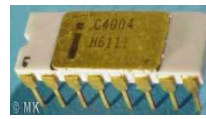


# CHAPTER 1

## A VERY BRIEF HISTORY OF MICROPROCESSORS & MICROCONTROLLERS

### Intel 4004



- o Introduced in 1971.
- o Federico Faggin design.



- o first-ever single-chip microprocessor
- o 2300 transistor
- o performs 60,000 operations / second.
- o approximately the same performance as the 18,000 vacuum tube ENIAC.
- o 4-bit Intel C4004 clock Speed is 108 kHz



### Intel 8008

- 1972: Faggin begins work on an **8-bit processor, the Intel 8008**. The prototype has serious problems with electrical charge leaking out of its memory circuits.
- Intel's 8008 is well-received, but system designers want increased speed, easier interfacing, and more I/O and instructions. The improved version is **8080**.

### Zilog Z80

- Faggin leaves Intel to start his own company **Zilog**, who later produce the **Z80**. 3.5MHz Z80 is a very popular processor taught in many universities)
- ... and, later, a 16-bit Z8000. But INTEL is getting more powerful.



### The Zilog Z80

- The Z80 is an 8 bit CPU with a 16 bit address bus capable of direct access of 64K of memory space.
- It was based on the 8080; it has a large instruction set.
- Programming features include an accumulator and six eight bit registers that can be paired as 3-16 bit registers. In addition to the general registers, a stack-pointer, program-counter, and two index (memory pointers) registers are provided.
- It had a 40 pin DIP package manufactured in A, B, and C models, differing only in maximum clock speed. It was also manufactured as a stand-alone microcontroller with various configurations of on-chip RAM and EPROM.
- It proves useful for low cost control applications.

27	M1	A0	30
		A1	31
19	MREQ	A2	32
20	I/O RD	A3	33
22	WR	A4	34
21	RD	A5	35
		A6	36
28	REFSH	A7	37
		A8	38
18	HALT	A9	39
		A10	40
24	WAIT	A11	1
		A12	2
16	INT	A13	3
17	MMI	A14	4
		A15	5
26	RESET	D0	14
25	BUSRD	D1	15
23	BUSWR	D2	16
		D3	17
6	CLK	D4	18
		D5	19
		D6	20
		D7	21

### Early Microcontrollers

- 1974: **Motorola** (originally car radio manufacturers) had introduced transistors in the 1950s and decided to make a late but serious effort in the microprocessor market. They announced their 8-bit **6800** processor.
- 1975: General Motors approach Motorola about a custom-built derivative of the 6800. Motorola's long experience with automobile manufacturers pays off and Ford follow GM's lead.
- 1976: Intel introduce an 8-bit microcontroller, the MCS-48. They ship 251,000 in this year.
- 1980: Intel introduced **8051**, an 8-bit microcontroller with on-board EPROM. They ship 22 million and 91 million in 1983.
- Early 90s: PIC is introduced to meet a demand for a cheap, small and practical microcontroller which was both easy to use and program.

# CHAPTER 2

## PIC Peripheral Interface Controller

## What is a microcontroller?

Microcontrollers are small computer on a chip with some special properties:

- CPU, code memory, data memory and IO ports all included on a single chip
- Dedicated to one task
- Small and low cost
- Embedded in many consumer devices

## Why PIC is popular?

PICs are popular with developers due to

- low cost
- wide availability
- large user base
- extensive collection of application notes
- availability of low cost or free development tools
- serial programming capability.

PIC is very small and easy to implement for not complex problems and usually accompanies to the microprocessors as an interface. For example:



## Family Core Architectural Differences

### Baseline Core Devices

(ex: 12C50x, 16C5x)

- 12 bit wide code memory,
- tiny two level deep call stack.
- **33 instructions**
- PIC10, PIC12 & PIC16 devices
- 6 pin to 40 pin packages.

### Mid-Range Core Devices

(ex: 12C50x, 16C5x)

- 14 bit wide code memory
- improved 8 level deep call stack.
- **35 instructions**
- increased opcode width allows addressing of more memory
- PIC12 and PIC16 devices.

