REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
الجمهورية الجزائرية الديمقراطية الشعبية

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR
ET DE LA RECHERCHE SCIENTIFIQUE

ECOLE SUPERIEURE EN SCIENCES APPLIQUEES
-T L E M C E N-

وزارة التعليم العالي والبحث العلمي

المدرسة العليا في العلوم التطبيقية
-تلمسان-

École Supérieure En Sciences Appliquées de Tlemcen

Département de la formation du Second Cycle

Filière: Automatique

Polycopié des Travaux Pratiques

Systèmes à Microcontrôleur

Élaboré Par
Dr MEGNAFI Hicham
Dr ABDELLAOUI Ghouti
Mr BRAHAMI Mustapha Anwar

Table des matières

Table des figures

Préface

Ce polycopié de TP regroupe un certain nombre d'applications au tour de la programmation du microcontrôleur PIC16F84A de Microchip. Il est destiné à illustrer les notions qui ont été présentées durant les séances de cours des élèves ingénieur 3eme année, filière Automatique. Ces élèves sont amenés à concevoir et à tester des systèmes à base de microcontrôleurs ou à microprocesseurs, de faire des montages expérimentaux de ces systèmes et de leur programmer en langage assembleur si nécessaire ou en langage C.

La première partie de ce polycopié est réservée à un rappel théorique sur les systèmes à microprocesseur/microcontrôleur et à la présentation des PIC en particulier le PIC16F84A alors que la deuxième partie inclut cinq Travaux pratiques qui ont pour objectifs de définir l'environnement extérieur du PIC (électronique d'interfaçage et de commande) depuis un cahier des charges et de pouvoir le programmer en langage C adapté aux PIC en utilisant comme compilateur le MikroC. Tous les exemples d'applications cités dans ces travaux pratiques ont été testés avec le logiciel de simulation ISIS Proteus V7.9.

Exemples d'application : Les exemples d'applications traités seront : clignotement des LEDs, commande un feu tricolore seul, deux feux tricolores synchronisés, gestion d'un afficheur 7 segments simple et un afficheur 7 segments BCD, mise en marche d'un moteur DC et pas à pas, commander le sens de rotation d'un moteur pas à pas et la réalisation d'un Chronomètre en

utilisant les interruptions hardware et TMR0.

Pré requis : Les pré-requis nécessaires sont :

- Connaissances de base en architectures des systèmes informatiques et en programmation en langage C (Cours Informatique 1 et 2 des classes préparatoires).
- Notions de l'électricité de base (lois des mailles et lois des nœuds, etc.) et de l'électronique (Cours d'électricité et de l'électronique des classes préparatoires).

Logiciels à disposition : Simulateur ISIS Proteus V7.9 et Compilateur MikroC.

Introduction Générale

1 Introduction

Le développement des microcontrôleurs, associés à leur faible coût en permet une large utilisation dans des domaines variés. Les microprocesseurs sont utilisés dans les systèmes automatisés pour réaliser un processus industriel :

- Matière d'œuvre : ce sur quoi s'effectue le processus,
- Energie : ce qui permet d'effectuer le processus,
- Partie opérative : ce qui permet de réaliser le processus (outils, machine),
- Partie commande : coordonne les différents processus de réalisation.

Pour les systèmes non automatisés, la partie commande est réalisée par l'homme tandis que pour les systèmes automatisés la partie commande est réalisée par les circuits électroniques. Deux solutions sont proposes pour cette dernière : La logique câblée et les Microprocesseur. Dans ce polycopie, nous nous intéressons à la deuxième solution.

1.1 Introduction sur les systèmes à Microcontrôleur

C'est un ordinateur monté dans un circuit intégré. Les avancées technologiques en matière d'intégration, ont permis d'implanter sur une puce de silicium de quelques millimètres carrés la totalité des composants qui forment la structure de base d'un ordinateur. Comme tout ordi-

nateur, on peut décomposer la structure interne d'un Microcontrôleur en trois parties (voir la figure 1) :

— Les mémoires,

— Le processeur,

— Les périphériques.

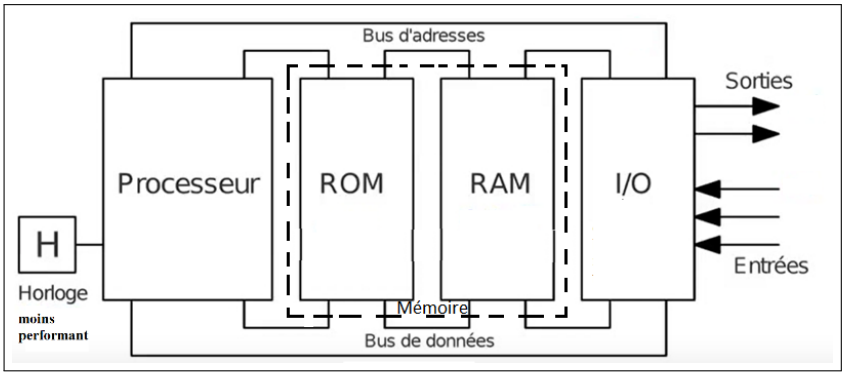


FIGURE 1: Les éléments d'un système à microprocesseur

— les mémoires sont chargées de stocker le programme qui sera exécuté ainsi que les données nécessaires et les résultats obtenus.

— le processeur est le cœur du système puisqu'il est chargé d'interpréter les instructions du programme en cours d'exécution et de réaliser les opérations qu'elles contiennent .Au sein du processeur, l'unité arithmétique et logique interprète, traduit et exécute les instructions de calcul.

— les périphériques ont pour tâche de connecter le processeur avec le monde extérieur dans les deux sens. Soit le processeur fournit des informations vers l'extérieur (périphérique de sortie), soit il en reçoit (périphérique d'entrée).

Pour l'organisation des différentes unités, il existe deux architectures :

— l'architecture Von Neumann

— l'architecture Harvard

L'architecture VON NEUMANN

L'architecture, dite architecture de **Von Neumann**, est un modèle qui utilise une structure de stockage unique pour conserver à la fois les instructions et les données requises ou générées par le calcul (voir figure 2). La séparation entre le stockage et le processeur est implicite dans ce modèle. Une même instruction permet l'accès au programme ou aux données, cependant pas simultanément. Cette architecture est maintenant principalement utilisée pour la conception des processeurs d'ordinateurs (PC, MAC).

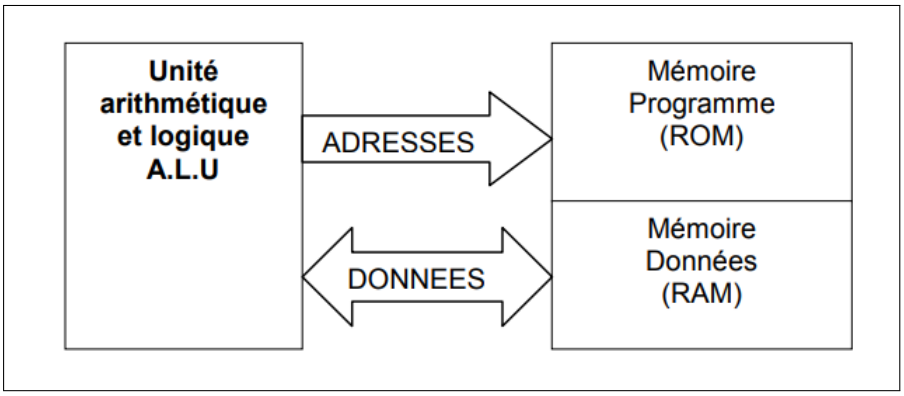


FIGURE 2: Principe de l'architecture de Von Neumann

Ce type d'architecture nécessite plusieurs cycles d'horloge pour exécuter une instruction (recherche d'instruction, décodage, lecture opérande, exécution)

L'architecture Harvard

L'architecture de type **Harvard** est une conception de microprocesseurs qui sépare physiquement la mémoire de données et la mémoire programme (voir figure 3). L'accès à chacune des deux mémoires s'effectue via deux bus distincts. Cette structure permet un accès simultané aux données et aux instructions ainsi l'exécution des programmes est plus rapide. En revanche elle nécessite des instructions différentes pour accéder à la mémoire programme et à la mémoire de données. Cette architecture très employée pour la conception des processeurs de traitement de signal (**DSP**) est de plus en plus utilisée pour les microcontrôleurs d'usage généraux.

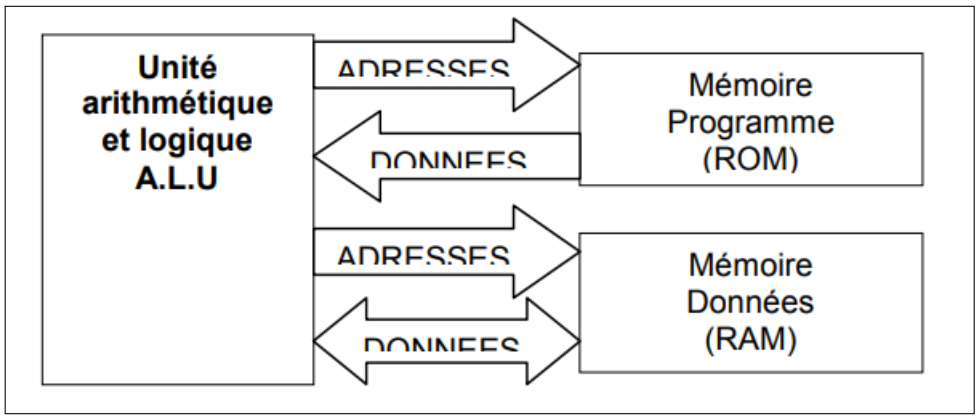


FIGURE 3: Principe de l'architecture de Harvard

Ce type d'architecture, plus complexe à fabriquer permet une exécution du programme en **PIPELINE** (recherche d'instruction, décodage, lecture opérande, exécution en un cycle d'horloge).

Comparaison CISC / RISC

Les processeurs généraux actuels se répartissent en deux grandes catégories appelées **CISC** pour Complex Instruction Set Computer et **RISC** pour Reduced Instruction Set Computer. Les processeurs de ces deux catégories se distinguent par la conception de leurs jeux d'instructions. Les processeurs **CISC** possèdent un jeu étendu d'instructions complexes. Chacune de ces instructions peut effectuer plusieurs opérations élémentaires comme charger une valeur en mémoire, faire une opération arithmétique et ranger le résultat en mémoire. Au contraire, les processeurs **RISC** possèdent un jeu d'instructions réduit où chaque instruction effectue une seule opération élémentaire. Le jeu d'instructions d'un processeur **RISC** est plus uniforme. Toutes les instructions sont codées sur la même taille et toutes s'exécute dans le même temps (un cycle d'horloge en général). L'organisation du jeu d'instructions est souvent appelé **ISA** pour Instruction Set Architecture. La répartition des principaux processeurs dans les deux catégories est la suivante.

CISC	RISC
S/360 (IBM)	MIPS
VAX (DEC)	Alpha (DEC)
68xx, 680x0 (Motorola)	PowerPC (Motorola)
x86, Pentium (Intel)	PA-RISC (Hewlett-Packard)

TABLE 1: Exemples des microcontrôleurs CISC & RISC

1.2 Rôle d'un système à microcontrôleur

Un système à microcontrôleur permet (figure 4) :

- d'acquérir des entrées logiques et analogiques représentant l'état du système technique,
- d'interpréter, la signification de ces entrées,
- de calculer, mémoriser, récupérer des variables logicielles intermédiaires,
- de gérer le temps,
- d'agir sur des sorties logiques et analogiques en fonction des entrées et des calculs réalisés de manière à modifier le fonctionnement du système technique (commande moteur, affichage d'informations,...),
- de communiquer par des liaisons séries avec d'autres systèmes techniques et/ou un ordinateur,
- ...,

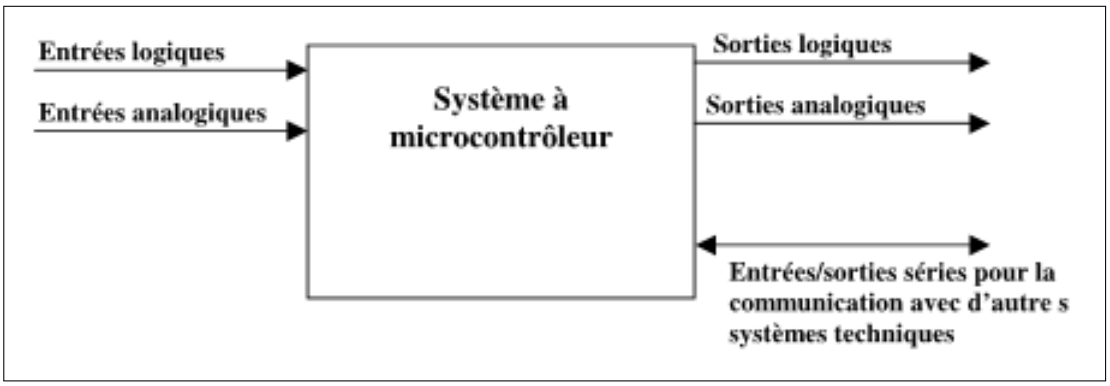


FIGURE 4: Liste des composants présentés dans la documentation n°DS30430D

2 Les PICs

Les PICs (Programmable Interface Controler) sont des microcontrôleurs fabriqués par Microship basés sur l'architecture RISC (Reduced Instructions Set Computer), ou encore composant à jeu d'instructions réduit. L'avantage est que plus on réduit le nombre d'instructions, plus leur décodage sera rapide ce qui augmente la vitesse de fonctionnement du microcontrôleur. Les PICs sont subdivisés en 3 grandes familles :

- La famille Base-Line, qui utilise des mots d'instructions de 12 bits,
- La famille Mid-Range, qui utilise des mots de 14 bits (et dont font partie les 16F8xx),
- La famille High-End, qui utilise des mots de 16 bits (les PIC 18Fxxx).

On trouve aussi des familles de DSPICs pour le traitement du signal et d'autres microcontrôleurs spécialisés ! ! Un PIC est identifié par un numéro de la forme suivant : xx(L)XXyy –zz

- xx : Famille du composant (12, 14, 16, 17, 18),
- L : Tolérance plus importante de la plage de tension,
- XX : Type de mémoire de programme,
  - C - EPROM ou EEPROM,
  - CR - PROM,
  - F - FLASH,
- yy : Identification,

— zz : Vitesse maximum du quartz.

Dans ce polycopie, on va présenter le microcontrôleur PIC16F84A, de la famille mid-range (16), utiliser une mémoire flash (F) et produit par la société MicroChip.

3 Présentation de PIC16F84A

Le PIC16F84A est un microcontrôleur produit par la société MicroChip. C'est un composant qui regroupe dans un même boîtier tous les éléments vitaux d'un système programmé : CPU, RAM, ROM, Interfaces d'entrées/sorties, etc.

Le PIC16F84A avec ces fonctionnalités entrées/sorties qui permettent de réaliser des montages avec un minimum de composants externes.

Les caractéristiques et les composants de PIC16F84A sont décrits dans la figure 5.

		PIC16F83	PIC16CR83	PIC16F84	PIC16CR84
Clock	Maximum Frequency of Operation (MHz)	10	10	10	10
Memory	Flash Program Memory	512	—	1K	—
	EEPROM Program Memory	—	—	—	—
	ROM Program Memory	—	512	—	1K
	Data Memory (bytes)	36	36	68	68
	Data EEPROM (bytes)	64	64	64	64
Peripherals	Timer Module(s)	TMR0	TMR0	TMR0	TMR0
	Interrupt Sources	4	4	4	4
	I/O Pins	13	13	13	13
Features	Voltage Range (Volts)	2.0-6.0	2.0-6.0	2.0-6.0	2.0-6.0
	Packages	18-pin DIP, SOIC	18-pin DIP, SOIC	18-pin DIP, SOIC	18-pin DIP, SOIC

FIGURE 5: Liste des composants présentés dans la datasheet du PIC16F84

Il s'agit d'un microcontrôleur 8 bits à 18 pattes. La documentation technique ou datasheet porte sur plusieurs composants, leurs principales caractéristiques sont décrites selon la figure 5.

3.1 Description et architecture externe

Le PIC16F84A est un circuit intégré de 18 broches (Figure 6) :

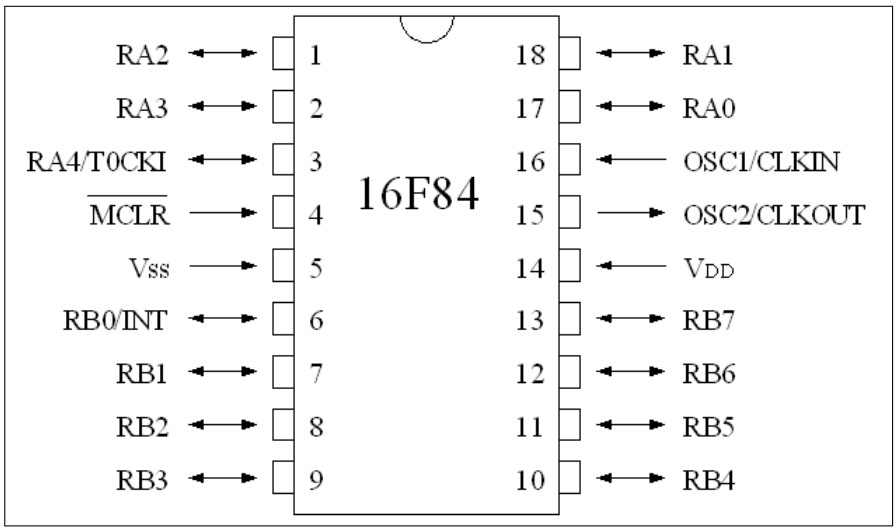


FIGURE 6: Brochage du circuit intégrer PIC16F84A

L'alimentation du circuit est assurée par les pattes VDD et VSS. Elles permettent à l'ensemble des composants électroniques du PIC de fonctionner. Pour cela on relie VSS (patte 5) à la masse (0 V) et VDD (patte 14) à la borne positive de l'alimentation qui doit délivrer une tension continue comprise entre 3 et 5.5 V.

