹⁾	FBBh	CCPR2L	F9Bh	OSCTUNE	F7Bh	ERDPTH	EFBh	EIE	EDBh	_(t)	EBBh	_(1)	E9Bh	_(1)
FFAh	PCLATH	FDAh	FSR2H	FBAh	CCP2CON	F9Ah	TRISJ ⁽³⁾	F7Ah	ERDPTL	EFAh	_(1)	EDAh	_(1)	EBAh	_(1)	E9Ah	_(1)
FF9h	PCL	FD9h	FSR2L	FB9h	CCPR3H	F99h	TRISH(3)	F79h	ECCP1DEL	EF9h	(2)	ED9h	EPKTCNT	EB9h	MIRDH	E99h	EPAUSH
FF8h	TBLPTRU	FD8h	STATUS	FB8h	CCPR3L	F98h	TRISG	F78h	TMR4	EF8h	_(2)	ED8h	ERXFCON	EB8h	MIRDL	E98h	EPAUSL
FF7h	TBLPTRH	FD7h	TMR0H	FB7h	CCP3CON	F97h	TRISF	F77h	PR4	EF7h	EDMACSH	ED7h	_(1)	EB7h	MIWRH	E97h	EFLOCON
FF6h	TBLPTRL	FD6h	TMROL	FB6h	ECCP1AS	F96h	TRISE	F76h	T4CON	EF6h	EDMACSL	ED6h	_(1)	EB6h	MIWRL	E96h	(2)
FF5h	TABLAT	FD5h	TOCON	FB5h	CVRCON	F95h	TRISD	F75h	CCPR4H	EF5h	EDMADSTH	ED5h	EPMOH	EB5h	(1)	E95h	(2)
FF4h	PRODH	FD4h	(2)	FB4h	CMCON	F94h	TRISC	F74h	CCPR4L	EF4h	EDMADSTL	ED4h	EPMOL	EB4h	MIREGADR	E94h	(2)
FF3h	PRODL	FD3h	OSCCON	FB3h	TMR3H	F93h	TRISB	F73h	CCP4CON	EF3h	EDMANDH	ED3h	_(2)	EB3h	(2)	E93h	(2)
FF2h	INTCON	FD2h	ECON1	FB2h	TMR3L	F92h	TRISA	F72h	CCPR5H	EF2h	EDMANDL	ED2h	_(2)	EB2h	MICMD	E92h	(2)
FE1h	INTCON2	FD1h	WDTCON	FB1h	T3CON	F91h	LATJ ⁽³⁾	F71h	CCPR5L	EF1h	EDMASTH	ED1h	EPMCSH	EB1h	_(1)	E91h	(2)
FF0h	INTCON3	FD0h	RCON	FB0h	PSPCON	F90h	LATH(3)	F70h	CCP5CON	EF0h	EDMASTL	ED0h	EPMCSL	EBOh	_(1)	E90h	(2)
FEFh	INDF0(1)	FCFh	TMR1H	FAFh	SPBRG1	F8Fh	LATG	F6Fh	SPBRG2	EEFh	ERXWRPTH	ECFh	EPMM7	EAFh	(2)	E8Fh	(2)
FEEh	POSTINCO(1)	FCEh	TMR1L	FAEh	RCREG1	F8Eh	LATF	F6Eh	RCREG2	EEEh	ERXWRPTL	ECEh	EPMM6	EAEh	_(1)	E8Eh	(2)
FEDh	POSTDECO(1)	FCDh	TICON	FADh	TXREG1	F8Dh	LATE	F6Dh	TXREG2	EEDh	ERXRDPTH	ECDh	EPMM5	EADh	_(1)	E8Dh	(2)
FECh	PREINCO(1)	FCCh	TMR2	FACh	TXSTA1	F8Ch	LATD	F6Ch	TXSTA2	EECh	ERXRDPTL	ECCh	EPMM4	EACh	_(1)	E8Ch	(2)
FEBh	PLUSW0 ⁽¹⁾	FCBh	PR2	FABh	RCSTA1	F8Bh	LATC	F6Bh	RCSTA2	EEBh	ERXNDH	ECBh	EPMM3	EABh	MAMXFLH	E8Bh	(2)
FEAh	FSR0H	FCAh	T2CON	FAAh	(2)	F8Ah	LATB	F6Ah	ECCP3AS	EEAh	ERXNDL	ECAh	EPMM2	EAAh	MAMXFLL	E8Ah	MISTAT
FE9h	FSROL	FC9h	SSP1BUF	FA9h	(2)	F89h	LATA	F69h	ECCP3DEL	EE9h	ERXSTH	EC9h	EPMM1	EA9h	_(1)	E89h	_(1)
FE8h	WREG	FC8h	SSP1ADD	FA8h	(2)	F88h	PORTJ(3)	F68h	ECCP2AS	EE8h	ERXSTL	EC8h	EPMM0	EA8h	_(1)	E88h	_(1)
FE7h	INDF1(1)	FC7h	SSP1STAT	FA7h	EECON2 ⁽¹⁾	F87h	PORTH(3)	F67h	ECCP2DEL	EE7h	ETXNDH	EC7h	EHT7	EA7h	MAIPGH	E87h	_(1)
FE6h	POSTINC1(1)	FC6h	SSP1CON1	FA6h	EECON1	F86h	PORTG	F66h	SSP2BUF	EE6h	ETXNDL	EC6h	EHT6	EA8h	MAIPGL	E86h	_(1)
FE5h	POSTDEC1(1)	FC5h	SSP1CON2	FA5h	IPR3	F85h	PORTF	F65h	SSP2ADD	EE5h	ETXSTH	EC5h	EHT5	EA5h	_(2)	E85h	MAADR2
FE4h	PREINC1(1)	FC4h	ADRESH	FA4h	PIR3	F84h	PORTE	F64h	SSP2STAT	EE4h	ETXSTL	EC4h	EHT4	EA4h	MABBIPG	E84h	MAADR1
FE3h	PLUSW1(1)	FC3h	ADRESL	FA3h	PIE3	F83h	PORTD	F63h	SSP2CON1	EE3h	EWRPTH	EC3h	EHT3	EA3h	MACON4	E83h	MAADR4
FE2h	FSR1H	FC2h	ADCONO	FA2h	IPR2	F82h	PORTC	F62h	SSP2CON2	EE2h	EWRPTL	EC2h	EHT2	EA2h	MACON3	E82h	MAADR3
FE1h	FSR1L	FC1h	ADCON1	FA1h	PIR2	F81h	PORTB	F61h	EDATA	EE1h	_(1)	EC1h	EHT1	EA1h	(1)	E81h	MAADR6
FEOh	BSR	FCOh	ADCON2	FA0h	PIE2	F80h	PORTA	F60h	EIR	EEOh	_(1)	ECOh	EHTD	EAOh	MACON1	E80h	MAADR5