### PIC17 High End Core Devices (Ex: 17C4x, 17C7xx)

- never became popular and superseded by the PIC18 architecture.
- 16 bit wide opcodes (allowing many new instructions) : **58 instructions**
- 16 level deep call stack. Packages of 40 to 68 pins.
- a memory mapped accumulator
- read access to code memory (table reads)
- direct register to register moves
- an external program memory interface to expand the code space
- an 8bit x 8bit hardware multiplier
- auto-increment/decrement addressing

### PIC18 High End Core Devices (ex: 18Cxxx)

- new high end pic architecture
- It inherits most of the features and instructions of the 17 series,
- **77 instructions**, much deeper call stack (31 levels deep)
- the call stack may be read and written
- offset addressing mode
- a new indexed addressing mode in some devices

### PIC24 and dsPIC 16 bit Microcontrollers

- architectures differ significantly from prior models.
- **dsPICs** are Microchip's newest family (started in 2004)
- **digital signal processing** capabilities.
- Microchip's first inherent 16-bit (data) microcontrollers.
- hardware **MAC** (multiply-accumulate)
- **barrel shifting**
- bit reversal
- (16x16)-bit multiplication
- other **digital signal processing** operations.
- Can be efficiently programmed in C

### PIC SPEED

- Can use crystals, clock oscillators, or even an RC circuit.
- Some PICs have a built in 4MHz RC clock, Not very accurate, but requires no external components!
- Instruction speed = 1/4 clock speed (T<sub>cyc</sub> = 4 \* T<sub>clk</sub>)
- All PICs can be run from DC to their maximum spec'd speed:

12C50x	4MHz
12C67x	10MHz
16Cxxx	20MHz
17C4x / 17C7xx	33MHz
18Cxxx	40MHz

## Register Addressing Modes

### Immediate Addressing

Movlw H'0F'

### Direct Addressing

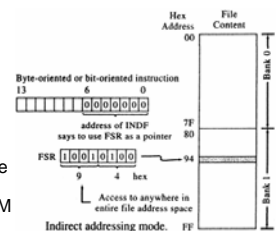
Uses 7 bits of 14 bit instruction to identify a register file address

8<sup>th</sup> and 9<sup>th</sup> bit comes from RP0 and RP1 bits of STATUS register.

Z equ D'2'  
btfs STATUS, Z

### Indirect Addressing

- Full 8 bit register address is written the special function register FSR
- INDF is used to get the content of the address pointed by FSR
- Exp : A sample program to clear RAM locations H'20' – H'2F'



PIC MEMORY CAPACITY					SPECIAL FUNCTION REGISTERS			
Device Type	Program size	Data EEPROM	Registers	Pins	Register	Address	Bank	Tutorial
PIC16F627	1024	128	224	18	EEADR	09	0	25
PIC16F628	2048	128	224	18	EEOCON1	08	1	25
PIC16F628	2048	128	224	18	EEOCON2	09	1	25
PIC16F83	512	64	36	18	EEDATA	08	0	25
PIC16C84	1024	64	36	18	FSR	04	0-1	16
PIC16F84	1024	64	68	18	INDF	00	0-1	16
PIC16F84A	1024	64	68	18	INTCON	0B	0-1	18
PIC16F870	2048	64	128	40	OPTION	01	1	18
PIC16F871	2048	64	128	40	PCL	02	0-1	4
PIC16F872	2048	64	128	40	PCLATH	0A	0-1	14
PIC16F873	4096	128	198	28	PORTA	05	0	4
PIC16F874	4096	128	192	40	PORTB	06	0	4
PIC16F876	8192	256	368	28	STATUS	03	0-1	2
PIC16F877	8192	256	368	40	TMR0	01	0	18
					TRISA	05	1	4
					TRISB	06	1	4