Le microcontrôleur est un système qui exécute des instructions les unes après les autres à une vitesse qui est fixée par une horloge interne au circuit. Cette horloge doit être stabilisée de manière externe au moyen d'un cristal de quartz connecté aux pattes OSC1/CLKIN (patte 16) et OSC2/CLKOUT (patte 15).

Les broches RB0 à RB7 et RA0 à RA4 sont les lignes d'entrées/sorties numériques. Elles sont au nombre de 13 et peuvent être configurées en entrée ou en sortie. Ce sont elles qui permettent au microcontrôleur de dialoguer avec les périphériques E/S. L'ensemble des lignes RB0 à RB7 forme le port B et les lignes RA0 àRA4 forment le port A. Certaines de ces broches ont aussi d'autres fonctions (interruption, timer).

La patte 4 est appelée MCLR(Master Clear Reset). Elle permet lorsque la tension appliquée est égale à 0V de réinitialiser le microcontrôleur. C'est à dire que si un niveau bas (0 V) est appliqué sur MCLR le microcontrôleur s'arrête, place tout ses registres dans un état connu et se redirige

vers le début de la mémoire de programme pour recommencer le programme au début (adresse dans la mémoire de programme :0000). A la mise sous tension, la patte MCLR étant à zéro, le programme démarre donc à l'adresse 0000.

3.2 Description et architecture interne

L'architecture interne simplifiée du PIC16F84A est donnée par la Figure 7. Il est constitué des éléments illustres dans le tableau 2.

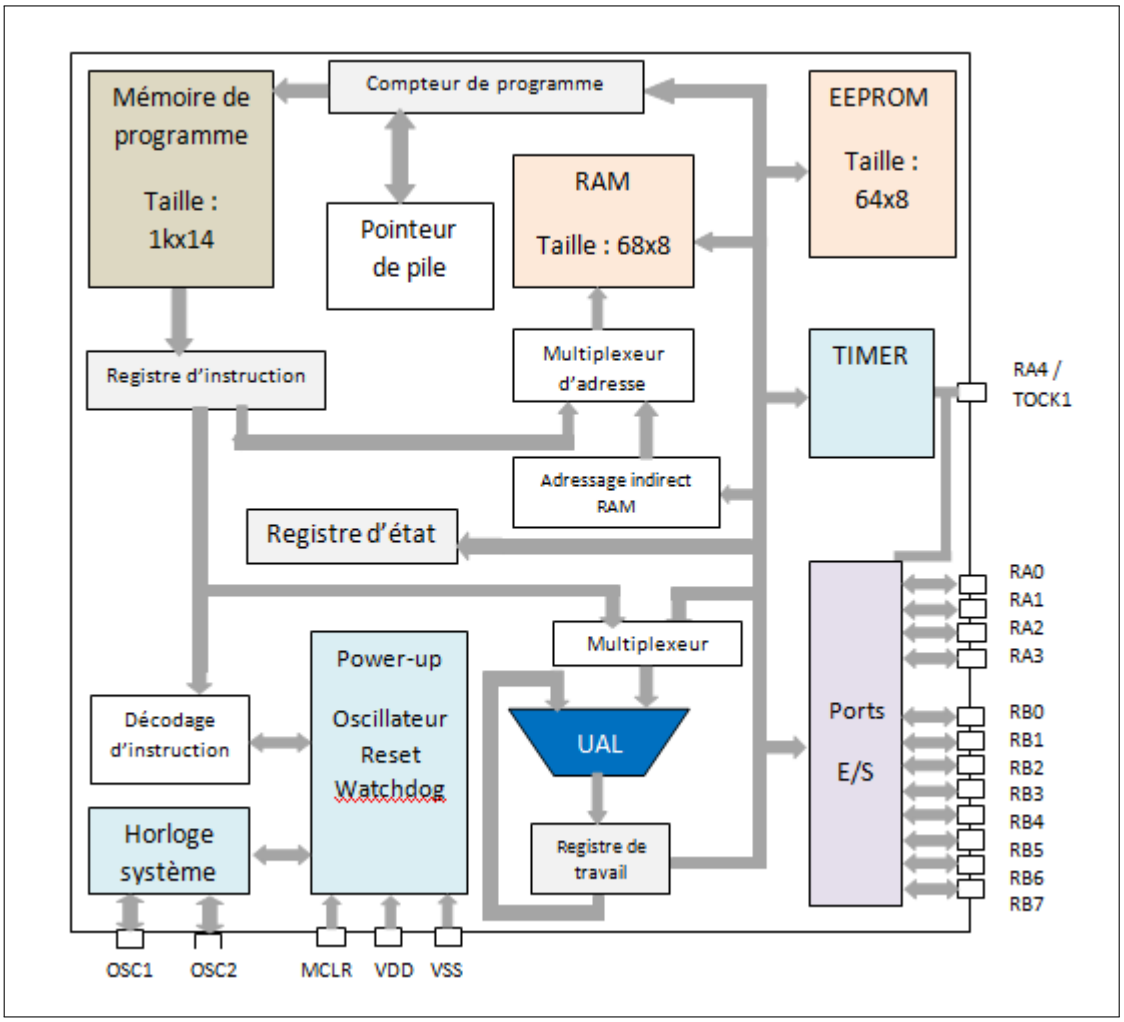


FIGURE 7: Architecture générale de PIC16F84A

Elément	Description
UAL	une unité arithmétique et logique
Bus interne	— Un bus spécifique pour les données (data bus). — Un bus spécifique pour le programme (program bus).
Mémoires	— Une mémoire flash de programme de 1k "mots" de 14 bits. — Une mémoire RAM contenant : les SFR (Special Function Registers), 68 octets de données. — Une mémoire EEPROM de 64 octets de données.
Compteurs	— Un compteur de programme (program counter) et une pile (stack). — Un compteur (timer). — Un chien de garde (watchdog).
	— Ssystème d'initialisation à la mise sous tension (power-up timer, ...) — Système de génération d'horloge à partir du quartz externe (timing génération),

TABLE 2: Description des éléments interne de PIC16F84A

4 Horloge de PIC16F84A

Le PIC16F84A fonctionne à 4 MHz avec une fréquence maximale 10 Mhz. Un cycle d'instruction utilise quatre périodes d'horloge, sauf les branchements qui nécessitent 2 cycles soit 8 périodes, avec 4 MHz, le PIC16F84A peut donc exécuter un million d'instructions simples par seconde. Le PIC16F84A possède deux modes de fonctionnement d'horloge, soit on utilise l'horloge interne ou bien une horloge externe obtenue par : un quartz (Figure 8.a)) , un circuit RC (Figure 8.b)) ou un signal externe(Figure 8.c)).

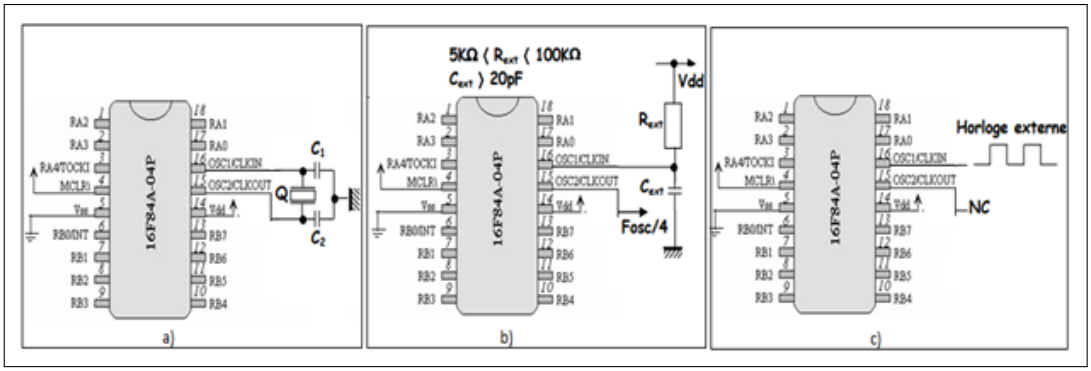


FIGURE 8: Branchement de l'horloge avec PIC16F84A obtenu par : a)quartz, b) circuit RC, c) signal externe

5 Organisation de la mémoire

Le PIC16F84A contient de la mémoire de programme et de la mémoire de données. La structure Harvard des PICs fournit un accès séparé à chacune. Ainsi, un accès aux deux est possible pendant le même cycle machine.

5.1 Mémoire de programme

L'organisation de la mémoire de programme est présentée dans la figure 9. Elle contient 1k "mots" de 14 bits, même si le compteur de programme (PC) de 13 bits peut en adresser 8k. L'adresse 0000h contient le vecteur du reset, l'adresse 0004h l'unique vecteur d'interruption

du PIC. Le PIC possède une pile de 8 niveaux pour la sauvegarde des adresses de retours suite aux appels du sous programme.

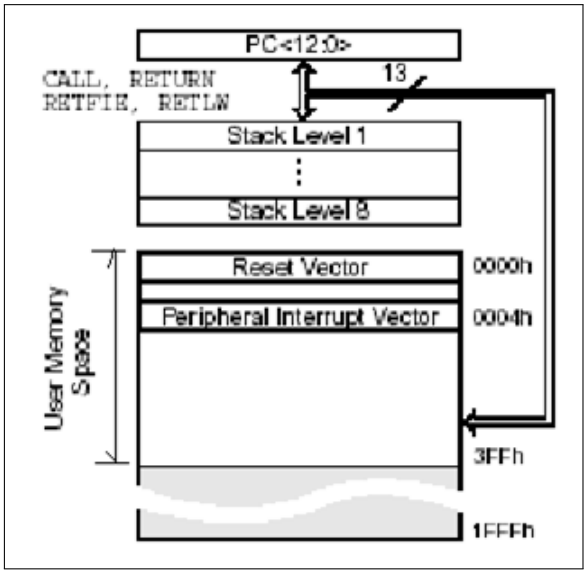


FIGURE 9: Organisation de la mémoire de programme et de la pile

5.2 Mémoire de données

La mémoire de données est décompose d'une RAM et une EEPROM. La RAM contient les SFRs qui permettent de contrôler les opérations sur le circuit (Figure 10). Tandis que La EEPROM contient des registres généraux, libres pour l'utilisateur. Les instructions orientées octets ou bits contiennent une adresse sur 7 bits pour désigner l'octet avec lequel l'instruction doit travailler. L'accès au registre TRISA d'adresse 85, par exemple, est impossible avec une adresse sur 7 bits. C'est pourquoi le constructeur a défini deux banques. Le bit RP0 du registre d'état (STATUS.5) permet de choisir entre les deux. Ainsi, une adresse sur 8 bits est composée de RP0 en poids fort et des 7 bits provenant de l'instruction à exécuter.

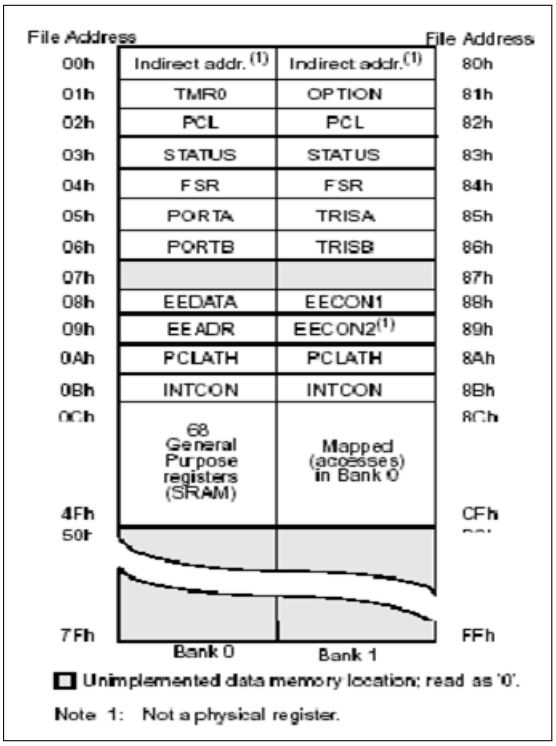


FIGURE 10: Organisation de la mémoire de données

6 Registres

6.1 Registres généraux

Ils sont accessibles soit directement soit indirectement à travers les registres FSR et INDF.

6.2 Registres spéciaux – SFRs

Ils permettent la gestion du circuit. Certains ont une fonction générale, d'autres une fonction spécifique attachée à un périphérique donné. La Figure 11 donne la fonction de chacun des bits de ces registres. Ils sont situés de l'adresse 00h à l'adresse 0Bh dans la banque 0 et de l'adresse 80h à l'adresse 8Bh dans la banque 1. Les registres 07h et 87h n'existent pas.La description de chaque registre est décrite selon le tableau 3.

Addr	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Page
Bank 0											
00h	INDF	Addressing this location uses contents of FSR to address data memory (not a physical register)								xxxx xxxx	44,201
01h	TMR0	Timer0 Module Register								xxxx xxxx	81,201
02h	PCL	Program Counter's (PC) Least Significant Byte								0000 0000	44,201
03h	STATUS	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	0001 1xxx	36,201
04h	FSR	Indirect Data Memory Address Pointer								xxxx xxxx	44,201
05h	PORTA(7)	—	—	RA5	RA4	RA3	RA2	RA1	RA0	--xx xxxx	59,201
06h	PORTB(7)	RB7	RB6	RB5	RB4	—	—	—	—	xxxx ----	69,201
07h	PORTC(7)	RC7	RC6	RC5	RC4	RC3	RC2	RC1	RC0	xxxx xxxx	76,201
08h	—	Unimplemented								—	—
09h	—	Unimplemented								—	—
0Ah	PCLATH	—	—	—	Write Buffer for upper 5 bits of Program Counter					---0 0000	44,201
0Bh	INTCON	GIE	PEIE	T0IE	INTE	RABIE	T0IF	INTF	RABIF(1)	0000 000x	38,201
0Ch	PIR1	—	ADIF(4)	RCIF(2)	TXIF(2)	SSPIF(5)	CCP1IF(3)	TMR2IF(3)	TMR1IF	-000 0000	41,201
0Dh	PIR2	OSFIF	C2IF	C1IF	EEIF	—	—	—	—	0000 ----	42,201
0Eh	TMR1L	Holding Register for the Least Significant Byte of the 16-bit TMR1 Register								xxxx xxxx	86,201
0Fh	TMR1H	Holding Register for the Most Significant Byte of the 16-bit TMR1 Register								xxxx xxxx	86,201
10h	T1CON	T1GINV	TMR1GE	T1CKPS1	T1CKPS0	T1OSCEN	$\overline{T1SYNC}$	TMR1CS	TMR1ON	0000 0000	88,201
11h	TMR2(3)	Timer2 Module Register								0000 0000	91,201
12h	T2CON(3)	—	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0	-000 0000	92,201

FIGURE 11: Description des registres

Registre	**Plage mémoire**	**description**
INDF	00h - 80h	Utilise le contenu de FSR pour l'accès indirect à la mémoire.
TMR0	01h	Registre lié au compteur.
PCL	02h - 82h	Contient les poids faibles du compteur de programmes (PC). Le registre
PCLATH	0Ah-8Ah	Contient les poids forts.
STATUS	03h - 83h	Il contient l'état de l'unité arithmétique et logique ainsi que les bits de sélection des banques.
FSR	04h - 84h	Permet l'adressage indirect.
PORTA	05h	Donne accès en lecture ou écriture au port A, 5 bits. Les sorties sont à drain ouvert. Le bit 4 peut être utilisé en entrée de comptage.

PORTB	06h	Donne accès en lecture ou écriture au port B. Les sorties sont à drain ouvert. Le bit 0 peut être utilisé en entrée d'interruption.
EEDATA	08h	Permet l'accès aux données dans la mémoire EEPROM.
EEADR	09h	Permet l'accès aux adresses de la mémoire EEPROM.
PCLATCH	0Ah - 8Ah	Donne accès en écriture aux bits de poids forts du compteur de programme.
INTCON	0Bh - 8Bh	Masque d'interruptions.
OPTION_REG	81h	Contient des bits de configuration pour divers périphériques.
TRISA	85h	Indique la direction (entrée ou sortie) du port A.
TRISB	86h	Indique la direction (entrée ou sortie) du port B.
EECON1	88h	Permet le contrôle d'accès à la mémoire EEPROM.
EECON2	89h	Permet le contrôle d'accès à la mémoire EEPROM.

Table 3: Description des registres spéciaux – SFRs

6.3 Registre STATUS

Il s'agit d'un registre spécial situé à l'adresse 0x03 (banque 0) de la mémoire des données (Data RAM). Ce registre est également accessible en banque 1 (adresse 0x83).La description de chaque bit de ce registre est décrit dans le tableau 4.

	Nom	Description
bit 7	IRP	Bit non utilisé (à laisser à 0)
bit 6	RP1	Bit non utilisé (à laisser à 0)
bit 5	RP0	Bit de sélection de la banque. — Il faut mettre ce bit à 0 pour accéder à la banque 0 (bcf STATUS , RP0), et à 1 pour accéder à la banque 1 (bsf STATUS , RP0).
bit 4	TONOT_TO	Bit "Time-out" (en lecture uniquement) — bit mis à 1 après : — une mise sous tension, — une instruction SLEEP, — une instruction CLRWDT, — un réveil (sortie du mode SLEEP) dû à une interruption, — bit mis à 0 quand la temporisation du watchdog est dépassée.
bit 3	PDNOT_PD	Bit "Power-down" (en lecture uniquement) — bit mis à 1 après : — une mise sous tension, — une instruction CLRWDT, — bit mis à 0 après une instruction SLEEP

bit 2	Z	Bit "Zero" — Cela ne concerne que les instructions qui affectent le bit Z (addwf, andlw, movf ...), — Ce bit est mis à 1 quand le résultat d'une opération est 0, — Ce bit est mis à 0 quand le résultat d'une opération est différent de 0.
bit 1	DC	Bit "Digit Carry" — Cela concerne les opérations arithmétiques (instructions : addwf, addlw, subwf et sublw) en système de numération DCB (binary coded decimal), — Ce bit est mis à 1 quand le résultat d'une opération génère une retenue, — Ce bit est mis à 0 quand le résultat d'une opération ne génère pas de retenue.
bit 0	C	Bit "Carry" — Cela concerne les opérations arithmétiques (instructions : addwf, addlw, subwf et sublw) en système de numération binaire en complément à 2, — Ce bit est mis à 1 quand le résultat d'une opération génère une retenue, — Ce bit est mis à 0 quand le résultat d'une opération ne génère pas de retenue.