© 2009 Microchip Technology Inc.

- This is not a physical register.
 Unimplemented registers are read as '0'.
 This register is not available on 64-pin devices.
 This register is not available on 64 and 80-pin devices.

Note 1: Reserved register location; do not modify. 2: Unimplemented registers are read as '0'.

DS39762E-page 84 © 2009 Microchip Technology Inc. DS39762E-page 83

Figure 3: Contenu des premiers SFR

File Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Values on POR, BOR	Details on Page:	
TOSU	-	-	_	Top-of-Stack	Register Uppe	r Byte (TOS<	20:16>)		0 0000	63, 75	
TOSH	Top-of-Stack	Register High	Byte (TOS<	15:8>)					0000 0000	63,75	
TOSL	Top-of-Stack	Register Low	Byte (TOS<7	7:0>)					0000 0000	63,75	
STKPTR	STKFUL ⁽¹⁾	STKUNF(1)		SP4	SP3	SP2	SP1	SP0	00-0 0000	63,76	
PCLATU	-		bit 21(2)	Holding Regi	ster for PC < 20	:16>			0.0000	63,75	
PCLATH	Holding Reg	ister for PC<1	5:8>						0000 0000	63,75	
PCL	PC Low Byt	e (PC<7:0>)	.v	70					0000 0000	63,75	
TBLPTRU	-	-	bit 21	Program Mer	mory Table Poi	nter Upper By	te (TBLPTR<	20:16>)	00 0000	63, 102	
TBLPTRH	Program Me	mory Table Po	inter High By	te (TBLPTR<1	5:8>)		0000 0000	63, 102			
TBLPTRL	Program Memory Table Pointer Low Byte (TBLPTR<7:0>)									63, 102	
TABLAT	Program Me	Program Memory Table Latch Product Register High Byte									
PRODH	Product Reg	gister High Byte	e						NUCK NUCK	63, 121	
PRODL	Product Reg	gister Low Byte	9)						XXXX XXXX	63, 121	
INTCON	GIE/GIEH	PEIE/GIEL	TMR0IE	INTOIE	RBIE	TMR0IF	INTOIF	RBIF	0000 000x	63, 125	
INTCON2	RBPU	INTEDG0	INTEDG1	INTEDG2	INTEDG3	TMROIP	INT3IP	RBIP	1111 1111	63, 126	
INTCON3	INT2IP	INTIIP	INT3IE	INT2IE	INT1E	INT3IF	INT2IF	INT1F	1100 0000	63, 127	
INDF0	Uses conten	its of FSR0 to	address data	memory - val	ue of FSR0 not	changed (not	a physical re	gister)	N/A	63, 93	
POSTINCO	Uses conter	ts of FSR0 to	address data	memory - valu	ue of FSR0 pos	st-incremented	(not a physic	al register)	N/A	63, 94	
POSTDECO	Uses conter	nts of FSR0 to	address data	memory – vali	ue of FSR0 pos	st-decremente	d (not a physi	cal register)	N/A	63, 94	
PREINC0	Uses contents of FSR0 to address data memory - value of FSR0 pre-incremented (not a physical register)								N/A	63, 94	
PLUSW0		nts of FSR0 to R0 offset by W		memory – val	ue of FSR0 pre	-incremented	(not a physica	l register) –	N/A	63, 94	
FSROH					Indirect Data	Memory Addr	ess Pointer 0	High Byte	xxxx	63,93	
FSROL	Indirect Data	Memory Add	ress Pointer (Low Byte	-				HOUR HOUR	63, 94	
WREG	Working Re	gister							жин жин	63	
INDF1	Uses conter	nts of FSR1 to	address data	memory – vali	ue of FSR1 not	changed (not	a physical rej	gister)	N/A	63, 93	
POSTINC1	Uses conter	nts of FSR1 to	address data	memory - val	ue of FSR1 pos	st-incremented	(not a physic	al register)	N/A	63, 94	
POSTDEC1	Uses conter	nts of FSR1 to	address data	memory – vali	ue of FSR1 pos	st-decremente	d (not a physi	cal register)	NA	63, 94	
PREINC1	Uses conter	nts of FSR1 to	address data	memory - val	ue of FSR1 pre	-incremented	(not a physica	l register)	N/A	63, 94	
PLUSW1		nts of FSR1 to R1 offset by W		memory – val	ue of FSR1 pre	-incremented	(not a physica	l register) –	N/A	63, 94	
FSR1H	-	-	-	-	Indirect Data	Memory Addr	ess Pointer 1	High Byte	xxxx	63, 93	
FSR1L	Indirect Data	Memory Add	ress Pointer 1	Low Byte	***	- 500			MHEN HIGH	63, 93	
BSR	-	-	-	-2	Bank Select F	Register			0000	63, 93	
INDF2	Uses conter	ts of FSR2 to	address data	memory – val	ue of FSR2 not	changed (not	a physical rep	gister)	N/A	63, 93	
POSTINC2	Uses conter	nts of FSR2 to	address data	memory - vali	ue of FSR2 pos	st-incremented	(not a physic	al register)	N/A	63, 94	
POSTDEC2	Uses conter	nts of FSR2 to	cal register)	N/A	63, 94						
PREINC2	Uses conter	nts of FSR2 to	address data	memory – val	ue of FSR2 pre	-incremented	(not a physica	l register)	N/A	63, 94	
PLUSW2		nts of FSR2 to R2 offset by W		memory – val	ue of FSR2 pre	-incremented	(not a physica	l register) –	N/A	63, 94	
FSR2H	-	-	-		Indirect Data	Memory Addr	ess Pointer 2	High Byte	нин	63, 93	
FSR2L	Indirect Data	Memory Add	ress Pointer 2	Low Byte					NEEK NEEK	63, 93	

