

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 IMPÉRATIVEMENT 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 : [PIC development in C on UNIX howto](#).
- 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 familiariser 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.

Énoncé

Le réveil matin aura les fonctionnalités suivantes :

1. l'horloge affichera l'heure suivant le format hh:mm:ss. L'affichage sera donc mis à jour au moins une fois par seconde.
2. 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:

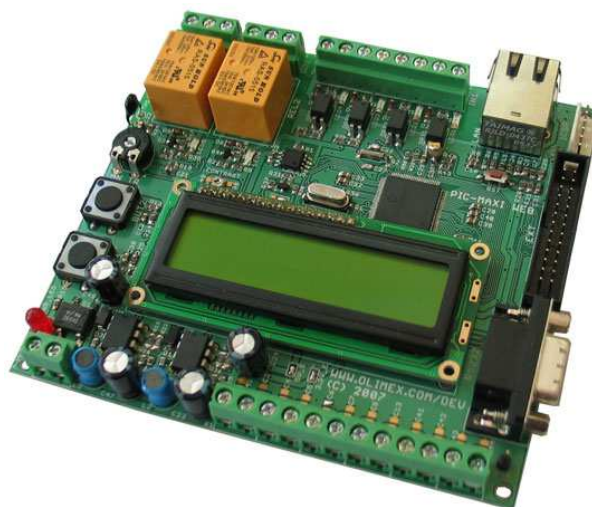
1. Le code source du ou des programmes qui tournent sur le PIC
2. Le makefile permettant de reconstituer le programme exécutable (un conseil: ne faites pas de modification de dernière minute avant de remettre votre résultat, aussi bénigne soit-elle, sans tout retester complètement: il peut suffire d'un rien pour qu'un programme qui marchait ne marche plus)
3. Un rapport contenant:
 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 $2^{16} = 64$ 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).

Figure 2: Diagramme bloc du PIC18F97J60

FIGURE 1-3: PIC18F96J60/96J65/97J60 (100-PIN) BLOCK DIAGRAM



Il y a 160 registres spécialisés dans le haut du banc 15 (les plus fréquemment 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'accès 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.

9/03/2013 11:33

Data Memory Map

BSR<3:0>	Bank	Access	Address Range
= 0000	Bank 0	Access RAM	00h - 05Fh
= 0001	Bank 1	GPR	060h - 0FFh
= 0010	Bank 2	GPR	100h - 1FFh
= 0011	Bank 3	GPR	200h - 2FFh
= 0100	Bank 4	GPR	300h - 3FFh
= 0101	Bank 5	GPR	400h - 4FFh
= 0110	Bank 6	GPR	500h - 5FFh
= 0111	Bank 7	GPR	600h - 6FFh
= 1000	Bank 8	GPR	700h - 7FFh
= 1001	Bank 9	GPR	800h - 8FFh
= 1010	Bank 10	GPR	900h - 9FFh
= 1011	Bank 11	GPR	A00h - AFFh
= 1100	Bank 12	GPR	B00h - BFFh
= 1101	Bank 13	GPR	C00h - CFFh
= 1110	Bank 14	GPR	D00h - DFFh
= 1111	Bank 15	SFR	E00h - FFFh

Access Bank

Access RAM Low	00h - 5Fh
Access RAM High (SFRs)	60h - FFh

When a = 0:
The BSR is ignored and the Access Bank is used.
The first 96 bytes are general purpose RAM (from Bank 0).
The remaining 160 bytes are Special Function Registers (from Bank 15).

When a = 1:
The BSR specifies the bank used by the instruction.