Table 4: Description de registre Status

7 Ports d'entrées/Sorties

7.1 Port A

Le port A désigné par PORTA est un port bidirectionnel de 5 bits (RA0 à RA4). La configuration de direction pour chaque bit du port est déterminée avec le registre TRISA (0 est configuré en sortie, 1 est configuré en entrée). Les broches RA0 à RA3 sont des entrées/sorties compatibles TTL alors que la broche RA4 peut être utilisée soit comme entrée/sortie normale du port A, soit comme entrée horloge externe pour le Timer TMR0. Par une mesure de protection, il faux prendre on considération les remarques qui figure dans le tableau 5 :

Broche	**E/S**	**Remarques**
RA4	Sortie	il ne faut pas oublier de mettre une résistance externe vers Vdd parce qu'elle est une sortie à drain ouvert (dans le cas ou la RA4 est configuré comme un port E/S de port A)
RA4,.., RA0	Entrée	Chaque broche peut accepter un courant de 25 mA au maximum, mais tout le port ne peut pas accepter un courant total supérieur à 80 mA.
RA4,.., RA0	Sortie	Chaque broche peut fournir un courant de 20 mA au maximum, mais tout le port A ne peut pas débiter un courant total supérieur à 50 mA.

Table 5: Remarques pour le bon fonctionnement du port A

7.2 Port B

Le port B désigné par PORTB est un port bidirectionnel de 8 bits (RB0 à RB7). La configuration de direction se fait à l'aide du registre TRISB (identique à celle du PORTA : voir TRISA). Toutes les broches sont compatibles TTL. En entrée, la broche RB0 appelée aussi INT peut déclencher l'interruption externe INT et une quelconque des broches RB4 à RB7 peut déclencher l'interruption RBI. Comme nous avons dit dans le port A, voir le tableau 6 pour une meilleur protection de port B.

Broche	**Configuration**	**Remarques**
RB7,.., RB0	Entrée	Chaque broche peut accepter un courant de 25 mA au maximum, mais tout le port ne peut pas accepter un courant total supérieur à 150 mA.
RB7,.., RB0	Sortie	Chaque broche peut fournir un courant de 20 mA au maximum, mais tout le port A ne peut pas débiter un courant total supérieur à 100 mA.

TABLE 6: Remarques pour le bon fonctionnement du port A

8 Travaux pratiques

- TP N°I : **Prise en main de l'environnement Proteus ISIS et du compilateur MikroC PRO pour PIC16F84A.**
- TP N°II : **Manipulation des entrées sorties d'un microcontrôleur PIC16F84A (Afficheur 7-segments).**
- TP N°III : **Manipulation des entrées sorties d'un microcontrôleur PIC16F84A (Commande d'un moteur).**
- TP N°IV : **Gestion des interruptions via le PIC16F84A.**
- TP N°V : **Utilisation de l'interruption TMR0 via le PIC16F84A.**

TP N° 1

Prise en main de l'environnement de développement pour le PIC1684A

1.1 Objectifs

- Se familiariser avec l'environnement du logiciel Proteus ISIS.
- Utiliser le compilateur MikroC PRO pour programmer un PIC.
- Comprendre l'architecture matérielle et logicielle du microcontrôleur PIC16F84A.

1.2 Introduction

La simulation des systèmes embarqués exige l'utilisation de deux logiciels, le premier est un logiciel dédie à la simulation des circuits électronique et le deuxième pour la programmation des PICs, dans ce contexte nous allons utiliser durons ces Travaux Pratiques les softwares : Proteus ISIS et MikroC PRO.

1.2.1 Proteus ISIS

ISIS Proteus est un logiciel de simulation électronique édité par la société « Lab Center Electronics ». Il intègre des simulateurs analogique, logique et mixte pour tout type de circuit électronique. Grâce à des modules additionnels, ISIS est également capable de simuler le comportement d'un microcontrôleur (PIC, Atmel, Intel 8051, ARM, Motorola HC11...) et son interaction avec les composants qui l'entourent, c'est cette dernière fonctionnalité d'ISIS qui nous intéresse dans ce TP. La figure 1.1 illustre la fenêtre principale du logiciel ISIS.

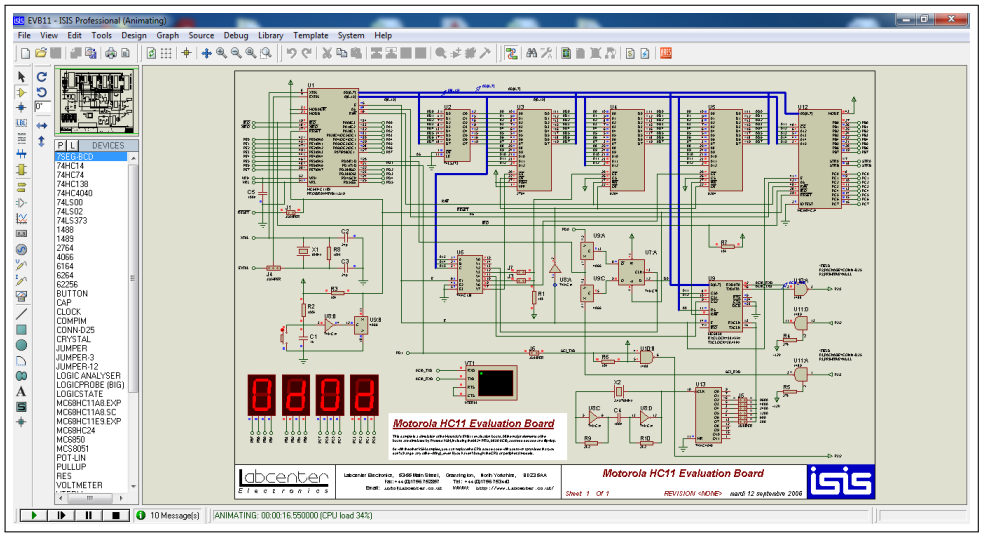


FIGURE 1.1: Fenêtre Proteus ISIS

1.2.2 MikroC PRO

Le « mikroC PRO » est un compilateur pour PIC conçu par la société «Mikroelektronika». Il comporte plusieurs outils intégrés (mode simulateur, terminal de communication, gestionnaire 7 segments,...). Il aune capacité à pouvoir gérer la plupart des périphériques rencontrés dans l'industrie (BusI2C, 1Wire, SPI, RS4805, Bus CAN, cartes compact Flash, signaux PWM, afficheurs LCD et 7 segments...), de ce fait il est un outil de développement incontournable et puissant. Il contient un large ensemble de bibliothèques de matériel, de composant et une documentation complète. La figure 1.2 illustre la fenêtre principale du logiciel MikroC PRO.

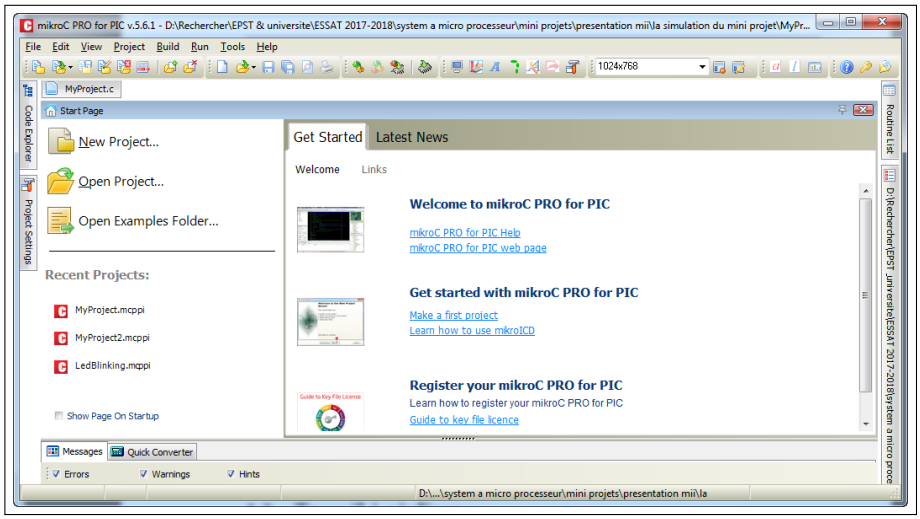


FIGURE 1.2: Fenêtre Mikro C PRO

1.3 Travail à réaliser

Dans ce qui suit, nous allons réaliser quelques applications basiques à base du microcontrôleur PIC16F84A, ces exemples vont nous permettre de bien se familiariser avec l'environnement de simulation ISIS et le compilateur mikroC PRO.

1.3.1 Application N°1 : Allumer une LED

Le but de cette première application est d'allumer une LED de couleur rouge connectée à la broche RB0 du port B du PIC16F84A, selon les étapes suivantes :

1. Lancer ISIS et le Compilateur mikroC PRO.
2. Réaliser le circuit de test (voir figure 1.3) sous ISIS :
3. Ouvrir un nouveau projet sous mikroC PRO, écrire le programme ci-dessous :
4. Compiler votre programme, et générer le fichier .hex correspondant à votre programme.
5. Dans ISIS, on double clique sur le microcontrôleur et on injecte le fichier .hex généré dans l'étape précédente.

6. Par la suite lancer la simulation sous ISIS (le bouton play du panneau de contrôle de l'animation), la LED sera normalement allumé.

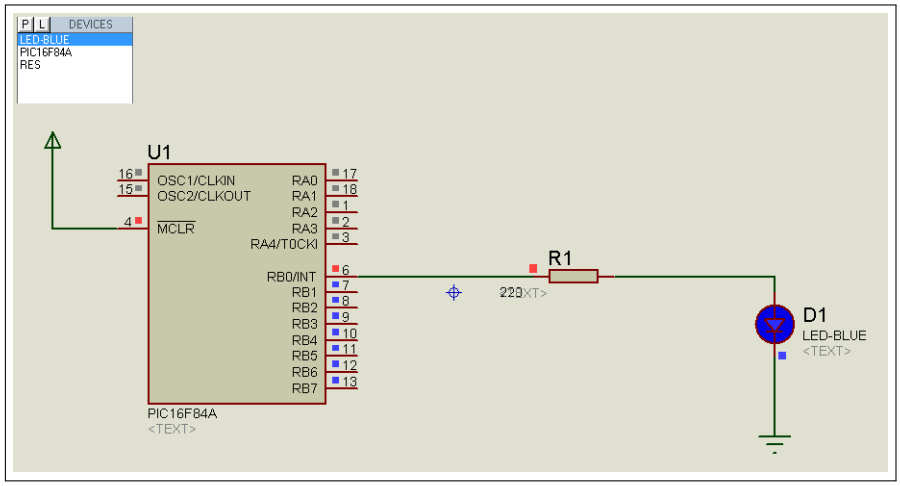


FIGURE 1.3: Schéma électronique « commande une LED par le PIC16F84A »

```
Void main() {
trisb= 0b11111110 ; // configuration du pinRB0 en sortie (1 : Entrée, 0 : Sortie)
while(1)                    // boucle infinie
{portb = 0b00000001 ;} // la led connectée au pin RB0 sera allumée
}
```

Dans les applications qui suivent, nous allons refaire les mêmes étapes de 1 à 6, afin de réussir la simulation.

1.3.2 Application N°2 : Clignotement d'une LED

Dans cette application, nous allons transformer notre programme pour faire clignoter la LED. La LED s'allume pendant une seconde et s'éteint pendant la seconde suivante. Le même circuit de la première application sera utilisé. Dans ce programme, nous utiliserons la fonction delay_ms() permettant d'ajouter un délai en millisecondes.

Le programme correspondant à cette application est le suivant :

```
void main() {
trisb= 0b11111110; // configuration du pinRB0 en sortie (1 : Entrée, 0 : Sortie)
while(1)                          // boucle infinie
{ portb = 0b00000001;             // allumer la led connectée au pin RB0
delay_ms(1000);                   // pause d'une seconde
portb = 0b00000000;               // éteindre la led connectée au pin RB0
delay_ms(1000);                   // pause d'une seconde
}}
```

1.3.3 Application N°3 : Clignotement de plusieurs LED

Le but de cette application est de faire clignoter d'une manière alternée 4 LED de couleur rouge connectées au port B du PIC16F84A correspondant aux broches allant de RB0 jusqu'à RB3. Les LED s'allument pendant une demi-seconde et s'éteignent pendant la demi-seconde suivante.

1. Réaliser le circuit de test (figure 1.4) sous ISIS.
2. Ecrire le programme illustré par la figure 1.5 sous mikroC PRO.

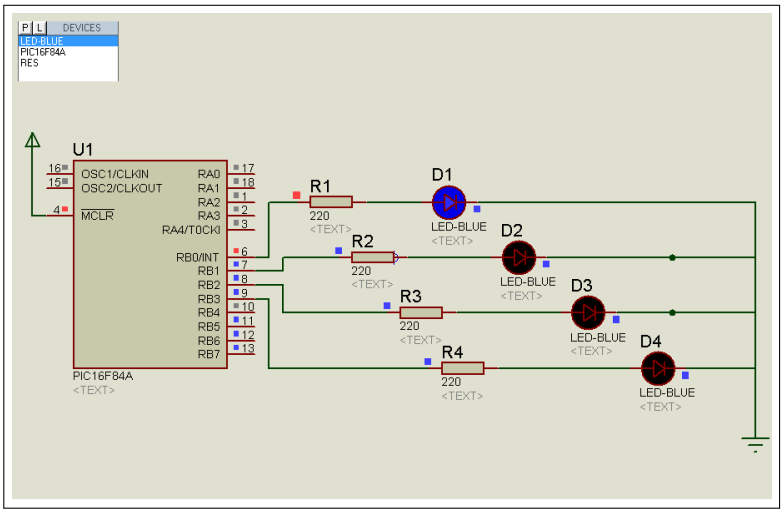


FIGURE 1.4: Schéma électronique « Clignotement de plusieurs LED »

```
void main() {
trisb= 0x10;          // configuration des pinsRB0 jusqu'à RB3 en sortie
while(1)                    // boucle infinie
{ portb = 0b00000000;
delay_ms(500); portb = 0b00000001;
delay_ms(500); portb = 0b00000010;
delay_ms(500); portb = 0b00000100;
delay_ms(500); portb = 0b00001000;
delay_ms(500); }}
```

FIGURE 1.5: Code source pour faire clignoter d'une manière alternée 4 LEDs

Dans l'exemple précédent on a allumé les LED en agissant sur le port en entier, cependant, nous pouvons accéder individuellement à chaque bit en utilisant les identifiantsF0, ..., F7. Par exemple pour allumer la première LED, on écrit Portb.F0 = 1.

1. Ecrire un autre programme illustré par la figure 1.6 qui permet de faire clignoter les LED paires pendant une seconde et les LED impaires pendant la seconde suivante.

1.3.4 Application N°4 : Un seul feu tricolore

Le but de cette application est de simuler la commande d'un seul feu tricolore à l'aide du PIC16F84A. Le fonctionnement du feu tricolore est détaillé comme suit :

```
void main() {
trisb= 0x10 ; // configuration des pinsRB0 jusqu'à RB3 en sortie
while(1) // boucle infinie
{ portb = 0b00000000 ;
delay_ms(500) ; portb.f0 = 1 ;
delay_ms(500) ; portb.f0 =0 ; portb.f1 = 1 ;
delay_ms(500) ; portb.f1 = 0 ; portb.f2 = 1 ;
delay_ms(500) ; portb.f2 = 0 ; portb.f3 = 1 ;
delay_ms(500) ; }}
```

FIGURE 1.6: Code source pour faire clignoter les LEDs paires et impaires d'une manière alternée

- Allumer le feu vert pendant 30 secondes.
- Allumer le feu orange pendant 04 secondes.
- Allumer le feu rouge pendant 20 secondes.
- Reprendre le cycle du début.

1.3.5 Application N°5 : Deux feux tricolores synchronisés

Dans cette application, vous allez simuler un cas réel des feux de croisement composés de deux feux tricolores synchronisés (voir figure 1.7). Les deux feux permettront l'régulation de la circulation d'un carrefour à deux voies (route principale et route secondaire).

FIGURE 1.7: Deux feux tricolores synchronisés.

Le fonctionnement des deux feux tricolores est détaillé comme suit :

- Allumer le feu vert pendant 40 secondes sur la route principale et le rouge sur la route secondaire.
- Allumer le feu orange pendant 05 secondes sur la route principale et toujours le rouge sur la route secondaire.
- Allumer le feu rouge pendant 25 secondes sur la route principale et le feu vert sur la route secondaire.
- Allumer le feu orange pendant 04 secondes sur la route secondaire et toujours le rouge sur la route principale.
- Reprendre le cycle du début

TP N° 2

Manipulation des entrées sorties d'un microcontrôleur PIC16F84A (Afficheur 7-segments)

2.1 Objectifs

- Maitriser les entrées sorties du 16F84A.
- Comprendre l'utilisation des afficheurs 7-segments (simple/BCD).
- Maitriser la gestion de temps.

2.2 Introduction

Les afficheurs 7 segments sont un type d'afficheur très présent sur les calculatrices et les montres à affichage numérique : les caractères (des chiffres, bien que quelques lettres soient utilisées pour l'affichage hexadécimal) s'écrivent en allumant ou en éteignant des segments, au nombre de sept. Chaque segment est désigné par une lettre a, b, c, d, e, f, g. La figure 2.1 montre la disposition de ces segments ainsi que les déférentes combinaisons utiles pour repré-

senter les caractères.