### STATUS REGISTER (16f84)

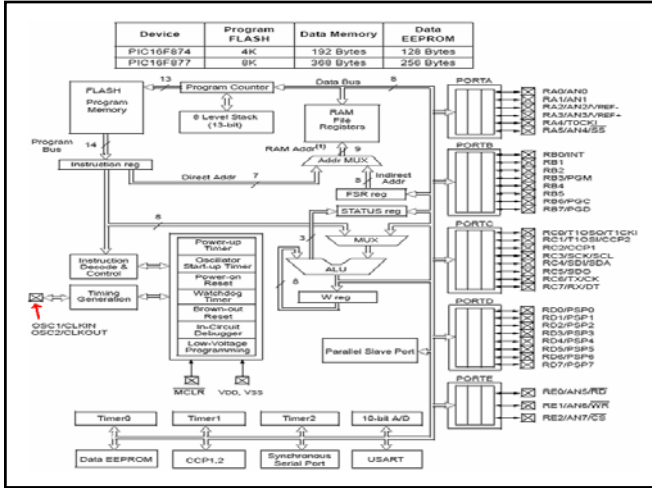
R/W-0	R/W-0	R/W-0	R-1	R-1	R/W-x	R/W-x	R/W-x	
IRP	RP1	RP0	TO	PD	Z	DC	C	bit0
bit 7: <b>IRP:</b> Register Bank Select bit (used for indirect addressing)								R = Readable bit
0 = Bank 0, 1 (00h - FFh)								W = Writable bit
1 = Bank 2, 3 (100h - 1FFh)								U = Unimplemented bit, read as '0'
The IRP bit is not used by the PIC16F8X. IRP should be maintained clear.								- n = Value at POR reset
bit 6-5: <b>RP1:RP0:</b> Register Bank Select bits (used for direct addressing)								
00 = Bank 0 (00h - FFh)								
01 = Bank 1 (80h - FFh)								
10 = Bank 2 (100h - 1FFh)								
11 = Bank 3 (180h - 1FFh)								
Each bank is 128 bytes.								
Only bit RP0 is used by the PIC16F8X.								
RP1 should be maintained clear.								
bit 4: <b>TO:</b> Time-out bit								
1 = After power-up, CLRWDI instruction, or SLEEP instruction 0 = A WDT time-out occurred								
bit 3: <b>PD:</b> Power-down bit								
1 = After power-up or by the CLRWDI instruction 0 = By execution of the SLEEP instruction								
bit 2: <b>Z:</b> Zero bit								
1 = The result of an arithmetic or logic operation is zero								
0 = The result of an arithmetic or logic operation is not zero								
bit 1: <b>DC:</b> Digit carry/borrow bit (for ADDWF and ADDLW instructions) (For borrow the polarity is reversed)								
1 = A carry-out from the 4th low order bit of the result occurred								
0 = No carry-out from the 4th low order bit of the result								
bit 0: <b>C:</b> Carry/borrow bit (for ADDWF and ADDLW instructions)								
1 = A carry-out from the most significant bit of the result occurred								
0 = No carry-out from the most significant bit of the result occurred								
<b>Note:</b> For borrow the polarity is reversed. A subtraction is executed by adding the two's complement of the second operand. For rotate (RRF, RLF) instructions, this bit is loaded with either the high or low order bit of the source register.								

### SAMPLE PROGRAM

List count	Prog count	Prog count	Code value	Source code
hex	hex	hex	hex	
0004	0	0000		STATUS EQU 3 ; name program location 3 as STATUS
0005	0	0000		PORTB EQU 6 ; name program location 6 as PORTB
0006	0	0000		
0007	0	0000		ORG 0 ; Reset Vector address
0008	0	0000	28 05	GOTO 5 ; go to PIC address location 5
0009	1	0001		ORG 4 ; Interrupt Vector address
0010	4	0004	28 05	GOTO 5 ; go to PIC address location 5
0011	5	0005		ORG 5 ; Start of Program Memory
0012	5	0005		
0013	5	0005	01 86	CLRF PORTB ; clear Port B data pins
0014	6	0006	16 83	BSF STATUS,5 ; set for Bank 1
0015	7	0007	01 86	CLRF PORTB ; set all Port B as output
0016	8	0008	12 83	BCF STATUS,5 ; set for Bank 0
0017	9	0009		
0018	9	0009	14 06	LOOPIT BSF PORTB,0 ; set Port B pin 0 to logic 1
0019	10	000A	14 86	BSF PORTB,1 ; set Port B pin 1 to logic 1
0020	11	000B	15 06	BSF PORTB,2 ; set Port B pin 2 to logic 1
0021	12	000C	15 86	BSF PORTB,3 ; set Port B pin 3 to logic 1
0022	13	000D	16 06	BSF PORTB,4 ; set Port B pin 4 to logic 1
0023	14	000E	16 86	BSF PORTB,5 ; set Port B pin 5 to logic 1
0024	15	000F	17 06	BSF PORTB,6 ; set Port B pin 6 to logic 1
0025	16	0010	17 86	BSF PORTB,7 ; set Port B pin 7 to logic 1
0026	17	0011	01 86	CLRF PORTB ; clear all PORTB pins
0027	18	0012	28 09	GOTO LOOPIT ; go to address LOOPIT