DS39762E-page 81

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

Address	Name	Address	Name	Address	Name	Address	Name
EFfH	__f1	EDfH	__f1	EBfH	__f1	EDfH	__f1
EFfE	ECON2	EDeH	__f1	EBfE	__f1	EDeH	__f1
EFfD	ESTAT	EDDh	__f1	EBfD	__f1	EDDh	__f1
EFfC	__f1	EDCh	__f1	EBfC	__f1	EDCh	__f1
EFfA	EIE	EDBh	__f1	EBfB	__f1	EDBh	__f1
EFfA	__f1	EDAh	__f1	EBfA	__f1	EDAh	__f1
EFfB	__f2	EDBh	EPKTCNT	EBfB	MIRDH	EDBh	EPAUSh
EFfB	__f2	EDBh	ERXFCON	EBfB	MIRDL	EDBh	EPAUSt
EF7h	EDMACSH	ED7h	__f1	EB7h	MIWRH	ED7h	EFLOCON
EF6h	EDMACSL	ED6h	__f1	EB6h	MIWRL	ED6h	__f2
EF5h	EDMADSTH	ED5h	EPMOH	EB5h	__f1	ED5h	__f1
EF4h	EDMADSTL	ED4h	EPMOL	EB4h	MIREGADR	ED4h	__f2
EF3h	EDMANDH	ED3h	__f2	EB3h	__f2	ED3h	__f2
EF2h	EDMANDL	ED2h	__f2	EB2h	MICMD	ED2h	__f2
EF1h	EDMASTH	ED1h	EPMCSh	EB1h	__f1	ED1h	__f2
EF0h	EDMASTL	ED0h	EPMCSL	EB0h	__f1	ED0h	__f2
EEfH	ERXWRPTh	ECfH	EPMM7	EAFh	__f2	EBfH	__f2
EEeH	ERXWRPTL	ECeH	EPMM6	EAEh	__f1	EBeH	__f2
EEDh	ERXRDPTh	ECDh	EPMM5	EADh	__f1	EDDh	__f2
EECh	ERXRDP TL	ECCh	EPMM4	EAC h	__f1	EDCh	__f2
EEBh	ERXNDH	ECBh	EPMM3	EABh	MAMXFLH	EBBh	__f2
EEA h	ERXNDL	ECAh	EPMM2	EAAh	MAMXFL L	EBAh	MISTAT
EE6h	ERXSTH	EC6h	EPMM1	EA6h	__f1	EB6h	__f1
EE8h	ERXSTL	EC8h	EPMM0	EA8h	__f1	EB8h	__f1
EE7h	ETXNDH	EC7h	EHT7	EA7h	MAIPGH	EB7h	__f1
EE6h	ETXNDL	EC6h	EHT6	EA6h	MAIPGL	EB6h	__f1
EE5h	ETXSTH	EC5h	EHT5	EAEh	__f2	EB5h	MAADR2
EE4h	ETXSTL	EC4h	EHT4	EAEh	MABBI PG	EB4h	MAADR1
EE3h	EWRPTh	EC3h	EHT3	EA3h	MACON4	EB3h	MAADR4
EE2h	EWRP TL	EC2h	EHT2	EAEh	MACON3	EB2h	MAADR3
EE1h	__f1	EC1h	EHT1	EA1h	__f1	EB1h	MAADR6
EE0h	__f1	EC0h	EHT0	EADh	MACON1	EB0h	MAADR5

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

Figure 3: Contenu des premiers SFR

TABLE 5-5: REGISTER FILE SUMMARY (PIC18F97J60 FAMILY)