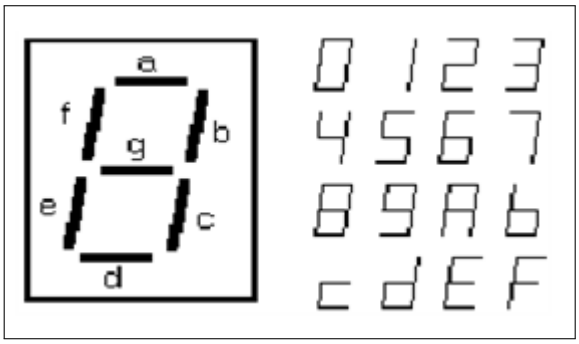


FIGURE 2.1: Agencement d'un afficheur 7 segments

Lorsque l'on veut optimiser le nombre des pins E/S de PIC utilisé pour commande l'afficheur 7 segments, on fait appel à un décodeur BCD. Ou bien directement un afficheur 7 segment –BCD (Figure 2.2).

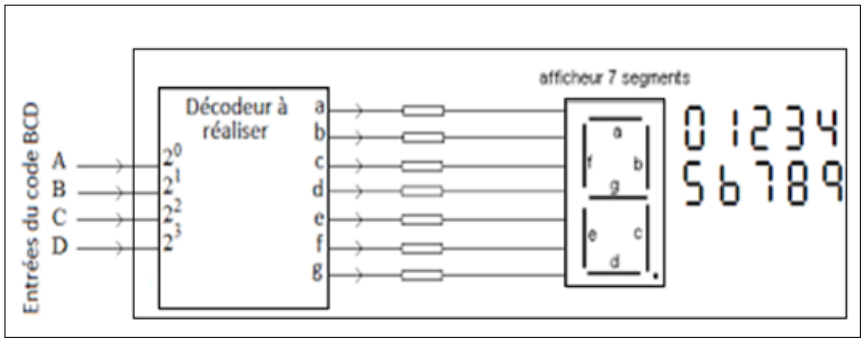


FIGURE 2.2: Afficheur 7segements - BCD

2.3 Travail à réaliser

Dans ce qui suit, nous allons réaliser quelques applications basiques d'utilisation des afficheurs 7-segments, commençant par l'afficheur le plus simple qui permet d'afficher un seul nombre jusqu'a des afficheurs plus complexe (deux nombre), pour cela nous allons utiliser l'environnement de conception ISIS Proteus et le compilateur mikroC PRO.

2.3.1 Application N°1 : Afficheur 7-segments simple

Le but de cette première application est de faire un compteur de 0 a 9à l'aide d'un afficheur 7-segments simple (7 entrées) qui seront connectées aux broches RB0..RB6 du port B du PIC16F84A. Selon la figure 2.3.

1. Ecrire le programme qui permet de gérer ce montage.

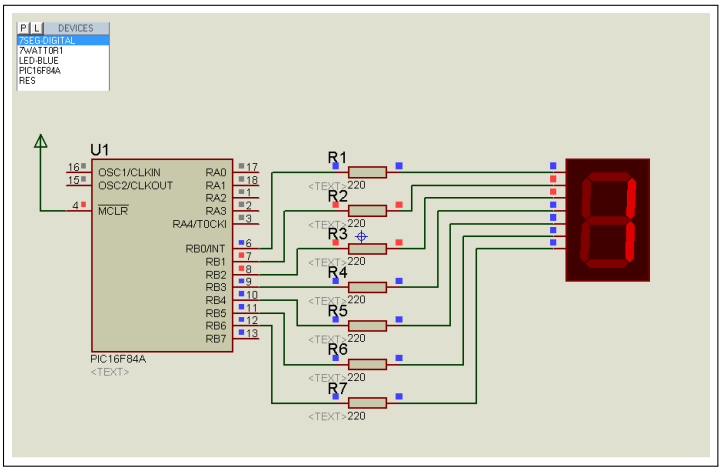


FIGURE 2.3: Schéma électronique « commande d'un afficheur 7 Segments simple »

2.3.2 Application N°2 : Afficheur 7-segments BCD (BinaryCodedDecimal)

Le but de cette application est de réaliser la même fonctionnalité que la première, en utilisons un afficheurs 7-segements BCD. Cet afficheur est connecté au PORTB comme le montre la figure 2.4.

1. Modifier le premier programme pour qu'il puisse gérer le nouvel afficheur.
2. Modifier le schématique et le programme pour qu'il puisse gérer deux afficheurs 7-segment BCD.

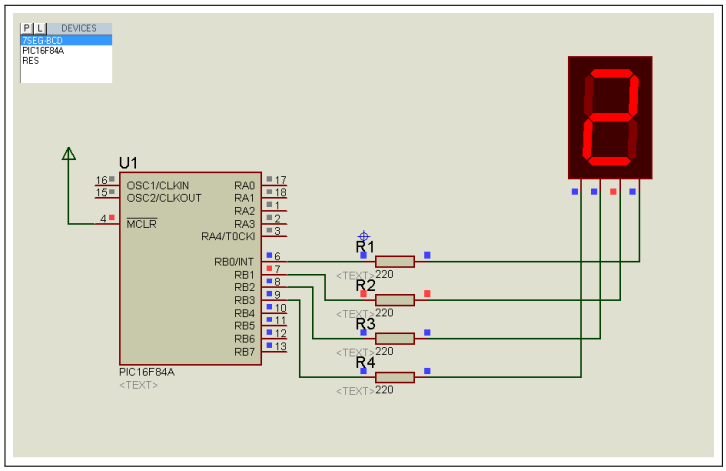


FIGURE 2.4: Schéma électronique « commande d'un afficheur 7 Segments BCD»

2.3.3 Application N°3 : Réalisation d'un chronomètre pour un feu tricolore

Le but de cette application est d'utiliser deux afficheurs 7-segements BCD qui serons connectés au PORTB, pour faire un chronomètre d'un feu tricolore qui sera connecté au PORTA comme le montre la figure 2.5.

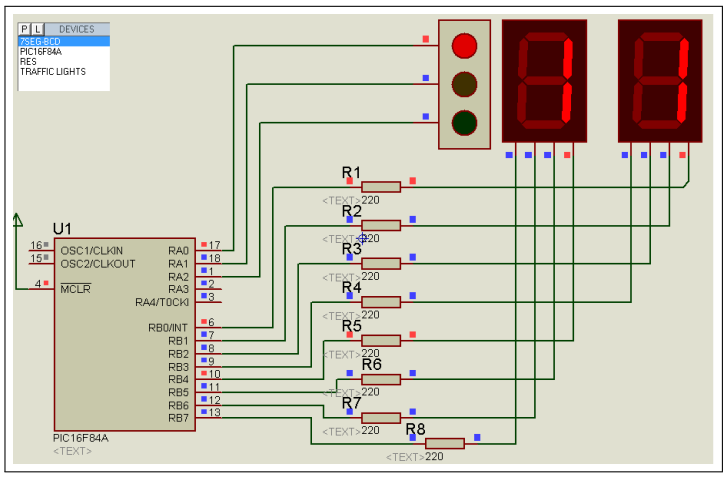


FIGURE 2.5: Schéma électronique « chronomètre pour un feu tricolore »

1. Réutiliser le programme fait dans le premier TP pour établir le nouveau programme qui permet de gérer les deux afficheurs et les trois LED.

TP N° 3

Manipulation des entrées / sorties d'un microcontrôleur PIC16F84A (Commande d'un moteur)

3.1 Objectifs

- Maitriser les entrées et les sorties du PIC16F84A.
- Comprendre l'utilisation des actionneurs (bouton poussoir, switch et relai).
- Apprendre à contrôler un moteur en courant continue et pas à pas on utilisant le microcontrôleur PIC16F84A et les circuits intégrés L293D et ULN2003A.

3.2 Introduction

Il est intéressant de commander des applications qui contiennent une partie mécanique par des PICs. Trois types des moteurs sont généralement utilisés dans ces applications à savoir : les moteur DC, moteur pas à pas et les servomoteurs. Dans cette partie, nous nous intéressons au moteur DC et pas a pas.

3.2.1 Moteur DC

Un moteur DC est un convertisseur électromécanique permettant la conversion bidirectionnelle d'énergie entre une installation électrique parcourue par un courant continu et un dispositif mécanique. Pour faire simple, cela signifie qu'un moteur à courant continu va pouvoir convertir de l'électricité en énergie mécanique. Les moteurs DC ont ainsi la particularité de pouvoir fonctionner dans les 2 sens, suivant la manière dont le courant lui est soumis.

Dans le fonctionnement d'un moteur DC si on connecte la borne + du moteur à la borne + d'un générateur de courant continue (ou batterie) et la borne – du moteur à la borne – du générateur. Le moteur se mettra alors à tourner dans le sens horaire.

Maintenant, si on inverse les branchements et vous verrez que le moteur se mettra à tourner dans le sens anti-horaire. Si on diminue aussi la tension du générateur DC vous verrez également que le moteur tourne moins vite : nous allons donc pouvoir régler la vitesse de fonctionnement du moteur DC.

Pour faire tourner un moteur DC dans les 2 sens grâce à un microcontrôleur. Nous allons utiliser un composant de contrôle : le circuit intégrée L293D (Figure 3.1). Le schéma suivant détaille les différentes broches du composant L293D :

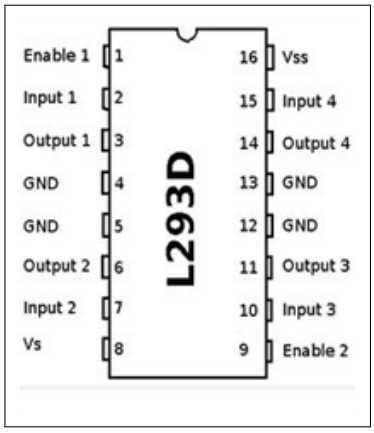


FIGURE 3.1: Branchement du L293D

Voici les caractéristiques techniques du composant L293D :

- Nbre de pont : 2 (on peut commander 2 moteurs) ;
- Courant Max Régime continu : 600mA (x2) ;
- Courant de pointeMax < 2ms : 1200mA ;
- VS Max Alim moteur : 36v ;
- VSS Max Alim logique : 7v ;
- Nbre de Broches : 16 DIP ;
- Perte de tension : 1.3v.

Avec un seul pont L293D et un PIC on va être capable de piloter 2 moteurs à courant continu indépendamment l'un de l'autre. Si la puissance de vos moteurs est faible vous pourrez même utiliser le 5V en sortie de votre PIC pour alimenter vos moteurs DC.

- Vss : Alimentation du circuit integré de commande (+5V).
- Vs : Alimentation de puissance des moteurs.
- GND : Doit être raccordé à la masse (GND).
- OUT1, OUT2 : Broches à raccorder à la charge (le moteur).
- IN1, IN2 : Broche de commande du pont 1. Se raccorde au PIC.
- E1ENABLE1 : permet d'envoyer (ou pas) la tension sur les sorties du moteur via OUT1 & OUT2.
- ENABLE1 commande l'activation du premier Pont :
  - Si ENABLE1 = GND, le pont 1 est déconnecté et le moteur ne fonctionne pas.
  - Si ENABLE1 = Vss ou 1 logique (+5v), le pont 1 est connecté aux sorties et le moteur fonctionne dans un sens ou l'autre ou pas en fonction des tensions appliquée sur INPUT1 & INPUT2.

3.2.2 Moteur pas à pas

Ce type de moteur se trouve dans un grand nombre de périphériques informatiques : imprimantes, lecteur de disquettes ou disque dur car il s'agit du composant mécanique par excellence pour tout ce qui demande une grande précision de positionnement. Les moteurs pas à pas sont très pratiques car ils permettent de faire tourner leur axe d'un angle précis. Un moteur pas à pas bipolaire est composé de 4 fils couplés deux par deux aux bobines constituant le moteur.

En électronique, on a souvent besoin de commander un périphérique d'une puissance respectable à partir d'un signal de faible puissance. Par exemple pour actionner un moteur à partir d'un microcontrôleur. Il existe de nombreuses solutions à ce problème : on peut par exemple utiliser un relais ou un transistor (BJT ou MOSFET). Ou encore un driver comme le circuit intégré ULN2003.

Ce circuit regroupe dans un même boitier 7 darlingtons. Ou dit plus simplement 8 interrupteurs commandés chacun par une broche du circuit (Figure 3.2). Chacun de ces 8 canaux peut être ouvert ou fermé indépendamment des autres et, dans le cas de l'ULN2003, est capable de piloter une charge jusqu'à 40V ou 500mA. La limitation principale résidant dans la puissance totale que peut dissiper le circuit et qui se situe autour de 0,7W pour une température ambiante de 85C.

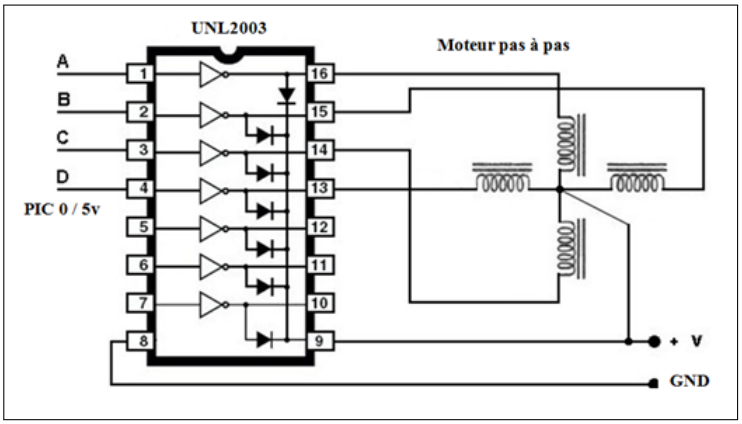


FIGURE 3.2: Brochage du driver UNL2003 avec un moteur pas à pas

3.3 Travail à réaliser

3.3.1 Application N°1 : Mise en marche d'un moteur DC

Le but de cette première application est la mise en marche d'un moteur DC à l'aide d'un PIC16F84A comme une première étape, ensuite le contrôle du sens de rotation à l'aide de circuit intégré L293D comme dernier étape.

Les étapes ci-dessous illustrent la réalisation de la première partie de cette application.

1. Lancer ISIS et le Compilateur mikroC PRO.
2. Réaliser le circuit de test (voir la figure 3.3) sous ISIS en utilisons :
   - Une alimentation supplémentaire (+12) pour fournir la puissance nécessaire au moteur DC.
   - Un Relai RL1 pour piloter la chargé électrique (+12v) vers le moteur.
   - Un bouton poussoir pour commander le relai.
   - Un Voltmètre pour visualiser la tension à la borne du moteur.

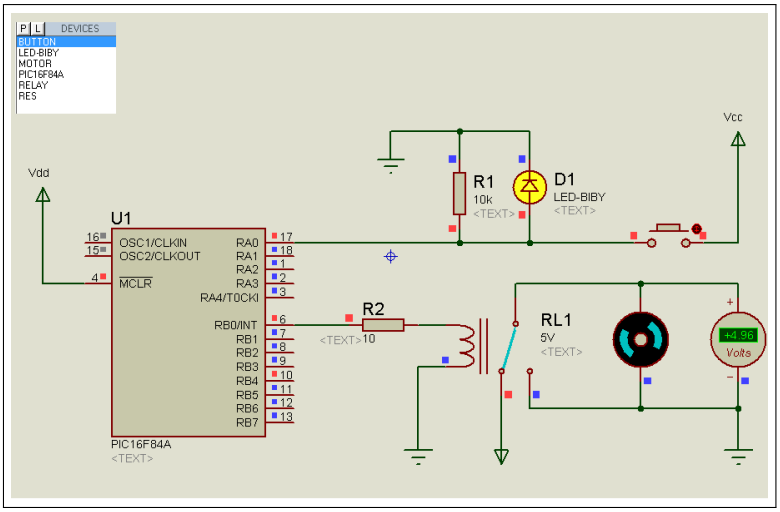


FIGURE 3.3: Schéma électronique « Commande d'un moteur DC à l'aide d'un PIC16F84A »

3. Ecrire le programme qui permet de gérer ce montage.

La deuxième partie de cette application est le contrôle du sens de rotation d'un moteur DC, Pour cela, il faut suivre les étapes ci-dessous :

1. Réaliser le circuit de test (voir la figure 3.4) sous ISIS en utilisons :
   - Circuit intégré L293D pour piloter le moteur.
   - Deux boutons poussoir pour changer la direction du moteur.
   - Circuit intégré L293D pour piloter le moteur.
   - Deux boutons poussoir pour changer la direction du moteur.

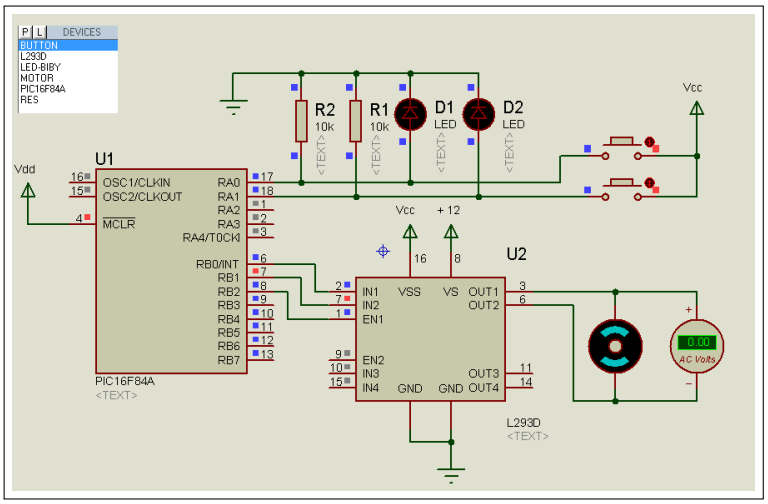


FIGURE 3.4: Schéma électronique « Commande du sens de rotation d'un moteur DC l'aide d'un PIC16F84A »

3.3.2 Application N°2 : Mise en marche d'un moteur pas à pas

Le but de cette application est la mise en marche d'un moteur pas à pas à l'aide d'un PIC16F84A,pour cela, il faut suivre les étapes ci-dessous :

1. Lancer ISIS et le Compilateur mikroC PRO.
2. Réaliser le circuit de test (voir la figure 3.5) sous ISIS en utilisant :
   - Une alimentation supplémentaire (+12) pour fournir la puissance nécessaire au moteur Pas à pas.
   - Un driver ULN2003A pour piloter le moteur.