- Indirect Usan Memory Address Pointer 2. Low Byte

 _se unknown, u = unchanged, e unimplemented, read as '0', q = value depends on condition, r = reserved bit, do not modify. Shade are unimplemented, read as '0'.

 Bit 21 of the PC is only available in Serial Programming modes.

 Bit 21 of the PC is only available in Serial Programming modes.

 Reset value is '0', when Two-Speed Start-up is enabled and '1' fit disabled.

 Alternate names and definitions for these bits when the MSSP module is operating in PC™ Stave mode.

 These bits and for registers are only available in 100-pin devices; otherwise, they are unimplemented and read as '0', Reset values shrapply only to 100-pin devices.

 These bits and for registers are only available in 80-pin and 100-pin devices; in 64-pin devices, they are unimplemented and read as '0', available in this register are unimplemented and read as '0'.

 In Micropotroller mode : the bits in this register are unwritable and read as '0'.

 In Micropotroller mode : the bits in this register are unwritable and read as '0'.

- values are snown for Jul-pin devices.

 In Microcontroller mode, the bits in this register are unwritable and read as '0'.

 PLLEN is only available when either ECPLL or HSPLL Oscillator mode is selected; othe Implemented in 100-pin devices in Microcontroller mode only.

DS39762E-page 85 DS39762E-page 86 © 2009 Microchip Technology Inc

Le fichier "/usr/local/share/sdcc/include/pic16/pic18f96j60.h" contient des définitions permettant d'utiliser les noms des registres SFR comme si c'était des variables statiques de 8 bits ordinaires, comme le montrent les exemples

La mémoire de programme

Le PIC 18F97J60 dispose de 64K fois 16 bits d'espace mémoire de programme. Il s'agit d'une mémoire de type FLASH. Au bas de cette mémoire, on trouve les 3 vecteurs d'interruption:

```
0x000000: reset vector
0x00008: high priority interrupt vector
0x00018: low priority interrupt vector
```

Au haut de cette mémoire on trouve les mots de configuration du processeur: ils ne sont pris en compte qu'ue fois après la mise sous tension de la machine et sont situés aux adresses 0x1FFF8 à Ox1FFFF. Tout cela est initialisé automatiquement par le compilateur qu donne des valeurs raisonnables aux mots de configuration (voir "/usr/local/share/sdcc/include/pic18f97j60.h") et qui sait où sont les deux routines d'interution (c'est lui qui les a placées où elles sont). Voici comment elles apparaissent dans le programme:

```
void LowISR(void) interrupt (2)
    //put the code here
void HighISR(void) __interrupt (1)
    //put the code here
```

Le temporisateur

Le principe d'un temporisateur est celui d'un registre qui est incrémenté régulièrement et automatiquement (à chaque coup d'horloge par exemple), et qui, lors d'un débordement, génère une interruption. Nous vous fournissons en annexes les feuilles du manuel d'utilisation concernant le premier temporisateur du PIC (le timer0). Le PIC contient plusieurs temporisateurs. Ils ne sont pas identiques. Si vous voulez en utiliser d'autres, consultez la manuel complet du PIC

Comme le temporisateur produit une interruption, nous vous fournissons également la description d'un registre spécial : le registre intcon, décrit dans la figure 5. Ce registre sert à la gestion des interruptions. Il permet d'activer ou non toutes ou seulement certaines interruptions. Il permet également de tester quelle interruption est produite.

Question: quels sont les bits en relation avec l'utilisation du timer0?

Figure 5: Le registre intcon

REGISTER FILE SUMMARY (PIC18F97J60 FAMILY) (CONTINUED) TABLE 5-5: File Nan Bit 7 Bit 6 Bit 5 Bit 4 Bit 3 Bit 2 Bit 1 Bit 0 OR, BOR 64, 91 MROH 64, 167 64, 167 TOSBIT 64, 167 1111 1111 IDLEN SCS1 SCSO 64, 47 RXRST DMAST TXRTS RXEN IPEN BOR -ql 110 64, 58, 137 64, 171 64, 171 ner1 Register Low Byt RD16 T1RUN T1CKPS1 T1CKPS0 T10SCEN T1SYNC TMR1CS TMR10N 64, 171 64, 177 64, 177 64, 177 64, 269 TMR2 MSSP1 Re SSP1BUF SSP1ADD MSSP1 Add 64, 269 D/A BF SPISTA SMP CKE R/W UA 64, 260, 270 S 64, 272 ACKSTAT ADMSK ADRES A/D Result 64, 337 ADCAL 64, 329 64, 330 64, 331 64, 189 VCFG0 ACQT1 ADCON1 PCFG3 ADFM Capture/Compare/PWM Register 1 High Byte P1M1 P1M0 DC181 DC180 CCP1M3 CCP1M2 CCP1M1 CCP1M0 DD190 DC180 CCP1M3 CCP1M2 CCP1M1 CCP1M0 DD190 DC180 are/PWM R ister 1 Low Byt 64, 189 64, 193 64, 189 64, 189 CP2CON P2M1 P2M0 DC2B1 DC2B0 CCP2M3 CCP2M2 CCP2M1 CCP2M0 64, 193 64, 189 64, 189 64, 193 ECCP1AS 64, 205 000 0000 **64,345** CVREN CVROE CVRR CVRSS
C20UT C10UT C2INV C1INV Timer3 Register High By ox 64, 179 = unchanged, ted, read as '0