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 Upper 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<7:0>)								0000 0000	63, 75
STKPTR	STKFUL ⁽¹⁾	STKUNF ⁽¹⁾	---	SP4	SP3	SP2	SP1	SP0	00-0 0000	63, 76
PCLATU	---	---	bit 21 ⁽²⁾	Holding Register for PC<20:16>					---0 0000	63, 75
PCLATH	Holding Register for PC<15:8>								0000 0000	63, 75
PCL	PC Low Byte (PC<7:0>)								0000 0000	63, 75
TBLPTRU	---	---	bit 21	Program Memory Table Pointer Upper Byte (TBLPTR<20:16>)					---0 0000	63, 102
TBLPTRH	Program Memory Table Pointer High Byte (TBLPTR<15:8>)								0000 0000	63, 102
TBLPTRL	Program Memory Table Pointer Low Byte (TBLPTR<7:0>)								0000 0000	63, 102
TABLAT	Program Memory Table Latch								0000 0000	63, 102
PRODH	Product Register High Byte								XXXXXXXX	63, 121
PRODL	Product Register Low Byte								XXXXXXXX	63, 121
INTCON	GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF	0000 0000	63, 125
INTCON2	RSPO	INTEDG0	INTEDG1	INTEDG2	INTEDG3	TMR0IP	INT3IP	RBIP	1111 1111	63, 126
INTCON3	INT2IP	INT1IP	INT3IE	INT2IE	INT1IE	INT3IF	INT2IF	INT1IF	1100 0000	63, 127
INDF0	Uses contents of FSR0 to address data memory – value of FSR0 not changed (not a physical register)								N/A	63, 93
POSTINC0	Uses contents of FSR0 to address data memory – value of FSR0 post-incremented (not a physical register)								N/A	63, 94
POSTDEC0	Uses contents of FSR0 to address data memory – value of FSR0 post-decremented (not a physical 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	Uses contents of FSR0 to address data memory – value of FSR0 pre-incremented (not a physical register) – value of FSR0 offset by W								N/A	63, 94
FSR0H	---	---	---	---	Indirect Data Memory Address Pointer 0 High Byte				---- XXXXXX	63, 93
FSR0L	Indirect Data Memory Address Pointer 0 Low Byte								XXXX XXXXXX	63, 94
WREG	Working Register								XXXXXXXX	63
INDF1	Uses contents of FSR1 to address data memory – value of FSR1 not changed (not a physical register)								N/A	63, 93
POSTINC1	Uses contents of FSR1 to address data memory – value of FSR1 post-incremented (not a physical register)								N/A	63, 94
POSTDEC1	Uses contents of FSR1 to address data memory – value of FSR1 post-decremented (not a physical register)								N/A	63, 94
PREINC1	Uses contents of FSR1 to address data memory – value of FSR1 pre-incremented (not a physical register)								N/A	63, 94
PLUSW1	Uses contents of FSR1 to address data memory – value of FSR1 pre-incremented (not a physical register) – value of FSR1 offset by W								N/A	63, 94
FSR1H	---	---	---	---	Indirect Data Memory Address Pointer 1 High Byte				---- XXXXXX	63, 93
FSR1L	Indirect Data Memory Address Pointer 1 Low Byte								XXXX XXXXXX	63, 93
BSR	---	---	---	---	Bank Select Register				---- 0000	63, 93
INDF2	Uses contents of FSR2 to address data memory – value of FSR2 not changed (not a physical register)								N/A	63, 93
POSTINC2	Uses contents of FSR2 to address data memory – value of FSR2 post-incremented (not a physical register)								N/A	63, 94
POSTDEC2	Uses contents of FSR2 to address data memory – value of FSR2 post-decremented (not a physical register)								N/A	63, 94
PREINC2	Uses contents of FSR2 to address data memory – value of FSR2 pre-incremented (not a physical register)								N/A	63, 94
PLUSW2	Uses contents of FSR2 to address data memory – value of FSR2 pre-incremented (not a physical register) – value of FSR2 offset by W								N/A	63, 94
FSR2H	---	---	---	---	Indirect Data Memory Address Pointer 2 High Byte				---- XXXXXX	63, 93
FSR2L	Indirect Data Memory Address Pointer 2 Low Byte								XXXX XXXXXX	63, 93

Legend: x = unknown, u = unchanged, - = unimplemented, read as '0', q = value depends on condition, r = reserved bit, do not modify. Shaded cells are unimplemented, read as '0'.

Note 1: Bit 7 and bit 6 are cleared by user software or by a POR.
2: Bit 21 of the PC is only available in Serial Programming modes.
3: Reset value is '0' when Two-Speed Start-up is enabled and '1' if disabled.
4: Alternate names and definitions for these bits when the MSSP module is operating in I²C™ Slave mode.
5: These bits and/or registers are only available in 100-pin devices; otherwise, they are unimplemented and read as '0'. Reset values shown apply only to 100-pin devices.
6: These bits and/or registers are only available in 80-pin and 100-pin devices; in 64-pin devices, they are unimplemented and read as '0'. Reset values are shown for 100-pin devices.
7: In Microcontroller mode, the bits in this register are unwriteable and read as '0'.
8: PLEN is only available when either ECPLL or HSPLL Oscillator mode is selected; otherwise, read as '0'.
9: Implemented in 100-pin devices in Microcontroller mode only.