— Un bouton poussoir pour commander le driver.

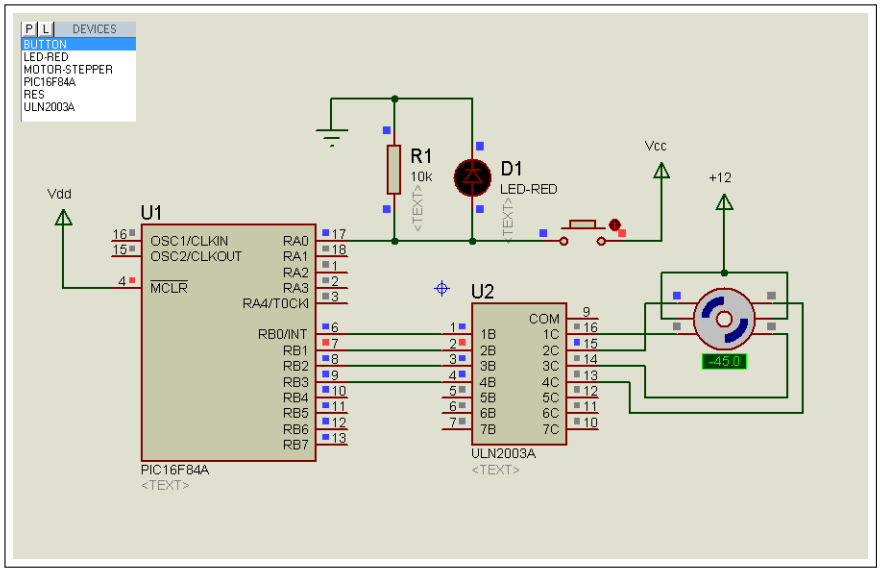


FIGURE 3.5: Schéma électronique « Commande d'un moteur pas à pas à l'aide d'un PIC16F84A »

3. Ecrire le programme qui permet la mise en marche du moteur par l'intermédiaire du driver ULN2003A.

3.3.3 Application N°3 : Commander le sens de rotation d'un moteur pas à pas

Dans cette application, nous allons commande le sens de rotation d'un moteur pas à pas dans le cas du PIC16F84A, comme le montre la figure 3.6.

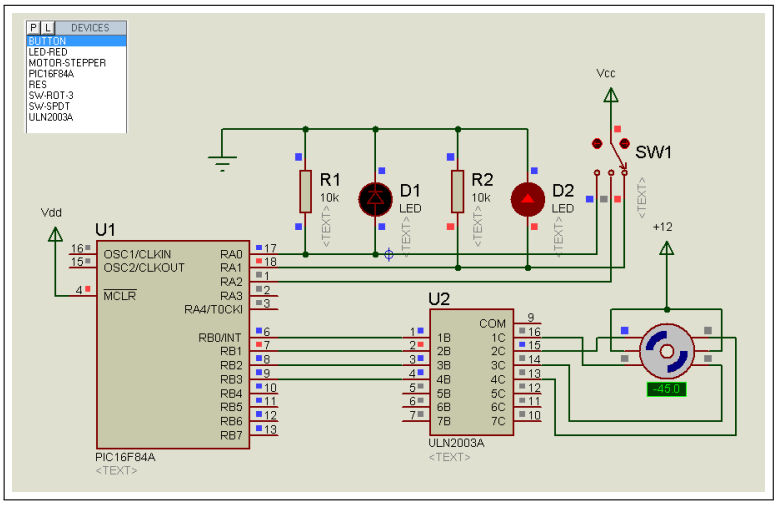


FIGURE 3.6: Schéma électronique « Utiliser le switch SW1 pour changé le sens de direction du moteur pas à pas »

1. Réutiliser le programme fait dans la partie précédente pour établir le nouveau programme qui permet de gérer le sens du moteur pas à pas.

2. Que pouvez-vous conclure ?

TP N° 4

Gestion des interruptions via le PIC16F84A

4.1 Objectifs

- Comprendre le mécanisme des interruptions.
- Mise en œuvre des interruptions à base du microcontrôleur PIC16F84A.

4.2 Introduction

Une interruption est un événement imprévisible qui provoque l'arrêt d'un programme en cours d'exécution pour aller exécuter un autre programme appelé programme (ou routine) d'interruption. A la fin du programme d'interruption, le microcontrôleur reprend le programme principal à l'endroit où il s'est arrêté comme le montre la figure 4.1.

On distingue deux types d'interruptions :

- Les interruptions externes, qui sont déclenchées lorsqu'un événement extérieur se produit tels que le changement d'état d'une entrée destinée à l'interruption.
- Les interruptions internes, qui sont déclenchées par le déroulement du programme tel que le résultat d'un calcul ou le débordement d'un Timer.

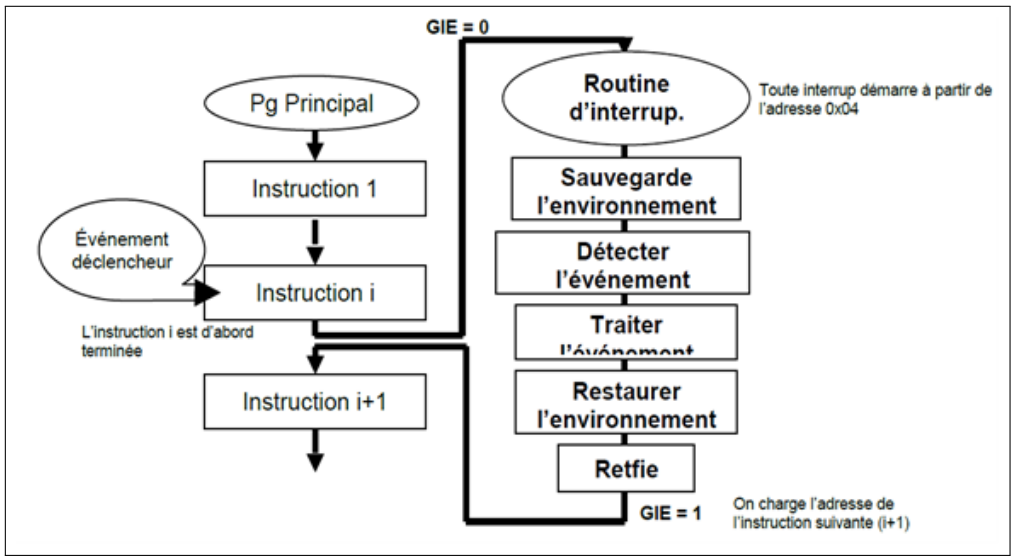


FIGURE 4.1: Séquence classique de fonctionnement d'une interruption

Dans le cas du 16F84 les différentes sources d'interruption sont au nombre de quatre :

1. **INT** : interruption externe sur la broche RB0/INT.
2. **TMR0** : interruption interne fin du comptage (débordement du registre TMR0).
3. **PORTB** : interruption externe sur changement du niveau logique d'au moins une de ces 4 broches : RB4, RB5, RB6 ou RB7 (port B).
4. **EEPROM** : interruption interne fin d'écriture en EEPROM

4.2.1 Registre de configuration des interruptions

Toute interruption est gérée à l'aide de 3 bits :

— Un bit d'activation globale (Global Enable bit). Ce bit permet d'activer ou de désactiver toutes les interruptions.

— Un bit d'activation (Enable bit). Ce bit permet d'activer ou de désactiver l'interruption correspondante.

— Un bit indicateur ou drapeau (Flag bit). Ce bit est mis à 1 lorsque l'interruption correspondante survient.

Pour le microcontrôleur PIC16F84A les bits de configuration des interruptions sont regroupés dans le registre INTCON (Figure 4.2).

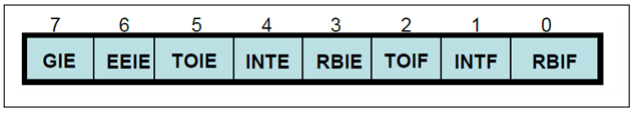


FIGURE 4.2: Structure du registre INTCON

- **GIE** : permet la validation générale des interruptions.
- **EEIE** : autorise l'interruption de fin d'écriture de l'EEPROM.
- **TOIE** :autorise l'interruption de débordement du registre TMR0.
- **INTE** :autorise l'interruption sur la broche RB0/INT.
- **RBIE** :autorise l'interruption sur les broches RB4, RB5, RB6, RB7 du port B.
- **TOIF** :drapeau (flag) mis à 1 lors du débordement du registre TMR0.
- **INTF** : drapeau (flag) mis à 1 lors d'un front (montant ou descendant selon l'état du bit INTEDG du registre OPTION_REG) sur la broche RB0/INT.
- **RBIF** : drapeau (flag) mis à 1 lors d'un changement de niveau logique d'au moins une des broches : RB4, RB5, RB6 ou RB7 (cela ne concerne que les broches configurées en entrée).

Remarque 1 : le drapeau EEIF qui signale la fin de l'écriture sur l'EEPROM se trouve dans le registre spécial EECON1 (bit 4).

Remarque 2 : Pour tous les bites de type drapeau (flag), c'est le matériel qui le met à 1 lors de la survenue de l'interruption, mais le logiciel (c'est-à-dire vous, le programmeur) qui le met à 0 à la fin de la routine d'interruption.

4.3 Travail à réaliser

Dans ce qui suit, nous allons réaliser quelques applications permettant la mise en œuvre des interruptions. Ces applications vont nous permettre de bien comprendre le mécanisme des interruptions.

4.3.1 Application N°1 : Interruption sur la broche RB0/INT

Le but de cette première application est de programmer une interruption externe sur la broche RB0/INT permettant d'inverser l'allumage d'une LED (connectée à la broche RB0) à chaque pression sur un bouton poussoir connecté à la broche RB0.

1. Réaliser le circuit correspondant sous ISIS.
2. Afin de bien comprendre la programmation du mécanisme d'interruption du PIC16F84A, on vous donne le programme de cette première application :

```
void main() {
trisa=0x00 ; trisb=0x01 ;
INTCON.GIE=1 ;          //validation générale des interruptions
INTCON.INTE=1 ;         //autorise l'interruption sur la broche RB0/INT
OPTION_REG.INTEDG=1 ;
//l'interruption de la broche RB0/INT est activé sur un front montant
while(1){
// attendre le déclenchement de l'interruption pour l'allumage de la LED
}
}
voidinterrupt()                 // routine d'interruption
{ porta.f0 = ∼porta.f0 ;        // inverser l'état de RA0
INTCON.INTF=0 ;                 //fin de l'interruption RB0/INT
}
```

Important : En MikroC, le sous-programme d'interruption est déclaré en tant que fonction avec le nom spécial « interrupt ». Cette fonction s'exécute automatiquement en réponse aux évènements déclencheurs des interruptions activées par l'utilisateur.

4.3.2 Application N°2 : Interruptions sur les broches (RB4-RB7)

Soit une LED connectée sur le pin RA0 du portA. Utiliser l'interruption externe sur les pins (RB4-RB7) pour changer la fréquence de clignotement de la LED :

1. Utiliser une fréquence de clignotement égale à 1500ms après la pression sur le bouton connecté à RB4.
2. Utiliser une fréquence de clignotement égale à 1000ms après la pression sur le bouton connecté à RB5.
3. Utiliser une fréquence de clignotement égale à 500ms après la pression sur le bouton connecté à RB6.
4. Utiliser une fréquence de clignotement égale à 200ms après la pression sur le bouton connecté à RB7.

4.3.3 Application N°3 : Chronomètre

Nous allons maintenant réaliser un chronomètre simplifié en utilisant trois afficheurs 7-segments BCD et un bouton connecté à la broche RB0. Le temps doit s'afficher sur les trois afficheurs 7-segments sous le format suivant (Figure 4.3) :« m –ss » (où m représente les minutes et s représente les secondes).

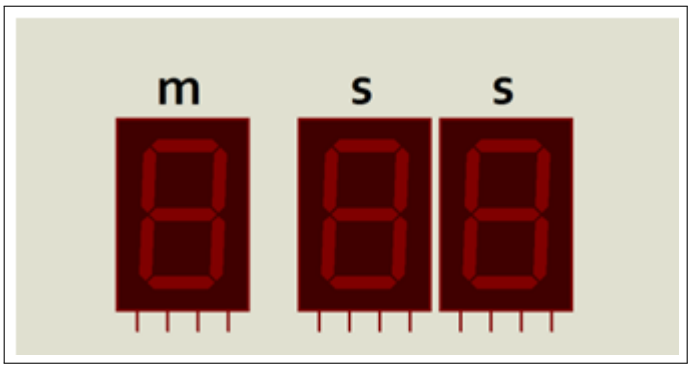


FIGURE 4.3: Format d'affichage du chronomètre

Programmer une interruption externe sur la broche RB0/INT permettant de réaliser le fonctionnement suivant :

1. A la mise sous tension, le chronomètre est initialisé à «0–00».
2. Un premier appui sur le bouton déclenche le chronomètre, le temps évolue sur les trois afficheurs.
3. Un deuxième appui sur le bouton arrête le chronomètre, le temps écoulé est affiché sur les trois afficheurs.
4. Un troisième appui sur le bouton réinitialisele chronomètre.

TP N° 5

Utilisation de l'interruption TMR0 via le PIC16F84A

5.1 Objectifs

- Comprendre le mécanisme de l'interruption TMR0.
- Comprendre l'utilisation du cycle d'horloge et le cycle d'instruction.
- Mise en œuvre de l'interruption TMR0 à base du microcontrôleur PIC16F84A.

5.2 Introduction

Dans le TP précédent nous avons déjà vue qu'il existe une interruption interne du Timer. Cette interruption utilise un registre de 8bits appelé TMR0 ou Timer0 qui s'incrémente à chaque instruction. Il est en quelque sorte, le fréquencemètre de nôtre Microcontrôleur. Plus la fréquence est élevée plus il compte vite, il a donc une relation avec le temps (Horloge).La figure 5.1 montre le registre TMR0 ainsi le mode de fonctionnement de l'interruption timer et l'ensemble des registres intervenants.

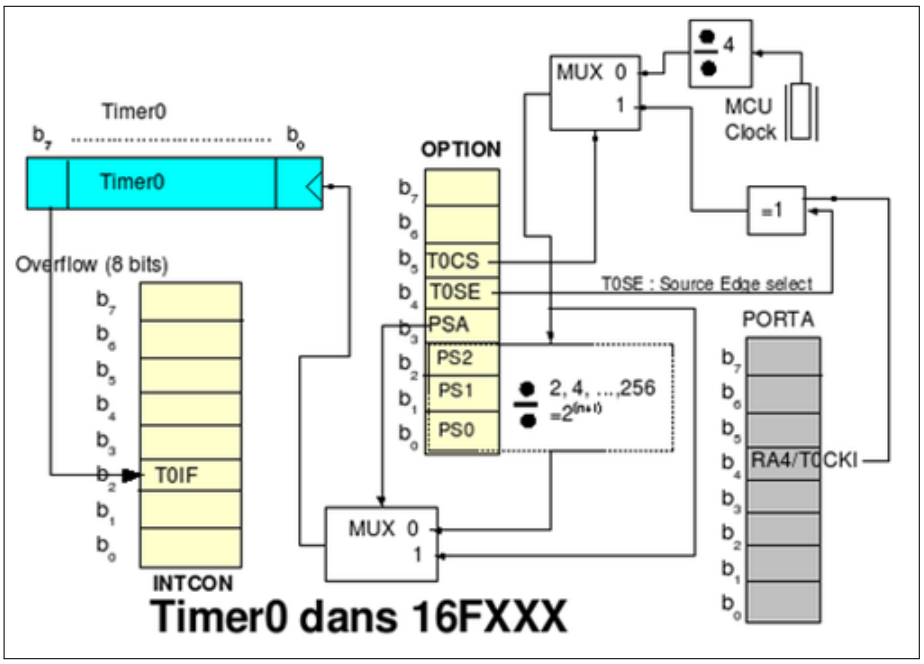


FIGURE 5.1: Les registres compléments pour le timer0

Les éléments remarquables dans la figure 5.1 :

- La fréquence du quartz est divisée par 4 avant utilisation (en haut à droite)
- C'est le registre OPTION qui permet une configuration du fonctionnement du TIMER0
- Lors d'un débordement (overflow) du timer 0, un bit nommé "T0IF" du registre INTCON est positionné à 1. Ce bit n'est jamais remis à 0 par le matériel car c'est un drapeau (flag).
- Le bit "T0CS" du registre OPTION permet, à l'aide du multiplexeur en haut à droite de choisir l'entrée du timer 0. On voit qu'il peut s'agir du quartz interne du PIC ou du bit "T0CKI" du PORTA.
- Le ou exclusif (=1) permet une éventuelle inversion
- Les trois bits "PS2", "PS1" et "PS0" permettent un choix de division entre 2 et 256
- La fameuse division ci-dessus ne sera prise en compte que si le multiplexeur du bas est positionné correctement par le bit "PSA" du registre OPTION.
- Cette figure vous donne aussi la position de tous les bits. Par exemple, le fameux bit "T0CKI" du PORTA est le bit RA4/T0CKI du PORTA.