- The substance of the su appy only to 10-pm devices. These bits and/or registers are only available in 80-pin and 100-pin devices; in 64-pin devices, they are unimpler values are shown for 100-pin devices. In Microcorrotical mode, the bits in this register are unwritable and read as '0'. PLLEN is only available when either ECPL or HPPLL Oscillator mode is selected; otherwise, read as '0'. Implementation 110-pin devices in Microcontroller mode only.

riate interrupt flag bits a abling an interrupt. This REGISTER 8-1: INTCON REGISTER R/W-0 GIE/GIEH PEIE/GIEL TMROIE INTOIE RBIE TMROIF INTOIF RBIF GIE/GIEH: Global Interrupt Enable hit When IPEN = 1: 1 = Enables all high priority interrupts o = Disables all interrupts PEIE/GIEL: Peripheral Interrupt Enable bi When IPEN = 0: 1 = Enables all unmasked peripheral in 0 = Disables all peripheral interrupts When IPEN = 1: 1 = Enables all low priority peripheral into - Disables all low priority peripheral into Disables all low priority peripheral interrup TMROIE: TMROO Overflow Interrupt Enable bit 1 = Enables the TMRO overflow interrupt o Disables the TMRO overflow interrupt IMTOIE: INTO External interrupt Enable bit 1 = Enables the INTO external interrupt o Disables the INTO external interrupt bit 3 RBIE: RB Port Change Interrupt Enable bit Enables the RB port change interrupt Disables the RB port change interrupt TMRoIF: TMRo Overflow interrupt Flag bit TMRo register has overflowed (must be TMR0 register did not overflow INTOIF: INTO External Interrupt Flag bit 1 = The INTO external Interrupt occurred (must be cleared in soft 0 = The INTO external interrupt did not occur hit 1 6 = The INT O external interrupt that is account. RBIF: RB Port Change Interrupt Rag bit 1 = At least one of the RB7:RB4 pins changed state (must be cleared in software) 5 = None of the RB7:RB4 pins have changed state A mismatch condition will continue to set this bit. Reading PORTB will end the mismatch condition and allow the bit to be cleared. W - Writable bit U - Unimplemented bit, read as '0' - n = Value at POR '1' - Bit is set '0' = Bit is cleared x = Bit is unkno

INTCON Registers

Mesurer le temps avec un temporisateur

Un des usages d'un temporisateur est de mesurer le temps qui passe, que ce soit pour exécuter périodiquement des tâches ou pour attendre un certain délai. La seule différence entre ces deux cas est que dans le second cas, on ne le fait qu'une fois.

Un premier choix est de se servir d'interruptions ou d'attente active. Dans le premier cas, on programme le temporiseur pour qu'il produise une interruption lorsqu'il déborde (c'est-à-dire lorsqu'il repasse de sa valeur maximale à sa valeur minimale, par exemple de 255 à 0 pour un temporiseur à 8 bits). On peut alors faire autre chose pendant que le temporisateur temporise. On fera ce qu'il faut dans la routine d'interruption. Si on veut faire de l'attente active, on examinera régulièrement la valeur du temporisateur dans une boucle jusqu'à ce qu'il atteigne la valeur souhaitée.

Le deuxième choix, indépendant du premier, est de laisser le temporisateur tourner librement ou pas. Dans le premier cas, chaque fois qu'il déborde, il produira une interruption (parfois, il y a une opération à faire dans la routine d'interruption pour autoriser l'interruption suivante car les sources d'interruptions se désactivent souvent automatiquement lorsqu'elles produisent une interruption). Dans ce cas il faut programmer la cadence du timer pour qu'il déborde après un sous multiple de la période ou du délai souhaité. Pour modifier la cadence du temporisateur, on peut utiliser un "prescaler" qui divise la fréquence de l'horloge d'entrée du temporisateur par une puissance de 2 à choisir. Si on arrive ainsi à faire déborder le temporisateur à un sous multiple de la période souhaitée, il suffit d'incrémenter un compteur logiciel dans la routine d'interruption et de lancer le travail voulu quand il atteint la bonne valeur, mais ce n'est pas facile: avec une horloge à 10 MHz et un timer de 8 bits, on aura 78125 interruptions toutes les deux secondes, ce qui veut dire que pour effectuer quelque chose chaque seconde, il faut attendre 39062,5 périodes, donc une régularité d'une seconde n'est même pas possible et si on approxime par 39062, le temps dérivera d'un peu plus d'une seconde par jour (3600 x24/78125). Si on admet de petites variations de délai, on peut évidement alterner entre compter 39062 et 39063 interruptions. Dans ce cas, il n'y aura pas de dérive (du moins si l'horloge d'entrée du temporiseur est exactement à la fréquence annoncée). Notons cependant que, à part pour faire donner l'heure à un humain, il est plutôt rare de devoir faire quelque chose exactement à une certaine cadence. En général, l'exigence sera de faire quelque chose exactement à une certaine cadence. Et dans ce cas choisir 39062 dans l'exemple précédent est parfait (mais pas pour une horloge, bien sûr! Notons aussi que pour représenter le temps dans un ordinateur, il est plus facile de tout exprimer en ticks (le délai entre deux cha

On peut aussi ne pas laisser tourner librement le temporisateur. Pour un délai, on calculera alors le nombre de ticks à attendre, p. ex. 5423, et on calculera à quelle valeur initialiser le temporisateur, p.ex. 65536 - 5423 = 60113 si on utilise un compteur de 16 bits, pour qu'il déborde exactement après le temps demandé. On peut faire la même chose pour une horloge, mais il faut alors réinitialiser le compteur à la bonne valeur à chaque interruption. Cette "bonne valeur" sera celle calculée précédemment moins le délai entre le débordement du compteur et le moment où on le réinitialise: il faut donc soustraire de la valeur calculée précédemment la valeur du compteur au moment où on le réinitialise.