TABLE 5-5: REGISTER FILE SUMMARY (PIC18F97J60 FAMILY) (CONTINUED)

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:
STATUS	---	---	---	N	OV	Z	DC	C	---0 XXXXXX	64, 91
TMR0H	Timer0 Register High Byte								0000 0000	64, 167
TMR0L	Timer0 Register Low Byte								XXXXXXXX	64, 167
T0CON	TMR0ON	T08BIT	T0CS	T0SE	PSA	T0PS2	T0PS1	T0PS0	1111 1111	64, 171
OSCCON	IDLEN	---	---	OSTS ⁽⁸⁾	---	SCS1	SCS0	---	0--- q-00	64, 47
ECON1	TXRST	RXRST	DMAST	CSUMEN	TXRTS	RXEN	---	---	0000 00--	64, 217
WDTCON	---	---	---	---	---	---	SWDTEN	---	--- --00	64, 357
RCON	IPEN	---	CM	RI	TO	PD	PCR	BOR	0-q-1 1100	64, 58, 137
TMR1H	Timer1 Register High Byte								XXXXXXXX	64, 171
TMR1L	Timer1 Register Low Byte								XXXXXXXX	64, 171
T1CON	RD16	T1RUN	T1CKPS1	T1CKPS0	T1OSCEN	T1SYNCR	TMR1CS	TMR1ON	0000 0000	64, 171
TMR2	Timer2 Register								0000 0000	64, 177
PR2	Timer2 Period Register								1111 1111	64, 177
T2CON	---	T2OUTPS3	T2OUTPS2	T2OUTPS1	T2OUTPS0	TMR2ON	T2CKPS1	T2CKPS0	0-00 0000	64, 177
SSP1BUF	MSSP1 Receive Buffer/Transmit Register								XXXXXXXX	64, 269
SSP1ADD	MSSP1 Address Register (I ² C™ Slave mode), MSSP1 Baud Rate Reload Register (I ² C Master mode)								0000 0000	64, 269
SSP1STAT	SMP	CKE	D/A	P	S	R/W	UA	BF	0000 0000	64, 260, 270
SSP1CON1	WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0	0000 0000	64, 261, 271
SSP1CON2	GCEN	ACKSTAT	ACKDT	ACKEN	RCEN	PEN	RSEN	SEN	0000 0000	64, 272
	GCEN	ACKSTAT	ADMSK3 ⁽⁹⁾	ADMSK4 ⁽⁹⁾	ADMSK3 ⁽⁹⁾	ADMSK2 ⁽⁹⁾	ADMSK1 ⁽⁹⁾	SEN		
ADRESH	A/D Result Register High Byte								XXXXXXXX	64, 337
ADRESL	A/D Result Register Low Byte								XXXXXXXX	64, 337
ADCON0	ADSCAL	---	CHS3	CHS2	CHS1	CHS0	GO/DONE	ADON	0-00 0000	64, 329
ADCON1	---	---	VCFG1	VCFG0	PCFG3	PCFG2	PCFG1	PCFG0	---0 0000	64, 330
ADCON2	ADFM	---	ACQT2	ACQT1	ADC2	ADC1	ADC0	---	0-00 0000	64, 331
CCPR1H	Capture/Compare/PWM Register 1 High Byte								XXXXXXXX	64, 189
CCPR1L	Capture/Compare/PWM Register 1 Low Byte								XXXXXXXX	64, 189
CCPR1CON	P1M1	P1M0	DC1B1	DC1B0	CCP1M3	CCP1M2	CCP1M1	CCP1M0	0000 0000	64, 193
CCPR2H	Capture/Compare/PWM Register 2 High Byte								XXXXXXXX	64, 189
CCPR2L	Capture/Compare/PWM Register 2 Low Byte								XXXXXXXX	64, 189
CCP2CON	P2M1	P2M0	DC2B1	DC2B0	CCP2M3	CCP2M2	CCP2M1	CCP2M0	0000 0000	64, 193
CCPR3H	Capture/Compare/PWM Register 3 High Byte								XXXXXXXX	64, 189
CCPR3L	Capture/Compare/PWM Register 3 Low Byte								XXXXXXXX	64, 189
CCP3CON	P3M1	P3M0	DC3B1	DC3B0	CCP3M3	CCP3M2	CCP3M1	CCP3M0	0000 0000	64, 193
ECCP1AS	ECCP1ASE	ECCP1AS2	ECCP1AS1	ECCP1AS0	PSS1AC1	PSS1AC0	PSS1BD1	PSS1BD0	0000 0000	64, 205
CVRCON	CVREN	CVRGE	CVRR	CVRSS	CVR3	CVR2	CVR1	CVR0	0000 0000	64, 345
QMRCON	C2OUT	C1OUT	C2INV	C1INV	CIS	CM2	CM1	CM0	0000 0111	64, 339
TMR3H	Timer3 Register High Byte								XXXXXXXX	64, 179
TMR3L	Timer3 Register Low Byte								XXXXXXXX	64, 179

Legend: x = unknown, u = unchanged, - = unimplemented, read as '0', q = value depends on condition, r = reserved bit, do not modify. Shaded cells are unimplemented, read as '0'.