### PIC 16F877

- 8 kbytes of FLASH Prog. Mem. (8kx14)
- 368 bytes of Data Memory (RAM)
- 256 bytes of EEPROM Data Memory
- 33 input or output pins
- 20 MHz operating speed(200 ns instruction cycle)
- Max. 25 mA current from an output pin

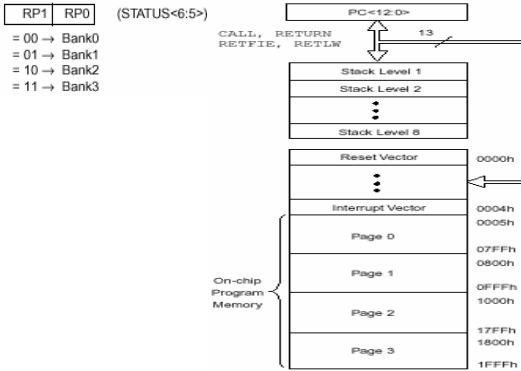


### PIC Program Memory

- 12C508 512 12bit instructions
- 16C71C 1024 (1k) 14bit instructions
- 16F877 8192 (8k) 14bit instructions
- 17C766 16384(16k) 16bit instructions

# PROGRAM MEMORY

## PIC16F877/876 PROGRAM MEMORY MAP AND STACK



# STATUS REGISTER (16f877)

RW-0	RW-0	RW-0	R-1	R-1	RW-x	RW-x	RW-x
IRP	RP1	RP0	TO	PD	Z	DC	C
bit7					bit0		

R = Readable bit  
W = Writable bit  
U = Unimplemented bit, read as '0'  
-n = Value at POR reset

bit 7: IRP: Register Bank Select bit (used for indirect addressing)  
1 = Bank 2, 3 (100h - 1Fh)  
0 = Bank 0, 1 (00h - FFh)

bit 6-5: RP1:RP0: Register Bank Select bits (used for direct addressing)  
11 = Bank 3 (180h - 1Fh)  
10 = Bank 2 (100h - 1Fh)  
01 = Bank 1 (80h - FFh)  
00 = Bank 0 (00h - 7Fh)  
Each bank is 128 bytes

# PIC Data Memory

PICs use general purpose "file registers" for RAM (each register is 8bits for all PICs)

Some examples are:

- 12C508 25 Bytes RAM
- 16C71C 36 Bytes RAM
- 16F877 368 Bytes (plus 256 Bytes of nonvolatile EEPROM)
- 17C766 902 Bytes RAM

Don't forget, programs are stored in program space (not in data space), so low RAM values are OK.