Le LCD

Le LCD qui sera utilisé pour ce projet comporte 2 lignes de 8 caractères. Le LCD se commande au moyen d'instructions qui lui sont spécifiques. Ces instructions sont par exemple l'effacement de l'écran, le déplacement du curseur ou l'écriture d'un caractère. Chacune de ces opérations prend un certain temps, qui varie selon l'opération effectuée. Considérez que vous ne savez pas le temps que prendra l'écriture sur le LCD et tenez compte de cette incertitude dans votre choix d'une technique pour mesurer l'heure. Le détail du fonctionnement de ce LCD n'est pas l'objet de ce projet. Nous vous fournissons donc des exemples d'écriture dont vous pouvez vous inspirer

Le port série

Il peut être utilisé pour communiquer avec l'ordinateur mais nous ne l'utiliserons pas: le téléchargement se fera par le réseau qui est beaucoup plus rapide.

L'environnement software

La programmation du PIC se fera dans un langage C Les étapes nécessaires pour la mise en route de votre programme sont les suivantes :

- 1. conception et écriture du programme en C,
- 2. compilation,
- 3. transfert du programme vers le microcontrôleur,
- 4. lancement du programme sur le PIC,
- 5. monitoring du fonctionnement, tests et validation du programme.

La compilation (make) produira un fichier muni de l'extension de .hex. Vous pouvez suivre ensuite les étapes décrites dans ce document pour transférer votre exécutable dans le PIC.

Environnement software du microcontrôleur

Vous allez programmer sur ``machine nue", vous aurez accès sans contrôle à tous les registres et toutes les ressources du microcontrôleur.

Indications

- Pour avoir une idée plus précise et vous faire la main avec l'environnement de développement, vous pouvez vous référer aux exemple de fichiers C pour PIC de l'archive testPIC.tgz, qui vous permettront de tester le fonctionnement et de voir à quoi ressemble le code. Ces exemples peuvent vous servir de base pour le reste du TP.
- Dans votre mini-projet, vous devrez utiliser les interruptions. Pour illuster la manière d'utiliser les interruptions sur le pic, il y a, dans testPIC, un programme appelé testint.c qui change l'état des leds rouges chaque foid qu'on appuie sut BUT2. BUT2 est connecté à l'entrée RB2/INT1 du pic. Attention: le timer0 utilise des bits de contrôle différents: INTCONbits.TMR0IE et INTCONbits.TMR0IF et il n'y a pas de bit INTEDG, qui n'a de sens que pour un signal externe, pas pour un événement interne.
- D'autres codes exemples sont disponibles sur http://www.picbook.com/downloads.html mais ils ne se compilent pas dans notre environnement. Ces exemples sont en effet prévus pour être compilés avec mcc18, nous utilisons sdcc.

Étapes du projet

Ces exercices sont principalement destinés à vous familiariser avec l'environnement de développement et le fonctionnement général du microcontrôleur.

Le ``Good morning Louvain-la-Neuve''

Pour ce premier exercice, on vous demande :

- écrivez un programme qui affiche à l'écran du LCD le message ``Good morning Louvain-la-Neuve" en veillant à ce que aucun mot ne soit coupé par une césure de ligne et fait clignoter une des leds rouges.
- compilez et transférez ce programme sur le microcontrôleur,
- · lancez le programme

Pour ce programme vous pouvez vous baser sur un des exemples de testPIC. Adaptez le Makefile pour qu'il puisse aussi gérer fichier que vous écrivez, la compilation produira un fichier .hex, à transférer vers le PIC par tftp..

Interruption INTO

Pour votre projet vous n'aurez pas besoin d'utiliser l'interruption INT1. Nous vous fournissons cependant un programme d'exemple, testint.c, qui peut vous servir de base pour comprendre comment manipuler les interruptions de timer, auxquelles vous devrez faire appel pour votre projet.

identification de la fréquence d'horloge du PIC

Dans les fichiers C exemples qui utilisent le timer, on annonce une certaine fréquence d'horloge pour la carte pic-maxi-web et certains étudiants concluent traditionellement, en cours de projet, que cette fréquence est fausse. Cela dépend de la configuration initiale de la carte qui est choisie par le serveur tftp qui télécharge votre programme. Commencez donc par définir une stratégie de test qui vous permettra de savoir quelle est la fréquence de l'horloge de base de la carte et, au besoin, un petit programme qui vous permet de faire ces tests. Evaluez la précision de votre mesure. Vous pourrez alors vous baser sur des données sûres pour concevoir votre réveil-matin.

Une manière de procéder pour votre mesure est d'utiliser le timer 1 comme référence. Le timer 1 peut, en effet, être programmé comme horloge pour compter des secondes bien plus simplement que le timer 0 (mais votre programme final devra quand même utiliser le timer 0). Voici quelques informations utiles, extraites du manuel du PIC.

Figure 6:Faire une horloge avec le timer 1

12.3 Timer1 Oscillator

An on-chip crystal oscillator circuit is incorporated between pins T1OSI (input) and T1OSO (amplifier output). It is enabled by setting the Timer1 Oscillator Enable bit, T1OSCEN (T1CON-3-). The oscillator is a low-power circuit rated for 32 kHz crystals. It will continue to run during all power-managed modes. The circuit for a typical LP oscillator is shown in Figure 12-3. Table 12-1 shows the capacitor selection for the Timer1 oscillator.