Le tableau suivant présente la structure du registre **OPTION** (OPTION_REG)

	Nom	Description
bit 7	NOT_RBPU	**"Port B Pull-up Enable"** - Ce bit doit être mis à 0 pour **activer les résistances de pull-up du port B**. - Ce bit doit être mis à 1 pour désactiver les résistances de pull-up du port B.
bit 6	INTEDG	**"Interruptedge select"** - Ce bit doit être mis à 0 pour que l'interruption de la broche RB0/INT soit active sur un front descendant. - Ce bit doit être mis à 1 pour que l'interruption de la broche RB0/INT soit active sur un front montant.
bit 5	T0CS	**"TMR0 Clock Source Select"** - Ce bit doit être mis à 0 pour que l'horloge du module TMR0 soit l'horloge interne (un quart de la fréquence du signal OSC1/CLKIN). - Ce bit doit être mis à 1 pour que l'horloge du module TMR0 soit le signal de la broche RA4/T0CKI.
bit 4	T0SE	**"TMR0 Source edge select"** Dans le cas où T0CS = 1, le signal d'horloge de la broche RA4/T0CKI est actif : -Sur front montant quand T0SE = 0. -Sur front descendant quand T0SE = 1.
bit 3	PSA	**"Prescalerassignment"** Le prédiviseur est attribué : -Au module TMR0 quand PSA = 0

		-Au Watchdog quand PSA = 1
bits 2, 1, 0	PS2, PS1, PS0	**"Prescaler rate select"**

Table 5.1: Description de registre OPTION_REG

5.2.1 Paramètres du Prescaler

Pour configure le taux de pré-division du Timer TMR0, vous pouvez définir les champs PSA, PS2, PS1 et PS0 dans le registre OPTION comme le montre le tableau 5.2.

PSA	**PS2, PS1, PS0**	**Taux de pré-division du module TMR0**
0	000	2
0	001	4
0	010	8
0	011	16
0	100	32
0	101	64
0	110	128
0	111	256

Table 5.2: Paramètres du Prescaler

5.2.2 Principe de fonctionnement

Pour mieux comprendre la configuration du TMR0 et son fonctionnement, nous allons supposer que notre MicroControleur fonctionne à une fréquence de 8 Mhz.

Il faut savoir que le microcontrôleur a besoin de 4 cycles d'horloge, pour faire une instruction.

Donc le temps d'exécution d'une instruction = (1/8 Mhz)*4 = 0.5 μs.

Alors, notre compteur va s'incrémenter toutes les 0.5 μs.

Cela va tellement vite, que dans de nombreux cas, il va encore falloir ralentir notre Timer. Pour cela on utilise ce qu'on appelle un Prescaler, qui va encore diviser notre fréquence. Ce Prescaler est réglable par la configuration du registre OPTION (PSA, PS2, PS1, PS0).

Si on choisit de diviser notre fréquence par 256 (PSA=0, PS2=1, PS1=1, PS0=1), c'est-à-dire qu'en fonction du temps nous allons multiplier la période par 256.

0.5 μs * 256 = 128 μs.

Cela veut dire, que si nous avons un oscillateur a 8 Mhz, avec un prescaler réglé sur 256 nous allons avoir une instruction qui s'exécute toutes les 128 μs, donc notre compteur(TMR0) va s'incrémenter toutes les 128 μs.

Sachant que notre compteur TMR0 est un registre de 8bits,il pourra compter 256 instructions avant de mettre le Flag T0IF à 1. Ce qui implique une interruption chaque 128 μs * 256 = 32.768 m s.

Si à chaque interruption du Timer on incrémente un compteur, il faudra 31 incrémentation de ce compteur pour obtenir approximativement une seconde (1000 ms) (1000/32.768 = 30.51 $\approx$ 31 fois).

5.3 Travail à réaliser

Dans ce qui suit, nous allons reprendre l'application 3 du TP précédant pour réaliser un chronomètre simplifié en utilisant trois afficheurs 7-segments BCD et un bouton connecté à la broche RB0. Le temps doit s'afficher sur les trois afficheurs 7-segments sous le format : « m – s s » (où m représente les minutes et s représente les secondes).

1. Garder le même mode de fonctionnement et remplacer la fonction delay_ms par l'utilisation du timer0.
2. Pour améliorer votre chronomètre modifier le schéma en utilisant le composant 7SEG-MPX6-CA (voir figure 5.2) et le composant 4511 (voir figure 5.3).

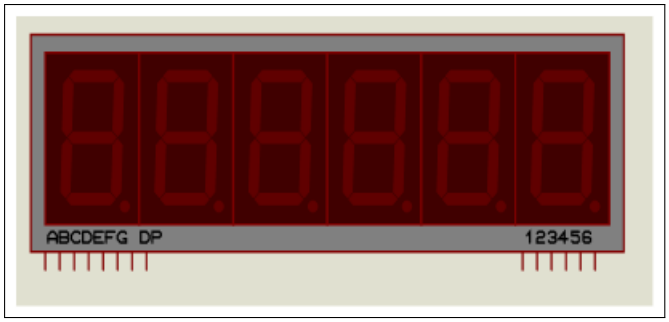


Figure 5.2: Afficheur 7 segments de type" 7SEG-MPX6-CA"

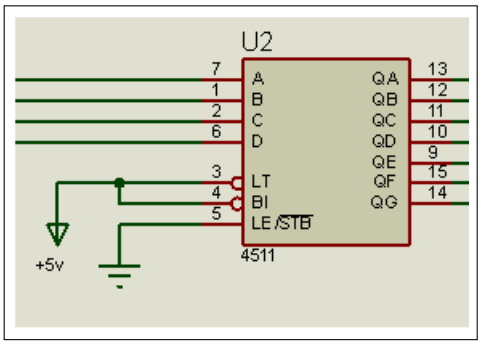


Figure 5.3: Composant 4511

Références

— Gérard Blanchet & Bertrand Dupouy. Architecture des ordinateurs : Principes fondamentaux . Hermes Science Publications.2013.

— JORDA, Jacques & M'ZOUGHI, Abdelaziz. Mini manuel d'architecture de l'ordinateur. Dunod, 2012.

— Pascal Mayeux. Apprendre la programmation des PIC High-Performance par l'expérimentation et la simulation. ETSF. 2010.

— Abdelmajid Oumnad.Les Microcontroleurs Etude Détaillée du PIC 16F887 Niveau C Broché – 17 juillet 2012. Technosup. 2012

— Christian Tavernier.Microcontrôleurs PIC 18. 2e édition - 2e édition.Description et mise en œuvre.Technique et ingénierie.2012

— VAN DAM Bert. 50 applications des microcontrôleurs PIC. Initiation et maîtrise par l'expérimentationInitiation et maîtrise par l'expérimentation. Coll. PUBLIT ELEKTOR.2010

— Pierre-Yves Rochat, EPFL.introduction au microcontrôleur. Ecole Polytechnique fédérale de Lausanne. 2016

— PIC16F84A Datasheet - Microchip Technology.

-A.A Chellal, H. Megnafi, A. Benhanifia, Design and conception of an autonomous watering system, arden (a muti application watering system), NewMat'21–1st International Conférence: New Trends on Innovative Construction Materials-ESSA-Tlemcen (Algeria)–22, 23 March, 2022.

-H. Megnafi, A. Ayad, W. Tabib, A. A Mouaziz, R. Ould Babaali, I. Medjhoud, Improved printing time by changing the mechanical part of the 3D printer, embedded system application, NewMat'21–1st International Conférence: New Trends on Innovative Construction Materials-ESSA-Tlemcen (Algeria)–22, 23 March, 2022.

-H. Megnafi, W.Y. Medjati, K. Haddouche, Implementation and use of Quadrotor UAV for the telecommunication applications, NewMat'21–1st International Conférence: New Trends on Innovative Construction Materials-ESSA-Tlemcen (Algeria)–22, 23 March, 2022.

-M.A. Brahami, H. Megnafi, H. Boukeffous, O. Benlaldj, Design and implementation of an embedded system for effective attendance management, NewMat'21–1st International Conférence: New Trends on Innovative Construction Materials-ESSA-Tlemcen (Algeria)–22, 23 March, 2022.

-Chellal, A. A., Gonçalves, J., Lima, J., Pinto, V., & Megnafi, H. (2021). Design of an Embedded Energy Management System for Li–Po Batteries Based on a DCC-EKF Approach for Use in Mobile Robots. Machines, 9(12), 313.

-Megnafi, H., & Medjati, W. Y. (2020, December). Study and Assembly of Quadrotor UAV for the Inspection of the Cellular Networks Relays. In International Conference in Artificial Intelligence in Renewable Energetic Systems (pp. 659-668). Springer, Cham.

-Chellal, A. A., Lima, J., Gonçalves, J., & Megnafi, H. (2021, July). Dual Coulomb Counting Extended Kalman Filter for Battery SOC Determination. In International Conference on Optimization, Learning Algorithms and Applications (pp. 219-234). Springer, Cham.

-Megnafi, H., Chellal, A. A., & Benhanifia, A. (2020, December). Flexible and automated watering system using solar energy. In International Conference in Artificial Intelligence in Renewable Energetic Systems (pp. 747-755). Springer, Cham.

Annexes

Liste des abréviations

Abréviation	Définition
BCD	Binary Coded Decimal
CISC	Complex Instruction Set Computer
CPU	Central Processing Unit
DSPIC	Digital Dignal Programmable Interface Controler
EEPROM	Electrically-Erasable Programmable Read-Only Memory
EPROM	Electrically-erasable programmable read-only memory
LCD	Liquid Crystal Display
LED	Light Emitting Diode
MCLR	Master Clear Reset
PIC	Programmable Interface Controler
PROM	Programmable Read Only Memory
RAM	Random Access Memory
RISC	Reduced Instruction Set Computer
ROM	Read-Only Memory
SFR	Special Function Registers
UAL	Arithmetic Logic Unit

7-Segment LED Controller Datasheet LED7SEG V 1.20

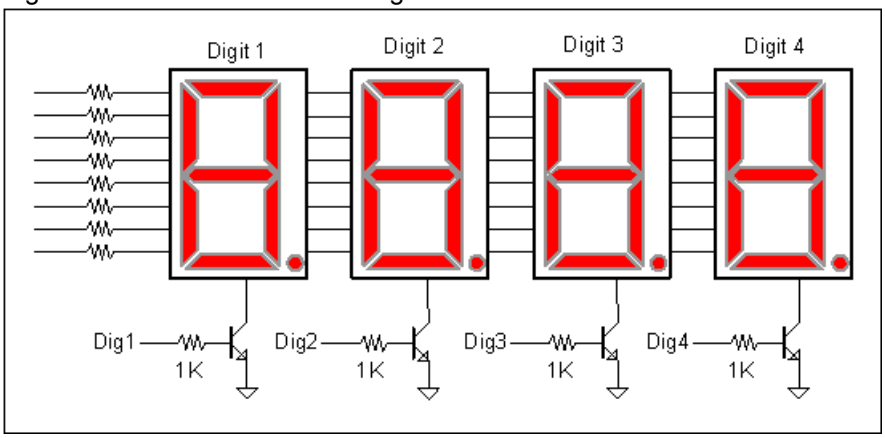
Resources	PSoC® Blocks			API Memory (Bytes)		Pins (per External I/O)
	Digital	Analog CT	Analog SC	Flash	RAM	
CY8C29/27/24/22/21xxx, CY8C23x33, CY8C28xxx, CY8CLED02/04/08/16, CY7C64215, CY8CPLC20, CY8CLED16P01, CYWUSB6953	1[a]	0	0	311	9	1

a. Only requires a digital block if the timer option is selected.

Features and Overview

- Supports 1 to 8 Digits
- Any combination of individual displays up to 8 total digits
- Displays both hex and integer values
- Supports decimal points built into 7-Segment display
- Supports both common cathode and common anode displays
- Configurable for both Active High and Active Low segment and digit drives

Figure 1. LED7SEG Block Diagram

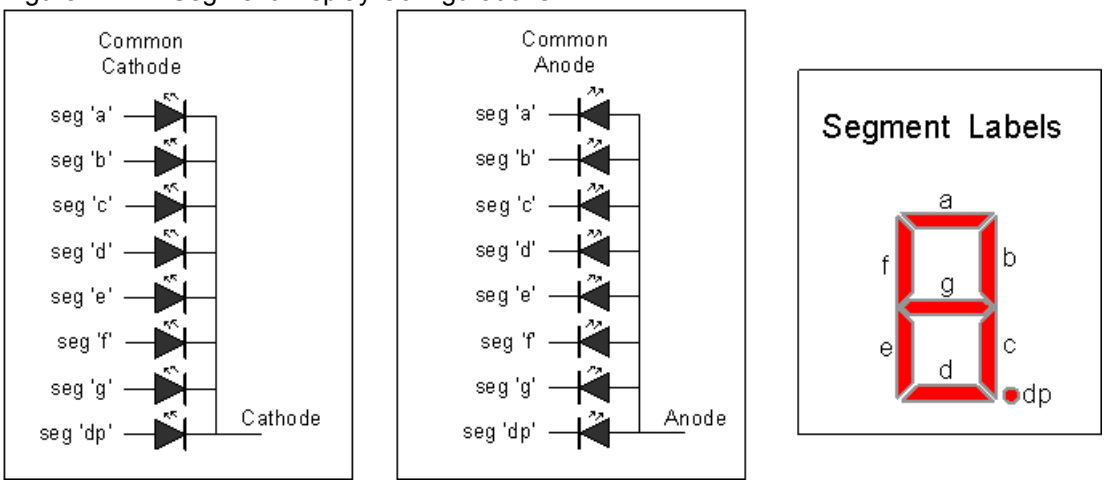


Functional Description

The LED7SEG User Module is capable of multiplexing up to eight 7-segment displays. This user module is compatible with common cathode, common anode, or any drive polarity. This allows a wide range of flexibility with various displays. Digits and segments may be driven directly by PSoC pins without the use of transistors or drivers as long as the current sinking and sourcing limits of the PSoC pins are not exceeded.

The following diagram shows how common anode and cathode 7-segment displays are configured:

Figure 2. 7-Segment Display Configurations



The project using the LED7SEG User Module that manipulates pins on a port shared with an instance of the LED7SEG must avoid direct PRTxDR writes. Use Shadow Registers for such manipulation to prevent incorrect LED7SEG operation.

How Multiplexing Works

When 2 or more displays are multiplexed, only one digit is on at a time, although to the eye it appears that all digits are on continuously. To achieve this illusion, the rate at which the displays are turned on and off must be higher than the response of the human eye. For a four digit display, the multiplex rate should be near 1 kHz; for an eight digit display the multiplex rate should be double that, or 2 kHz. When multiplexing an N-digit display, each digit is bon for 1/N of the total time. For example, for a system with 8 digits and a refresh rate of 2 kHz (500 uS period), each display is on for 500 uSec every 4 mSec (8 digits * 500 uSec/digits).

Why Multiplex?

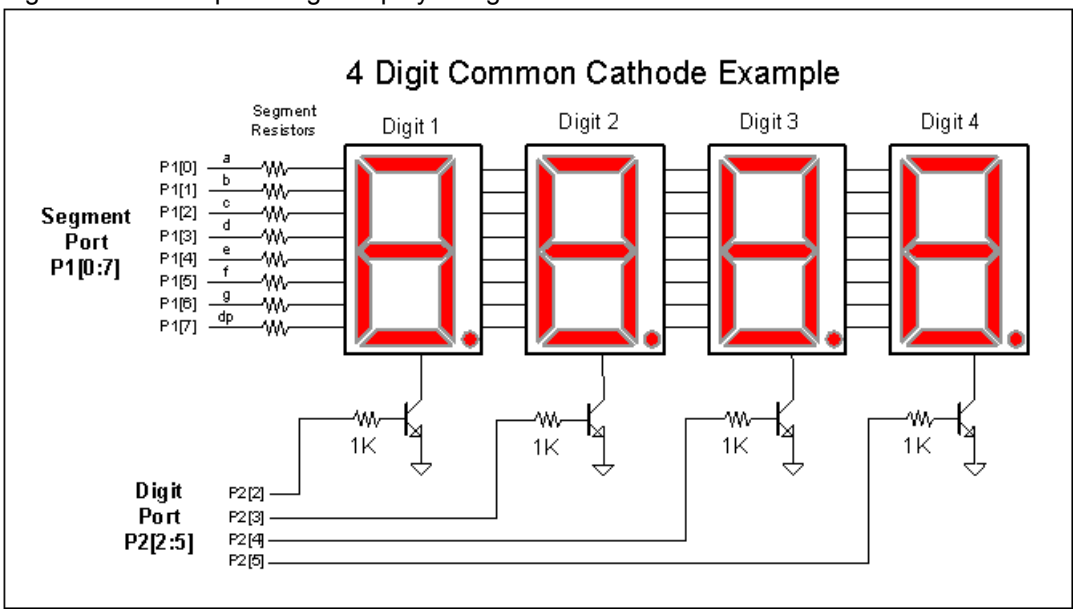
The main reason to multiplex a display is to save I/O pins. A multiplexed eight digit display requires only 16 pins. If the same display were driven directly, 64 I/O pins would be required. On the other hand, multiplexing does require some amount of CPU overhead. In most systems the overhead is small enough not to be of concern. With this user module, if the CPU speed is set to 12 MHz, the overhead is less than 2% with a multiplex rate of 1 kHz.

The following diagram shows an example of a four digit display using this user module. The digit numbering always starts from left (MSB) to right (LSB). Resistors must be used on the segment signals to limit the LED current and the PSoC sinking or sourcing current per pin. The segment resistors are connected between the PSoC port pins and all similar segments for all digits. For example, the segment “a" signals for all displays are connected in parallel, the segment “b" signals are connected in parallel, and so on. A full 8-bit port is required for the port used to drive the display segments (Segment port). In this

example, Port 1 is used. The port used to drive the common anode or cathode (Digit Port) only requires as many pins as there are digits, but the pins must be consecutive.

In this example port pins P2[2], P2[3], P2[4], and P2[5] are used. Port pins P2[0], P2[1], P2[2], and P[3] could have equally been used as well as several other combinations.