Note 1: Bit 7 and bit 6 are cleared by user software or by a POR.
2: Bit 21 of the PC is only available in Serial Programming modes.
3: Reset value is '0' when Two-Speed Start-up is enabled and '1' if disabled.
4: Alternate names and definitions for these bits when the MSSP module is operating in I²C™ Slave mode.
5: These bits and/or registers are only available in 100-pin devices; otherwise, they are unimplemented and read as '0'. Reset values shown apply only to 100-pin devices.
6: These bits and/or registers are only available in 80-pin and 100-pin devices; in 64-pin devices, they are unimplemented and read as '0'. Reset values are shown for 100-pin devices.
7: In Microcontroller mode, the bits in this register are unwriteable and read as '0'.
8: PLEN is only available when either ECPLL or HSPLL Oscillator mode is selected; otherwise, read as '0'.
9: Implemented in 100-pin devices in Microcontroller mode only.

Le fichier "/usr/local/share/sdcc/include/pic16/pic18f9j60.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:

```
0x00000: 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'une fois après la mise sous tension de la machine et sont situés aux adresses 0x1FFF8 à 0x1FFFF. Tout cela est initialisé automatiquement par le compilateur qui donne des valeurs raisonnables aux mots de configuration (voir "/usr/local/share/sdcc/include/pic16/pic18f9j60.h") et qui sait où sont les deux routines d'interruption (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 le 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

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 INT0

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 traditionnellement, 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. Évaluez 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

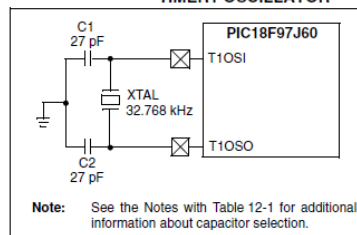


TABLE 12-1: CAPACITOR SELECTION FOR THE TIMER OSCILLATOR^(2,3,4)

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.

3: Since each resonator/crystal has its own characteristics, the user should consult the resonator/crystal manufacturer for appropriate values of external components.

4: 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_RUN 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<6>), 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 preload 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.

EXAMPLE 12-1: IMPLEMENTING A REAL-TIME CLOCK USING A TIMER1 INTERRUPT SERVICE

```

RTCinit
    MOVLW    80h           ; Preload TMR1 register pair
    MOVWF    TMR1H         ; for 1 second overflow
    CLRF     TMR1L
    MOVLW    b'00001111'   ; Configure for external clock,
    MOVWF    T1CON         ; Asynchronous operation, external oscillator
    CLRF     secs          ; Initialize timekeeping registers
    CLRF     mins
    MOVLW    .12
    MOVWF    hours
    BSF      PIE1, TMR1IE   ; Enable Timer1 interrupt
    RETURN

RTClscr
    BSF      TMR1H, 7       ; Preload for 1 sec overflow
    BCF      PIR1, TMR1IF   ; Clear interrupt flag
    INCF     secs, F        ; Increment seconds
    MOVLW    .59           ; 60 seconds elapsed?
    CPFSGT   secs
    RETURN    ; No, done
    CLRF     secs          ; Clear seconds
    INCF     mins, F        ; Increment minutes
    MOVLW    .59           ; 60 minutes elapsed?
    CPFSGT   mins
    RETURN    ; No, done
    CLRF     mins          ; Clear minutes
    INCF     hours, F       ; Increment hours
    MOVLW    .23           ; 24 hours elapsed?
    CPFSGT   hours
    RETURN    ; No, done
    CLRF     hours         ; Reset hours
    RETURN    ; 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	INT0IE	RBIE	TMR0IF	INT0IF	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 Register Low Byte								60
TMR1H	Timer1 Register High Byte								60
T1CON	RD16	T1RUN	T1CKPS1	T1CKPS0	T1OSCEN	T1SYN	TMR1CS	TMR1ON	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 T0CON, on écrit simplement, en C : `T0CONbits.PSA = 1;` Il est également possible de garnir directement tout le registre avec une valeur hexadécimale : `T0CON = 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 TIMER0 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 prescaler
- Selectable clock source (internal or external)
- Edge select for external clock
- Interrupt on overflow