The user must provide a software time delay to ensure proper start-up of the Timer1 oscillator.

FIGURE 12-3: EXTERNAL COMPONENTS FOR THE TIMER1 OSCILLATOR

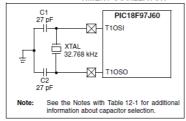


TABLE 12-1: CAPACITOR SELECTION FOR THETIMEROSCILLATOR^(2,3,4)

Oscillator Type	Freq.	C1	C2
LP	32 kHz	27 pF ⁽¹⁾	27 pF(1)

- Note 1: Microchip suggests these values as a starting point in validating the oscillator circuit.
 - 2: Higher capacitance increases the stability of the oscillator but also increases the start-up time.
 - Since each resonator/crystal has its own characteristics, the user should consult the resonator/crystal manufacturer for appropriate values of external components.
 - Capacitor values are for design guidance only.

12.3.1 USING TIMER1 AS A CLOCK SOURCE

The Timer1 oscillator is also available as a clock source in power-managed modes. By setting the Clock Select bits, SCS1:SCS0 (OSCCON<1:0>), to '01', the device switches to SEC_RUO mode; both the CPU and peripherals are clocked from the Timer1 oscillator. If the IDLEN bit (OSCCON<7>) is cleared and a SLEEP instruction is executed, the device enters SEC_IDLE mode. Additional details are available in Section 3.0 "Power-Managed Modes".

Whenever the Timer1 oscillator is providing the clock source, the Timer1 system clock status flag, T1RUN (T1CON-65>), is set. This can be used to determine the controller's current clocking mode. It can also indicate the clock source being currently used by the Fail-Safe Clock Monitor. If the Clock Monitor is enabled and the Timer1 oscillator fails while providing the clock, polling the T1RUN bit will indicate whether the clock is being provided by the Timer1 oscillator or another source.

12.4 Timer1 Interrupt

The TMR1 register pair (TMR1H:TMR1L) increments from 0000h to FFFFh and rolls over to 0000h. The Timer1 interrupt, if enabled, is generated on overflow which is latched in interrupt flag bit, TMR1IF (PIR1-0>). This interrupt can be enabled or disabled by setting or clearing the Timer1 Interrupt Enable bit, TMR1IE (PIE1-0>).

12.6 Using Timer1 as a Real-Time Clock

Adding an external LP oscillator to Timer1 (such as the one described in Section 12.3 "Timer1 Oscillator") gives users the option to include RTC functionality to their applications. This is accomplished with an inexpensive watch crystal to provide an accurate time base and several lines of application code to calculate the time. When operating in Sleep mode and using a battery or supercapacitor as a power source, it can completely eliminate the need for a separate RTC device and battery backup.

The application code routine, RTCisr, shown in Example 12-1, demonstrates a simple method to increment a counter at one-second intervals using an Interrupt Service Routine. Incrementing the TMR1 register pair to overflow triggers the interrupt and calls the routine which increments the seconds counter by one. Additional counters for minutes and hours are incremented as the previous counter overflows.

Since the register pair is 16 bits wide, counting up to overflow the register directly from a 32.768 kHz clock would take 2 seconds. To force the overflow at the required one-second intervals, it is necessary to pre-load it. The simplest method is to set the MSb of TMR1H with a BSF instruction. Note that the TMR1L register is never preloaded or altered; doing so may introduce cumulative error over many cycles.

For this method to be accurate, Timer1 must operate in Asynchronous mode and the Timer1 overflow interrupt must be enabled (PIE1<0> = 1), as shown in the routine, RTCinit. The Timer1 oscillator must also be enabled and running at all times.



```
RTCinit
                    MOVLW
                                                                               ; Preload TMR1 register pair
; for 1 second overflow
                    MOVWF
CLRF
MOVLW
MOVWF
                                         TMR1H
TMR1L
b'00001111'
T1CON
                                                                               ; Configure for external clock,
                                                                               ; Asynchronous operation, external oscillator
; Initialize timekeeping registers
                    CLRF
                                         secs
                     CLRF
                                         mins
                    MOVLW
MOVWF
                                          .12
                                         hours
PIE1, TMR1IE
                                                                              ; Enable Timerl interrupt
                    BSF
RETURN
RTCisr
                                         TMR1H, 7
PIR1, TMR1IF
secs, F
                                                                              ; Preload for 1 sec overflow
; Clear interrupt flag
; Increment seconds
; 60 seconds elapsed?
                    BCF
INCF
                    MOVLW
                                          .59
                    CPFSGT
RETURN
CLRF
INCF
                                         secs
                                                                              ; No, done
; Clear seconds
; Increment minutes
; 60 minutes elapsed?
                                        secs
mins, F
                    MOVLW
                                          .59
                    CPFSGT
RETURN
CLRF
INCF
                                        mins
                                                                              ; No, done
; clear minutes
                                         hours, F
                                                                               ; Increment hours
; 24 hours elapsed?
                    MOVLW
                                         .23
                    CPFSGT
RETURN
CLRF
RETURN
                                         hours
                                                                               ; No, done
; Reset hours
; Done
```

TABLE 12-2: REGISTERS ASSOCIATED WITH TIMER1 AS A TIMER/COUNTER

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Reset Values on page	
INTCON	GIE/GIEH	PEIE/GIEL	TMR0IE	INTOIE	RBIE	TMR0IF	INTOIF	RBIF	59	
PIR1	PSPIF	ADIF	RC1IF	TX1IF	SSP1IF	CCP1IF	TMR2IF	TMR1IF	61	
PIE1	PSPIE	ADIE	RC1IE	TX1IE	SSP1IE	CCP1IE	TMR2IE	TMR1IE	61	
IPR1	PSPIP	ADIP	RC1IP	TX1IP	SSP1IP	CCP1IP	TMR2IP	TMR1IP	61	
TMR1L	Timer1 Reg	gister Low By	rte						60	
TMR1H	Timer1 Reg	Timer1 Register High Byte								
T1CON	RD16	T1RUN	T1CKPS1	T1CKPS0	T10SCEN	TISYNC	TMR1CS	TMR10N	60	

Legend: Shaded cells are not used by the Timer1 module.