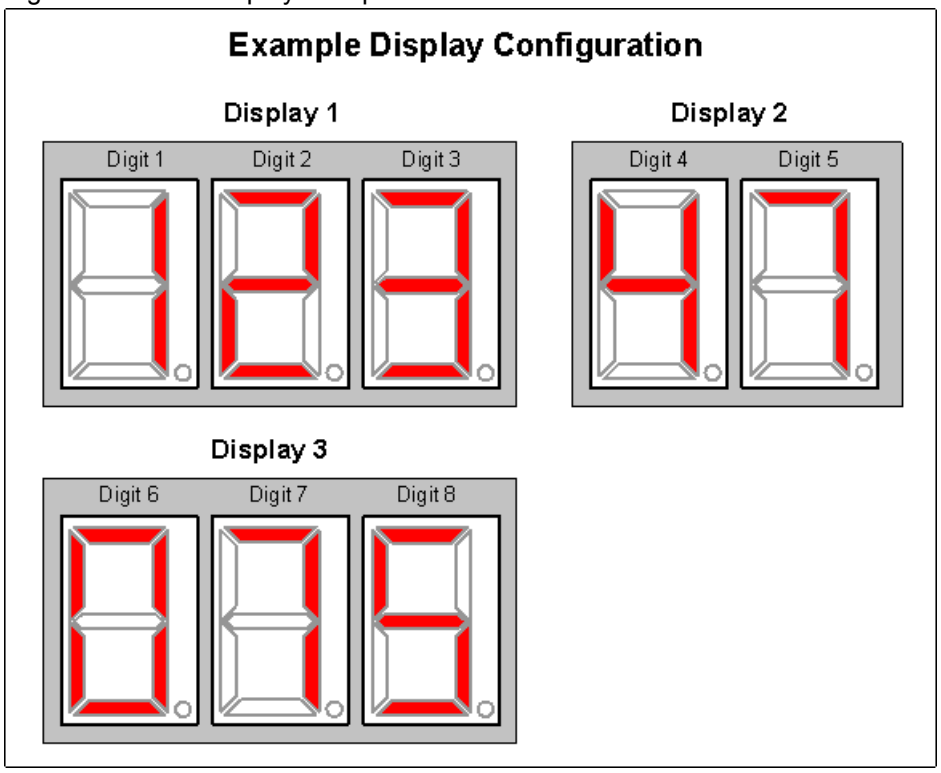
Figure 3. Example 4-Digit Display using LED7SEG User Module



Although the 1 to 8 digit display may look like one contiguous display, it may be arranged in one single eight digit display or eight 1 digit displays and everything in between. For example, you could configure 8 digits into two 3-digit displays and one 2-digit display, or maybe four 2-digit displays. The API can handle any combination as long as the groups of digits are contiguous.

In the following figure, eight digits have been arranged into three display groups. Display 1 contains Digits 1,2, and 3. Display 2 contains digits 4 and 5, and Display 3 contains digits 6, 7, and 8. Notice that although the 8 digits have been arranged in 3 groups, the digit numbers are still consecutive.

Figure 4. LED Display Groups



The following is an example of a code snippet that would work with the configuration defined above:

```
iNum1 = 123;
iNum2 = 47
iNum3 = 75;
LED7SEG_DispInt( iNum1, 1, 3);   // Start at Digit 1, three digits long
LED7SEG_DispInt( iNum2, 4, 2);   // Start at Digit 4, two digits long
LED7SEG_DispInt( iNum3, 6, 3);   // Start at Digit 6, three digits long.
```

DC and AC Electrical Characteristics

See the device datasheet.

Placement

Before placing the LED7SEG User Module, the user must choose between a built in multiplex timer and a user generated multiplex timer. If the build in multiplex timer is selected, a digital block is added to the project for the sole purpose of display multiplexing. This is the simplest option if an extra digital block is available. An extra user parameter “Multiplex Rate" is present, when this option is selected.

If an extra digital block is not available, the user may use any user module’s interrupt service routine that is periodic and close to 1 kHz or more. User modules that may have periodic interrupt service routines are counters, timers, PWMs, and some ADCs. The LED7SEG_Update() function call should be placed in any user module’s interrupt service routine. Unlike most functions, the LED7SEG_Update() function preserves both A and X registers.

Parameters and Resources

Segment Port

This parameter is used to select the GPIO port that drives the display segments (refer to Figure 3). All eight pins of the selected GPIO port are used by this user module. Any GPIO port that has all eight pins available is shown as an option to this parameter.

Digit Port

This parameter selects the GPIO port used for driving the "common" signals of the 7-segment display. Each digit of the display has one common signal that controls whether that digit is on or off. To see how the common signals connect to the 7-segment display hardware, refer to Figure 3. Whenever one of the common signals is in an active state, the cathodes of the LEDs that make up the corresponding digit are connected to ground. Then, if any of the segment signals are driven, the corresponding LEDs will light up. However, if the common signal for a digit is not active, the cathodes of the corresponding digit's LEDs are not connected to ground, and therefore the LEDs for that digit cannot be lit up. So, the common signal for a digit serves as a global on/off signal for the entire digit.

The pins that drive the common signals must all be part of the same GPIO port so that the code for this user module can be simple and efficient.

The selection for this parameter only chooses which GPIO port is to be used for the common signals. The Digit Drive Pins parameter chooses which specific pins within the chosen GPIO port are used.

Digit Drive Pins

This parameter is used to choose the number of digits that make up the 7-segment display and the specific pins that drive the "common" signals that correspond to each digit. Refer to Figure 3 to see how the common signals connect to the 7-segment display hardware.

For example, assume that a 7-segment display with three digits is being used and assume that the desired pins for the common signals are P2[2], P2[3], and P2[4]. In this case, "Port_2" must be chosen for the Digit Port parameter and "3 Digits Px[2:4]" must be chosen for the Digit Drive Pins parameter.

The 'x' character in the text for this parameter's options is a placeholder for the port number that was chosen with the Digit Port parameter. The "[x:y]" text in each parameter option specifies which pins of the port are to be used. The first digit is the first pin and the last digit is the last pin. So, "[2:4]" means GPIO pins 2, 3, and 4 of the GPIO port are to be used. The common signal pins must always be consecutive on the GPIO port. The first common pin (P2[2] in the example) must be connected to the first digit, the second common pin (P2[3] in the example) must be connected to the second digit, and so on.

Digit Drive Polarity

This parameter selects what polarity is required to turn on a single digit.

Option	Description
Active Low	Signal at pin driven low to turn on digit.
Active High	Signal at pin is driven high to turn on digit.

Segment Drive Polarity

This parameter selects what polarity is required to turn on the display's segments.

Option	Description
Active Low	Signal at pin driven low to turn on segment.
Active High	Signal at pin is driven high to turn on segment.

Multiplex Rate (Only valid if Multiplex timer option selected)

This function selects the display refresh rate. The table below shows the valid options for this parameter. The build in timer uses the 32kHz clock so that it would be independent of VC!, VC2, and VC3 clock. Larger displays should use the fastest multiplex rates. Smaller displays (2 to 4 digits) may use the lower rates. If the display seems to flicker, increase the multiplex rate.

Option	Description
500 Hz	Multiplex at a 500 Hz rate (2 to 3 digits)
1 kHz	Multiplex at a 1 kHz rate (3 to 5 digits)
2 kHz	Multiplex at a 2 kHz rate (4 to 6 digits)
4 kHz	Multiplex at a 4 kHz rate (6 to 8 digits)

Application Programming Interface

The Application Programming Interface (API) routines are provided as part of the user module to allow the designer to deal with the module at a higher level.

Though some user module API function may leave A and X unchanged, there is no guarantee they may do so in the future.

For Large Memory Model devices, it is also the caller's responsibility to perserve any value in the CUR_PP, IDX_PP, MVR_PP, and MVW_PP registers. Even though some of these registers may not be modified now, there is no guarantee that will remain the case in future releases.

Each time a user module is placed, it is assigned an instance name. By default, PSoC Designer assigns the LED7SEG_1 to the first instance of this user module in a given project. It can be changed to any unique value that follows the syntactic rules for identifiers. The assigned instance name becomes the prefix of every global function name, variable and constant symbol. In the following descriptions the instance name has been shortened to LED7SEG for simplicity.

The following constants are defined in the user module header:

Constant	Value
LED7SEG_Digit1	0x01
LED7SEG_Digit2	0x02
LED7SEG_Digit3	0x04
LED7SEG_Digit4	0x08
LED7SEG_Digit5	0x10
LED7SEG_Digit6	0x20
LED7SEG_Digit7	0x40
LED7SEG_Digit8	0x80
LED7SEG_DimOn	0x01
LED7SEG_DimOff	0x00
LED7SEG_DpOn	0x01
LED7SEG_DpOff	0x00

LED7SEG_Start

Description:

Clears all digit memory and enables scanning. If the multiplex timer is selected, it also enables the timer and enable its interrupt. Global interrupts need to be enabled for the multiplexing to work.

C Prototype:

```
void  LED7SEG_Start(void)
```

Assembler:

```
lcall  LED7SEG_Start
```

Parameters:

None

Return Value:

None

Side Effects:

None

LED7SEG_Stop

Description:

Stops display scan and turns off all digits. If the multiplexing timer is selected, this function also disables the timer and its interrupt.

C Prototype:

```
void  LED7SEG_Stop(void)
```

Assembler:

```
lcall  LED7SEG_Stop
```

Parameters:

None.

Return Values:

None

Side Effects:

The update ISR no longer scans the display.

LED7SEG_PutHex

Description:

Places a single hex digit (0 through F) at the given digit.

C Prototype:

```
void  LED7SEG_PutHex(BYTE bValue, BYTE bDigit)
```

Assembler:

```
mov  a,0x5     ; Write a '5' to the display
mov  x,0x01    ; Write the '5' at digit 1

lcall  LED7SEG_PutHex
```

Parameters:

bValue: Hex value between 0 and F to display.

bDigit: Digit location to place the given bValue.

Return Value:

None

Side Effects:

The A and X registers may be altered by this function.

LED7SEG_PutPattern

Description:

Writes a byte pattern directly to the display memory. A high bit corresponds to an active segment no matter if the display is common anode or cathode. The LSB is segment ‘a’ and the MSB is the “dp", decimal point.

C Prototype:

```
void  LED7SEG_PutPattern(BYTE bPattern, BYTE bDigit)
```

Assembler:

```
mov  a,0xFF   ; Turn all segments on
mov  x,0x01

lcall  LED7SEG_PutPattern
```

Parameters:

bPattern: A bit pattern to display. For example, 0x80 activates only the decimal point and 0x7F activates all segments other than the decimal point.

bDigit: Digit location to place the given bValue.

Return Value:

None

LED7SEG_DP

Description:

Turn the decimal point on or off at the given location.

C Prototype:

```
void  LED7SEG_DP(BYTE bOnOff, BYTE bDigit)
```

Assembler:

```
mov  a,0x1
mov  x,0x01
lcall  LED7SEG_DP
```

Parameters:

bOnOff: 0 => Turn off decimal point, 1 => turn on decimal point.

bDigit: Digit location for decimal point.

Return Value:

None

Side Effects:

The A and X registers may be altered by this function.

LED7SEG_Dim

Description:

Dim or undim the display. When dim is enabled, the update skips every other refresh cycle.

C Prototype:

```
void  LED7SEG_Dim(BYTE bDim)
```

Assembler:

```
mov   x,0x01          ; Dim the display
lcall  LED7SEG_Dim
```

Parameters:

bDim: 0 => Do not dim display, 1 => Dim display

Return Value:

None

Side Effects:

The A and X registers may be altered by this function.

LED7SEG_DispInt

Description:

Display an integer value on the display. It can be between 1 and 5 digits in length and start at any location. Range of numbers to display, 0 to 65535.

C Prototype:

```
void  LED7SEG_DispInt(int iValue, BYTE bPos, BYTE bLSD)
```

Assembler:

```
mov  X, SP
add   SP, 4
mov   [X], 0x34      ; Load MSB of value to display
mov   [X+1], 0x12  ; Load LSB of value to display
mov   [X+2], 0        ; Set starting location for value in display
mov   [X+3], 4        ; Set number of digits to display
lcall  LED7SEG_DispInt
```

Parameters:

iValue: Integer value to be displayed.

bPos: Digit position in display in which to start displaying the value.

BLSD: Length of digit to display, for example only display 4 digits.

Return Value:

None

Side Effects:

The A and X registers may be altered by this function.

LED7SEG_Update

Description:

The Update function controls the multiplexing rate of the display. Each time it is called, the previous displayed digit is turned off and the next is turned on. This function is automatically called periodically if the “LED7SEG with MPX Timer" is selected in the “User Module Selection Options..." menu. If the “LED7SEG without MPX Timer" option is chosen, the user needs to call this function periodically every 2 to 0.5 mSec to provide flicker free multiplexing. This function can be called in the users main program loop or from an existing interrupt service routine.

C Prototype:

```
void  LED7SEG_Update(void)
```

Assembler:

```
lcall  LED7SEG_Update
```

Parameters:

None

Return Values:

None

Side Effects:

All registers are preserved so there is no need to preserve A and X before calling this function.

Sample Firmware Source Code

Sample Assembly Code

```
;-------------------------------------------------------------------------
; // Example ASM code using LED7SEG User Module
;-------------------------------------------------------------------------

include "m8c.inc"       ; part specific constants and macros
include "memory.inc"    ; Constants & macros for SMM/LMM and Compiler
include "PSoCAPI.inc"   ; PSoC API definitions for all User Modules

export _main

_main:
    lcall  LED7SEG_Start

 ;; Enable multiplex timer here if using
    ;; non-built-in-timer version.

  M8C_EnableGInt         ; Enable global interrupts
     push  X
     mov   A,4              ; Push display size on stack "4"
     push  A
     mov   A,1              ; Push display start on stack "1"
     push  A
     mov   A,>1234          ; Push MSB of value 1234 on stack
     push  A
     mov   A,<1234          ; Push LSB of value 1234 on stack
```

```
     push  A
     lcall LED7SEG_DispInt  ; Display "1234"
     mov   X,3              ; Turn on decimal point at digit 3
     mov   A,1              ; Enable DP
     lcall LED7SEG_DP

.terminate:
    jmp .terminate
```

Sample C Code

```
//-------------------------------------------------------
// Example C code using LED7SEG User Module
//-------------------------------------------------------

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

void main(void)
{
   LED7SEG_Start();             // Enable display
   M8C_EnableGInt;              // Enable IRQ

// Enable multiplex timer here if using
   // non-built-in-timer version.

M8C_EnableGInt;
LED7SEG_DispInt(1234, 1, 4); // Display "1234"
   LED7SEG_DP(1, 3);            // Turn on DP ( 123.4 )
}
```

Configuration Registers

None

Version History

Version	Originator	Description
1.1	DHA	Added Version History.
1.20	DHA	Added support for CY8C21x12 devices.

Note PSoC Designer 5.1 introduces a Version History in all user module datasheets. This section documents high level descriptions of the differences between the current and previous user module versions.

FAIRCHILD SEMICONDUCTOR™

October 1987
Revised January 1999

CD4511BC
BCD-to-7 Segment Latch/Decoder/Driver

General Description

The CD4511BC BCD-to-seven segment latch/decoder/driver is constructed with complementary MOS (CMOS) enhancement mode devices and NPN bipolar output drivers in a single monolithic structure. The circuit provides the functions of a 4-bit storage latch, an 8421 BCD-to-seven segment decoder, and an output drive capability. Lamp test (LT), blanking (BI), and latch enable (LE) inputs are used to test the display, to turn-off or pulse modulate the brightness of the display, and to store a BCD code, respectively. It can be used with seven-segment light emitting diodes (LED), incandescent, fluorescent, gas discharge, or liquid crystal readouts either directly or indirectly.

Applications include instrument (e.g., counter, DVM, etc.) display driver, computer/calculator display driver, cockpit display driver, and various clock, watch, and timer uses.

Features

- Low logic circuit power dissipation
- High current sourcing outputs (up to 25 mA)
- Latch storage of code
- Blanking input
- Lamp test provision
- Readout blanking on all illegal input combinations
- Lamp intensity modulation capability
- Time share (multiplexing) facility
- Equivalent to Motorola MC14511

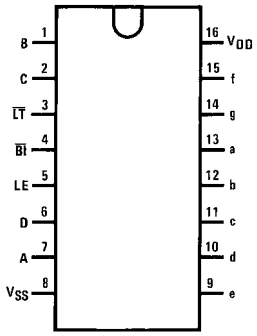
Ordering Code:

Order Number	Package Number	Package Description
CD4511BCWM	M16B	16-Lead Small Outline Integrated Circuit (SOIC), JEDEC MS-013, 0.300" Wide
CD4511BCN	N16E	16-Lead Plastic Dual-In-Line Package (PDIP), JEDEC MS-001, 0.300" Wide

Devices also available in Tape and Reel. Specify by appending suffix letter "X" to the ordering code.

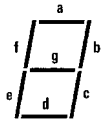
Connection Diagrams

Pin Assignments for SOIC and DIP



Top View

Segment Identification

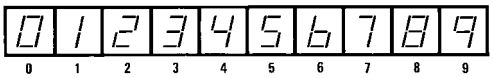


Truth Table

Inputs							Outputs							
LE	$\overline{BI}$	$\overline{LT}$	D	C	B	A	a	b	c	d	e	f	g	Display
X	X	0	X	X	X	X	1	1	1	1	1	1	1	B
X	0	1	X	X	X	X	0	0	0	0	0	0	0	
0	1	1	0	0	0	0	1	1	1	1	1	1	0	0
0	1	1	0	0	0	1	0	1	1	0	0	0	0	1
0	1	1	0	0	1	0	1	1	0	1	1	0	1	2
0	1	1	0	0	1	1	1	1	1	1	0	0	1	3
0	1	1	0	1	0	0	0	1	1	0	0	1	1	4
0	1	1	0	1	0	1	1	0	1	1	0	1	1	5
0	1	1	0	1	1	0	0	0	1	1	1	1	1	6
0	1	1	0	1	1	1	1	1	1	0	0	0	0	7
0	1	1	1	0	0	0	1	1	1	1	1	1	1	8
0	1	1	1	0	0	1	1	1	1	0	0	1	1	9
0	1	1	1	0	1	0	0	0	0	0	0	0	0	
0	1	1	1	0	1	1	0	0	0	0	0	0	0	
0	1	1	1	1	0	0	0	0	0	0	0	0	0	
0	1	1	1	1	0	1	0	0	0	0	0	0	0	
0	1	1	1	1	1	0	0	0	0	0	0	0	0	
0	1	1	1	1	1	1	0	0	0	0	0	0	0	
1	1	1	X	X	X	X				*				*

X = Don't Care
*Depends upon the BCD code applied during the 0 to 1 transition of LE.