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
TMR0ON	T08BIT	T0CS	T0SE	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

bit 7	TMR0ON: Timer0 On/Off Control bit 1 = Enables Timer0 0 = Stops Timer0
bit 6	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
bit 5	T0CS: Timer0 Clock Source Select bit 1 = Transition on T0CKI pin 0 = Internal instruction cycle clock (CLKO)
bit 4	T0SE: Timer0 Source Edge Select bit 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.
bit 2-0	T0PS2:T0PS0: Timer0 Prescaler Select bits 111 = 1:256 Prescale value 110 = 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

The T0CON 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 mode.

11.1 Timer0 Operation

Timer0 can operate as either a timer or a counter; the mode is selected with the T0CS bit (T0CON<5>). In Timer mode (T0CS = 0), the module increments on 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.

An external clock source can be used to drive Timer0; however, it must meet certain requirements to ensure that the external clock can be synchronized with the

internal phase clock (TOSC). There is a delay between synchronization and the onset of incrementing the timer/counter.

11.2 Timer0 Reads and Writes in 16-Bit Mode

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.

Similarly, a write to the high byte of Timer0 must also take place through the TMR0H Buffer register. The high byte is updated with the contents of TMR0H when a write occurs to TMR0L. This allows all 16 bits of Timer0 to be updated at once.

FIGURE 11-1: TIMER0 BLOCK DIAGRAM (8-BIT MODE)

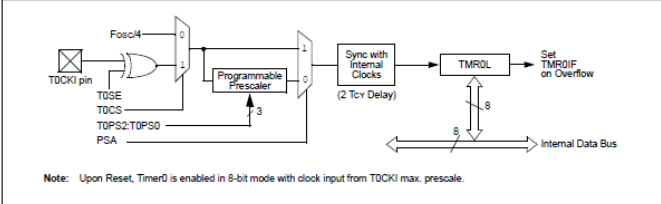
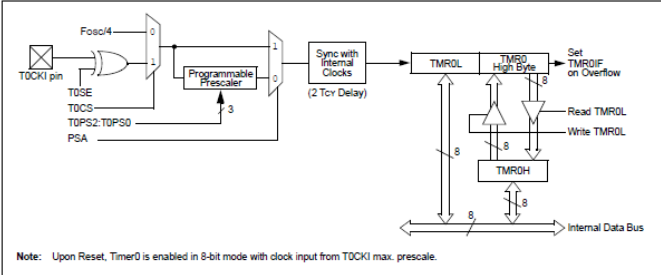


FIGURE 11-2: TIMER0 BLOCK DIAGRAM (16-BIT MODE)



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 selectable.

When assigned to the Timer0 module, all instructions writing to the TMR0 register (e.g., CLRF TMR0, MOVWF TMR0, BSF TMR0, 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.

11.3.1 SWITCHING PRESCALER 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 FFFFh 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: REGISTERS ASSOCIATED WITH TIMER0

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Reset Values on Page:
TMR0L	Timer0 Register Low Byte								64
TMR0H	Timer0 Register High Byte								64
INTCON	GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF	63
INTCON2	RBP0	INTEDG0	INTEDG1	INTEDG2	INTEDG3	TMR0IP	INT3IP	RBIP	63
T0CON	TMR0ON	T08BIT	T0CS	T0SE	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 sdcc 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 mcc18permet 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 ERDPHT) 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 bootloader 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.

- downloader le [bootloader](#)
- bits de config :
 - Oscillator : HS
 - Watchdog Timer : Disable
 - Power Up Timer : Enable
 - Stack Overflow Reset : Enable
 - Osc. Switch Enable : Enable
 - Brown Out Detect : Enable
 - Brown Out Voltage : 4.5V
 - Low Voltage Program : Disable
 - CCP2 Mux : RC1
 - 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 câble de l'ICD2-pocket de la carte pic-maxi-web

Sébastien Tandel

(version d'origine)

Sébastien Barré -- sebastien dot barre at uclouvain dot be

(révision)

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 