Initialisation de l'horloge

Écrivez un programme qui affiche un message demandant à l'utilisateur de régler l'heure et l'heure de réveil avec les 2 boutons. Attention, il faut contrôler la validité de l'heure rentrée (ex.: 35:26:00 n'est pas une heure valide).

Le réveil-matin

Vous êtes maintenant normalement prêt pour programmer le réveil-matin. Vous devez programmer le timer0 du microcontrôleur afin qu'il provoque une interruption toutes les secondes.

La led jaune doit clignoter avec une période de 1 seconde.

La programmation du timer se fait par le chargement de certaines valeurs dans les registres liés au timer. Ceci peut se faire en langage C. Exemple : si l'on désire mettre à 1 le bit PSA du registre TOCON, on écrit simplement, en C : TOCONDits.PSA = 1; Il est également possible de garnir directement tout le registre avec une valeur hexadécimale : TOCON = 0x88;.

Indication: pour la programmation du timer, lisez attentivement les feuilles du manuel d'utilisation en annexe. Pour la gestion des interruptions du timer, examinez le registre INTCON.

Annexes

11.0 TIMERO MODULE

The Timer0 module incorporates the following features

- Software selectable operation as a timer or counter in both 8-bit or 16-bit modes
- Readable and writable registers
- Dedicated, 8-bit, software programmable
- Selectable clock source (internal or external)
- Edge select for external clock
- · Interrupt on overflow

The TOCON register (Register 11-1) controls all aspects of the module's operation, including the prescale selection. It is both readable and writable

A simplified block diagram of the Timer0 module in 8-bit mode is shown in Figure 11-1. Figure 11-2 shows a simplified block diagram of the Timer0 module in 16-bit

REGISTER 11-1: T0CON: TIMER0 CONTROL REGISTER

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
TMR00N	T08BIT	TOCS	TOSE	PSA	T0PS2	T0PS1	T0PS0
bit 7						•	bit 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				

TMR0ON: Timer0 On/Off Control bit

1 = Enables Timer0

0 = Stops Timer0

T08BIT: Timer0 8-Bit/16-Bit Control bit

1 = Timer0 is configured as an 8-bit timer/counter 0 = Timer0 is configured as a 16-bit timer/counter

T0CS: Timer0 Clock Source Select bit

1 = Transition on TOCKI pin

0 = Internal instruction cycle clock (CLKO) T0SE: Timer0 Source Edge Select bit bit 4

bit 2-0

1 = Increment on high-to-low transition on T0CKI pin 0 = Increment on low-to-high transition on T0CKI pin

bit 3 PSA: Timer0 Prescaler Assignment bit

1 = Timer0 prescaler is NOT assigned. Timer0 clock input bypasses prescaler.
0 = Timer0 prescaler is assigned. Timer0 clock input comes from prescaler output

T0PS2:T0PS0: Timer0 Prescaler Select bits

111 = 1:256 Prescale value
110 = 1:128 Prescale value
100 = 1:128 Prescale value
101 = 1:64 Prescale value
100 = 1:32 Prescale value
011 = 1:16 Prescale value
010 = 1:8 Prescale value
001 = 1:4 Prescale value

000 = 1:2 Prescale value

11.1 Timer0 Operation

Timer0 can operate as either a timer or a counter; the mode is selected with the TOCS bit (TOCON-5>). In Timer mode (TOCS = 0), the module increments on every clock by default unless a different prescaler value every clock by default unless a different prescaler value is selected (see Section 11.3 "Prescaler"). If the TMR0 register is written to, the increment is inhibited for the following two instruction cycles. The user can work around this by writing an adjusted value to the TMR0 register.

The Counter mode is selected by setting the T0CS bit (= 1). In this mode, Timer0 increments either on every rising or falling edge of pin RA4/T0CKI. The incrementing edge is determined by the Timer0 Source Edge Select bit, T0SE (T0CON<4>); clearing this bit selects the rising edge. Restrictions on the external clock input are discussed below.

however, it must meet certain requirements to ensure that the external clock can be synchronized with the

11.2 Timer0 Reads and Writes in 16-Bit Mode

TMR0H is not the actual high byte of Timer0 in 16-bit TMR0H is not the actual high byte of Timer0 in 16-bit mode. It is actually a buffered version of the real high byte of Timer0 which is not directly readable nor writable (refer to Figure 11-2). TMR0H is updated with the contents of the high byte of Timer0 during a read of TMR0L. This provides the ability to read all 16 bits of Timer0 without having to verify that the read of the high and low byte were valid, due to a rollover between successive reads of the high and low byte. successive reads of the high and low byte.