Display



Absolute Maximum Ratings(Note 1)

DC Supply Voltage (V_{DD})	−0.5V to +18V
Input Voltage (V_{IN})	−0.5V to V_{DD} +0.5V
Storage Temperature Range (T_S)	−65°C to +150°C
Power Dissipation (P_D)	
Dual-In-Line	700 mW
Small Outline	500 mW
Lead Temperature (T_L)	
(Soldering, 10 seconds)	260°C

Recommended Operating Conditions

DC Supply Voltage (V_{DD})	3V to 15V
Input Voltage (V_{IN})	0V to V_{DD}
Operating Temperature Range (T_A)	−40°C to +85°C

Note 1: Devices should not be connected with power on.

DC Electrical Characteristics

Symbol	Parameter	Conditions	−40°C		+25°C			+85°C		Units
			Min	Max	Min	Typ	Max	Min	Max	
I_{DD}	Quiescent	V_{DD} = 5V		20			20		150	µA
	Supply Current	V_{DD} = 10V		40			40		300	µA
		V_{DD} = 15V		80			80		600	µA
V_{OL}	Output Voltage	V_{DD} = 5V		0.01		0	0.01		0.05	V
	Logical “0”	V_{DD} = 10V		0.01		0	0.01		0.05	V
	Level	V_{DD} = 15V		0.01		0	0.01		0.05	V
V_{OH}	Output Voltage	V_{DD} = 5V	4.1		4.1	4.57		4.1		V
	Logical “1”	V_{DD} = 10V	9.1		9.1	9.58		9.1		V
	Level	V_{DD} = 15V	14.1		14.1	14.59		14.1		V
V_{IL}	LOW Level	V_{DD} = 5V, V_{OUT} = 3.8V or 0.5V		1.5		2	1.5		1.5	V
	Input Voltage	V_{DD} = 10V, V_{OUT} = 8.8V or 1.0V		3.0		4	3.0		3.0	V
		V_{DD} = 15V, V_{OUT} = 13.8V or 1.5V		4.0		6	4.0		4.0	V
V_{IH}	HIGH Level	V_{DD} = 5V, V_{OUT} = 0.5V or 3.8V	3.5		3.5	3		3.5		V
	Input Voltage	V_{DD} = 10V, V_{OUT} = 1.0V or 8.8V	7.0		7.0	6		7.0		V
		V_{DD} = 15V, V_{OUT} = 1.5V or 13.8V	11.0		11.0	9		11.0		V
V_{OH}	Output	V_{DD} = 5V, I_{OH} = 0 mA	4.1		4.1	4.57		4.1		V
	(Source) Drive	V_{DD} = 5V, I_{OH} = 5 mA				4.24				V
	Voltage	V_{DD} = 5V, I_{OH} = 10 mA	3.6		3.6	4.12		3.3		V
		V_{DD} = 5V, I_{OH} = 15 mA				3.94				V
		V_{DD} = 5V, I_{OH} = 20 mA	2.8		2.8	3.75		2.5		V
		V_{DD} = 5V, I_{OH} = 25 mA				3.54				V
		V_{DD} = 10V, I_{OH} = 0 mA	9.1		9.1	9.58		9.1		V
		V_{DD} = 10V, I_{OH} = 5 mA				9.26				V
		V_{DD} = 10V, I_{OH} = 10 mA	8.75		8.75	9.17		8.45		V
		V_{DD} = 10V, I_{OH} = 15 mA				9.04				V
		V_{DD} = 10V, I_{OH} = 20 mA	8.1		8.1	8.9		7.8		V
		V_{DD} = 10V, I_{OH} = 25 mA				8.75				V
		V_{DD} = 15V, I_{OH} = 0 mA	14.1		14.1	14.59		14.1		V
		V_{DD} = 15V, I_{OH} = 5 mA				14.27				V
		V_{DD} = 15V, I_{OH} = 10 mA	13.75		13.75	14.18		13.45		V
		V_{DD} = 15V, I_{OH} = 15 mA				14.07				V
		V_{DD} = 15V, I_{OH} = 20 mA	13.1		13.1	13.95		12.8		V
		V_{DD} = 15V, I_{OH} = 25 mA				13.8				V
I_{OL}	LOW Level	V_{DD} = 5V, V_{OL} = 0.4V	0.52		0.44	0.88		0.36		mA
	Output Current	V_{DD} = 10V, V_{OL} = 0.5V	1.3		1.1	2.25		0.9		mA
		V_{DD} = 15V, V_{OL} = 1.5V	3.6		3.0	8.8		2.4		mA
I_{IN}	Input Current	V_{DD} = 15V, V_{IN} = 0V		−0.30		-10^{-5}	−0.30		−1.0	µA
		V_{DD} = 15V, V_{IN} = 15V		0.30		10^{-5}	0.30		1.0	µA

AC Electrical Characteristics (Note 2)

$T_A = 25°C$ and $C_L = 50$ pF, typical temperature coefficient for all values of $V_{DD} = 0.3\%/°C$

Symbol	Parameter	Conditions	Min	Typ	Max	Units
C_{IN}	Input Capacitance	$V_{IN} = 0$		5.0	7.5	pF
t_r	Output Rise Time	$V_{DD} = 5V$		40	80	ns
	(Figure 1**a**)	$V_{DD} = 10V$		30	60	ns
		$V_{DD} = 15V$		25	50	ns
t_f	Output Fall Time	$V_{DD} = 5V$		125	250	ns
	(Figure 1**a**)	$V_{DD} = 10V$		75	150	ns
		$V_{DD} = 15V$		65	130	ns
t_{PLH}	Turn-Off Delay Time	$V_{DD} = 5V$		640	1280	ns
	(Data) (Figure 1**a**)	$V_{DD} = 10V$		250	500	ns
		$V_{DD} = 15V$		175	350	ns
t_{PHL}	Turn-On Delay Time	$V_{DD} = 5V$		720	1440	ns
	(Data) (Figure 1**a**)	$V_{DD} = 10V$		290	580	ns
		$V_{DD} = 15V$		195	400	ns
t_{PLH}	Turn-Off Delay Time	$V_{DD} = 5V$		320	640	ns
	(Blank) (Figure 1**a**)	$V_{DD} = 10V$		130	260	ns
		$V_{DD} = 15V$		100	200	ns
t_{PHL}	Turn-On Delay Time	$V_{DD} = 5V$		485	970	ns
	(Blank) (Figure 1**a**)	$V_{DD} = 10V$		200	400	ns
		$V_{DD} = 15V$		160	320	ns
t_{PLH}	Turn-Off Delay Time	$V_{DD} = 5V$		313	625	ns
	(Lamp Test) (Figure 1**a**)	$V_{DD} = 10V$		125	250	ns
		$V_{DD} = 15V$		90	180	ns
t_{PHL}	Turn-On Delay Time	$V_{DD} = 5V$		313	625	ns
	(Lamp Test) (Figure 1 **a**)	$V_{DD} = 10V$		125	250	ns
		$V_{DD} = 15V$		90	180	ns
t_{SETUP}	Setup Time	$V_{DD} = 5V$	180	90		ns
	(Figure 1**b**)	$V_{DD} = 10V$	76	38		ns
		$V_{DD} = 15V$	40	20		ns
t_{HOLD}	Hold Time	$V_{DD} = 5V$	0	–90		ns
	(Figure 1**b**)	$V_{DD} = 10V$	0	–38		ns
		$V_{DD} = 15V$	0	–20		ns
PW_{LE}	Minimum Latch Enable	$V_{DD} = 5V$	520	260		ns
	Pulse Width (Figure 1 **c**)	$V_{DD} = 10V$	220	110		ns
		$V_{DD} = 15V$	130	65		ns

Note 2: AC Parameters are guaranteed by DC correlated testing.

Switching Time Waveforms

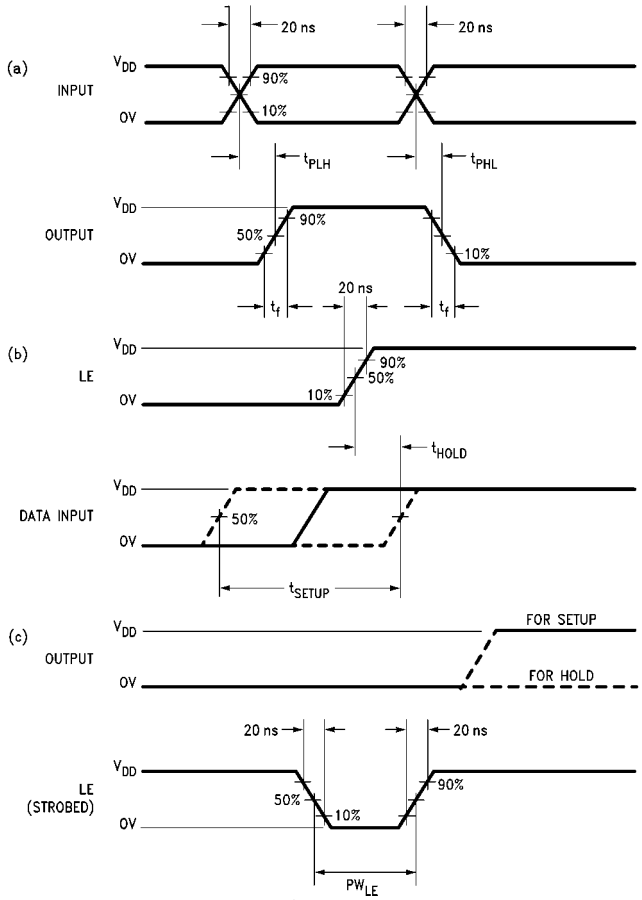
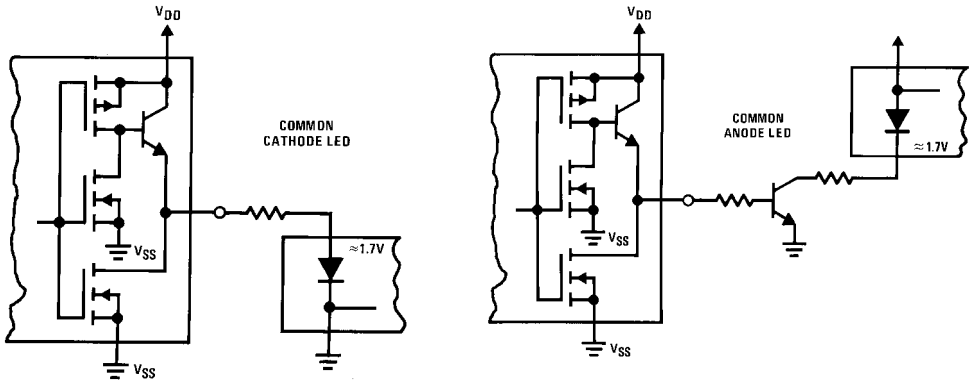


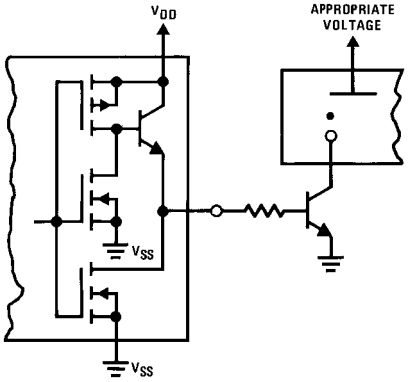
FIGURE 1.

Typical Applications

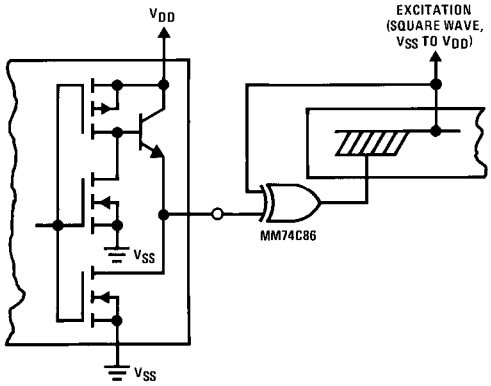
Light Emitting Diode (LED) Readout



Gas Discharge Readout

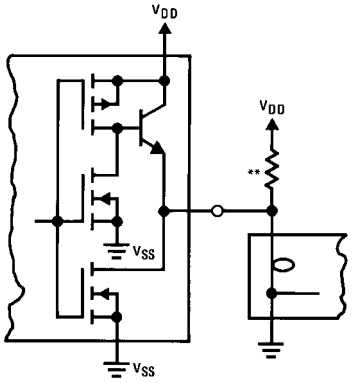


Liquid Crystal (LC) Readout

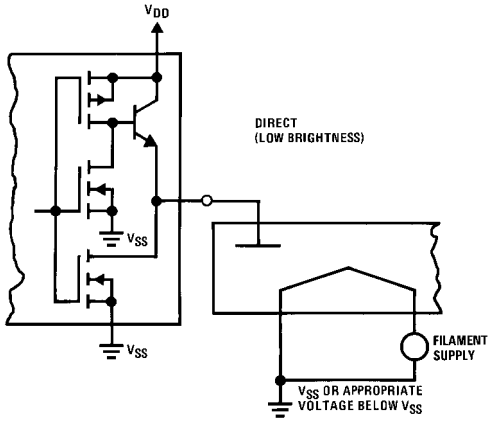


Direct DC drive of LC's not recommended for life of LC readouts.

Incandescent Readout

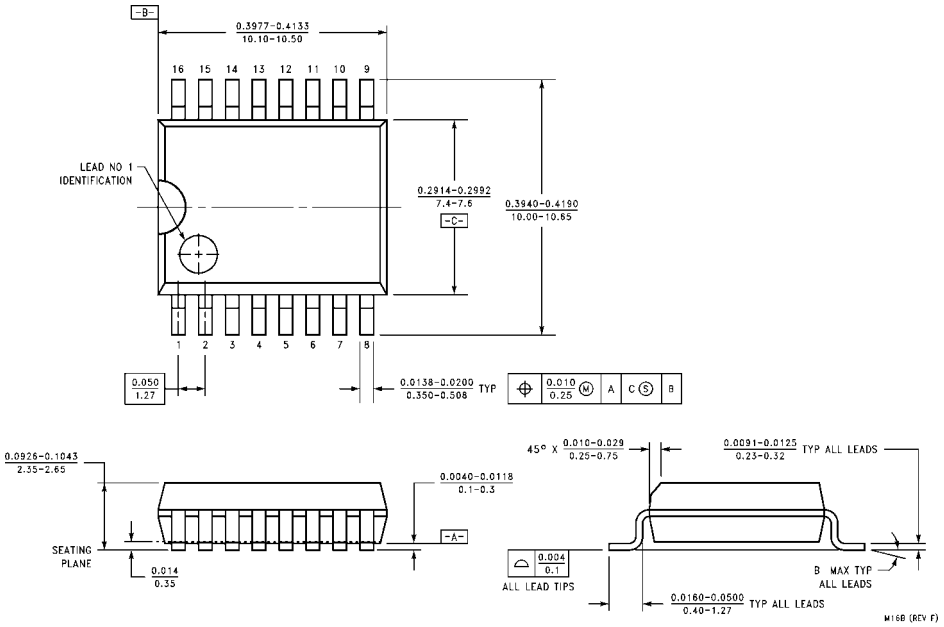


Fluorescent Readout



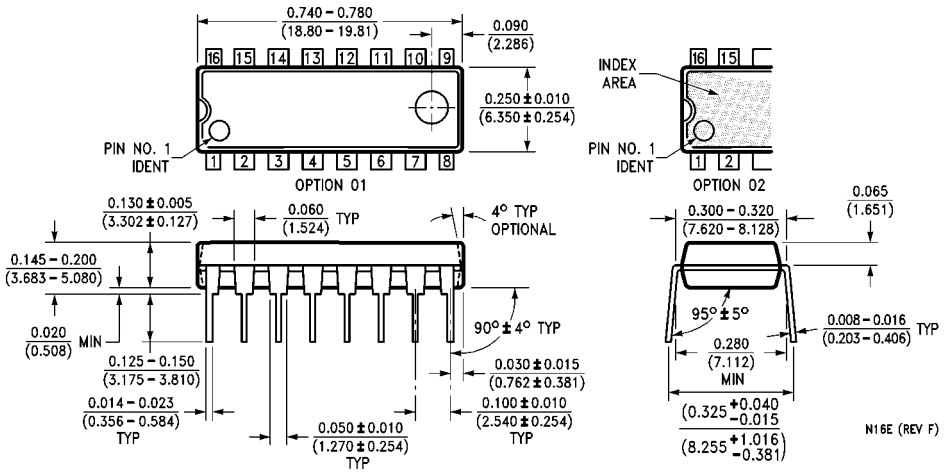
**A filament pre-warm resistor is recommended to reduce filament thermal shock and increase the effective cold resistance of the filament.

Physical Dimensions inches (millimeters) unless otherwise noted



16-Lead Small Outline Integrated Circuit (SOIC), JEDEC MS-013, 0.300" Wide
Package Number M16B

Physical Dimensions inches (millimeters) unless otherwise noted (Continued)



16-Lead Plastic Dual-In-Line Package (PDIP), JEDEC MS-001, 0.300" Wide
Package Number N16E

LIFE SUPPORT POLICY

FAIRCHILD'S PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS WRITTEN APPROVAL OF THE PRESIDENT OF FAIRCHILD SEMICONDUCTOR CORPORATION. As used herein:

1. Life support devices or systems are devices or systems which, (a) are intended for surgical implant into the body, or (b) support or sustain life, and (c) whose failure to perform when properly used in accordance with instructions for use provided in the labeling, can be reasonably expected to result in a significant injury to the user.
2. A critical component in any component of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system, or to affect its safety or effectiveness.

www.fairchildsemi.com