Similarly, a write to the high byte of Timer0 must also take place through the TMROH Buffer register. The high byte is updated with the contents of TMROH when a write occurs to TMROL. This allows all 16 bits of Timer0 to be updated at once

FIGURE 11-1: TIMERO BLOCK DIAGRAM (8-BIT MODE)

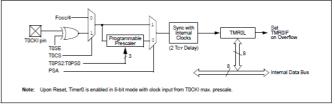
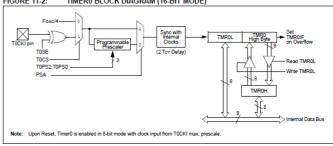


FIGURE 11-2: TIMERO BLOCK DIAGRAM (16-BIT MODE)



DS39762E-page 168 © 2009 Microchip Technology Inc DS39762E-page 167

11.3 Prescaler

An 8-bit counter is available as a prescaler for the Timer0 module. The prescaler is not directly readable or writable. Its value is set by the PSA and T0PS2:T0PS0 bits (T0CON<3:0>) which determine the prescaler assignment and prescale ratio

Clearing the PSA bit assigns the prescaler to the Timer0 module. When it is assigned, prescale values from 1:2 through 1:256 in power-of-2 increments are

When assigned to the Timer0 module, all instructions writing to the TMR0 register (e.g., CLRF TMR0, MOVWF TMRO, BSF TMRO, etc.) clear the prescaler count.

Note: Writing to TMR0 when the prescaler is assigned to Timer0 will clear the prescaler count but will not change the prescaler assignment.

SWITCHING PRESCALER 11.3.1 ASSIGNMENT

The prescaler assignment is fully under software control and can be changed *on-the-fly* during program execution.

11.4 Timer0 Interrupt

The TMR0 interrupt is generated when the TMR0 register overflows from FFh to 00h in 8-bit mode, or from FFFh to 0000h in 16-bit mode. This overflow sets the TMR0IF flag bit. The interrupt can be masked by clearing the TMR0IE bit (INTCON-5>). Before re-enabling the interrupt, the TMR0IF bit must be cleared in software by the Interrupt Service Routine.

Since Timer0 is shut down in Sleep mode, the TMR0 interrupt cannot awaken the processor from Sleep

TABLE 11-1	I: REGIS	TERS ASS	SOCIATEL	WITH III	MERO					
Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Reset Values on Page:	
TMR0L	Timer0 Reg	Timer0 Register Low Byte								
TMR0H	Timer0 Reg	Timer0 Register High Byte								
INTCON	GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INTOIF	RBIF	63	
INTCON2	RBPU	INTEDG0	INTEDG1	INTEDG2	INTEDG3	TMR0IP	INT3IP	RBIP	63	
TOCON	TMR00N	T08BIT	TOCS	TOSE	PSA	T0PS2	T0PS1	T0PS0	64	
TRISA	_	_	TRISA5	TRISA4	TRISA3	TRISA2	TRISA1	TRISA0	65	

Legend: - = unimplemented, read as '0'. Shaded cells are not used by Timer0

Différences entre mcc18 et sdcc:

- Dans mcc18 il est possible d'affecter une valeur en binaire à un registre, par exemple ADCON1=0b00000111. Avec sdcc Cela n'est possible qu'en passant par l'hexadécimal : ADCON1=0x07.
- Le schéma de gestion des interruptions n'est pas le même non plus. Avec mcc18, vous devez spécifier vous même l'adresse du vecteur d'interruptions par une directive préprocesseur #pragma (voir pour cela les exemples du site http://www.picbook.com/). Avec sacc les choses sont beaucoup plus simples puisque le compilateur s'occupe des questions d'adressag : puisqu'il y a trois types d'interruptions (reset, rapides ou lentes), une fonction de traitement correspond à chacun de ces types. Ensuite à l'intérieur de cette fonction, il faut examiner les flags pour savoir exactement quelle interruption s'est produite. Pour que le compilateur sache ce que l'on a choisi comme fonction de traitement pour les trois types d'interruptions, il suffit d'ajouter en fin de déclaration interrupt n, où n vaut 0, 1 ou 2 respectivement pour le type reset, haute priorité ou basse priorité. Les détails relatifs à ceci sont fournis dans le manuel de sdcc.
- Alors que mcc18 suppose d'inclure le fichier d'en-tête p18cxxx.h, sdcc suppose l'inclusion de pic18fregs.h.
- Alors que mccl 8permet d'effectuer des opérations sur des paires de sfr contenant les parties basse et haute d'un nombre de 16 bits (p. ex. ERDPTL et ERDPTH) comme s'il s'agissait d'un seul registre de 16 bits (ERDPT), sdcc ne gère pas ces pseudo-registres de 16 bits et exige 2 écritures séparées: une écriture dans ERDPT ne modifiera que ERDPTL.

Installation du booloader avec MPLab 6.60 :

Normalement, vous n'avez pas besoin de cette procédure d'installation. Cependant, il peut arriver que vous détruisiez le bootloader sur le PIC. Alors il faudra effectivement le ré-installer.

Créer un nouveau projet.

LINGI1113 - FODITIC

• bits de config :

• downloader le <u>bootloader</u>

 Oscillator : HS
 Watchdog Timer : Disable
 Power Up Timer : Enable
 Stack Overflow Reset : Enable o Osc. Switch Enable: Enable Brown Out Detect : Enable
 Brown Out Voltage : 4.5V
 Low Voltage Program : Disable
 CCP2 Mux : RC1

o ethernet led : enable

connecter l'Olimex ICD2-Pocket en mode debugger avec MPLab à la carte pic-maxi-web
 importer le fichier hexadécimal downloadé

programmer avec debugger

reset dans debugger

• click sur run, et cliquer sur 'no' si MPLab vous propose de refaire un build parce qu'il n'est plus à jour. Normalement, c'est la seule question posée!:).
• débranchez le cable de l'ICD2-pocket de la carte pic-maxi-web

Sébastien Tandel (version d'origine) Sébastien Barré -- sebastien dot barre at uclouvain dot be

Marc Lobelle; -- marc dot lobelle at uclouvain dot be (révision 19 mars 2010 et le 1 mars 2012)

Gestionnaire(s) du cours LINGI1113 : Marc Lobelle Administrateur FODITIC : Foditic Admin Utilise la plate-forme Claroline © 2001 - 2005 Avec le soutien du Fonds social européen