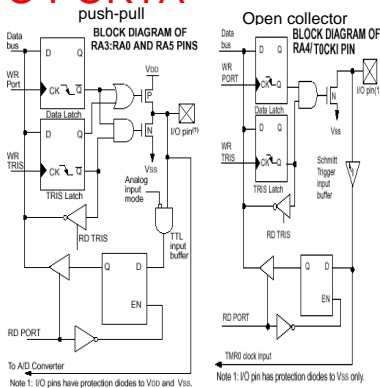
# DATA MEMORY

Indirect addr. <sup>(*)</sup>	File Address	Indirect addr. <sup>(*)</sup>	File Address
00h	00h	100h	100h
TMR0	01h	TMR0	101h
PCL	02h	PCL	102h
STATUS	03h	STATUS	103h
FSR	04h	FSR	104h
PORTA	05h	PORTA	105h
PORTB	06h	PORTB	106h
PORTC	07h	PORTC	107h
PORTD	08h	PORTD	108h
TRISA	09h	TRISA	109h
TRISB	0Ah	TRISB	10Ah
TRISC	0Bh	TRISC	10Bh
TRISD	0Ch	TRISD	10Ch
PCLATH	0Dh	PCLATH	10Dh
INTCON	0Eh	INTCON	10Eh
PIR1	0Fh	PIR1	10Fh
PIR2	10h	PIR2	110h
PIE1	11h	PIE1	111h
PIE2	12h	PIE2	112h
PCON	13h	PCON	113h
SSPBUF	14h	SSPBUF	114h
SSPCON	15h	SSPCON	115h
CCP1CON	16h	CCP1CON	116h
CCP1RS1	17h	CCP1RS1	117h
CCP1RS2	18h	CCP1RS2	118h
CCP2CON	19h	CCP2CON	119h
CCP2RS1	1Ah	CCP2RS1	11Ah
CCP2RS2	1Bh	CCP2RS2	11Bh
ADRESH	1Ch	ADRESH	11Ch
ADRESL	1Dh	ADRESL	11Dh
ADCON0	1Eh	ADCON0	11Eh
ADCON1	1Fh	ADCON1	11Fh
ADCON2	20h	ADCON2	120h
General Purpose Register 80 Bytes	EFh	General Purpose Register 80 Bytes	1EFh
accesses 70h-7Fh	F0h	accesses 70h-7Fh	1F0h
Bank 0	7Fh	Bank 2	17Fh
		Bank 3	1FFh

# I/O PORTA

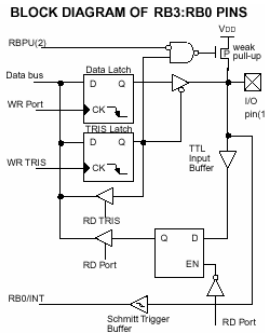
## EXAMPLE 3-1: INITIALIZING PORTA

```
BCF STATUS, RP0 ;
CLRF PORTA ; Initialize PORTA by
                ; clearing output
                ; data latches
BSF STATUS, RP0 ; Select Bank 1
MOVLW 0xC0F ; Value used to
            ; initialize data
            ; direction
MOVWF TRISA ; Set RA<3:0> as inputs
            ; RB<5:4> as outputs
            ; TRISA<7:6> are always
            ; read as '0'.
```

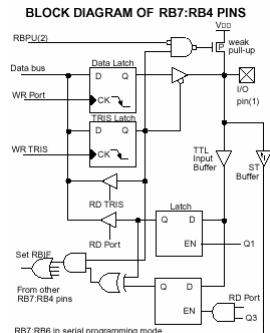


# I/O PORTB

## High Z inputs



## RB7:RB4 Interrupts



# CHAPTER 3

## PIC PROGRAMMING

Mnemonic, Operands	Description	Cycles	14-Bit Opcode				Status Affected	Notes
			Msb	LSb				
<b>BYTE-ORIENTED FILE REGISTER OPERATIONS</b>								
ADDFW	f,d Add W and f	1	00	0111	ffff	ffff	C,DC,Z	1,2
ANDWF	f,d AND W with f	1	00	1011	ffff	ffff	Z	1,2
CLRF	f Clear f	1	00	0001	1fff	ffff	Z	2
CLRWF	- Clear W	1	00	0001	0xxx	xxxx	Z	
COMF	f,d Complement f	1	00	1001	ffff	ffff	Z	1,2
DECf	f,d Decrement f	1	00	0011	ffff	ffff	Z	1,2
DECFSZ	f,d Decrement f, Skip if 0	1(2)	00	1011	ffff	ffff	Z	1,2,3
INCF	f,d Increment f	1	00	1010	ffff	ffff	Z	1,2
INCFSZ	f,d Increment f, Skip if 0	1	00	1111	ffff	ffff	Z	1,2,3
IORWF	f,d Inclusive OR W with f	1	00	0100	ffff	ffff	Z	1,2
MOVF	f,d Move f	1	00	1000	ffff	ffff	Z	1,2
MOVWF	f Move W to f	1	00	0000	1fff	ffff		
NOP	- No Operation	1	00	0000	0xxx	0000		
RLF	f,d Rotate Left f through Carry	1	00	1101	ffff	ffff	C	1,2
RRF	f,d Rotate Right f through Carry	1	00	1100	ffff	ffff	C	1,2
SUBWF	f,d Subtract W from f	1	00	0010	ffff	ffff	C,DC,Z	1,2
SWAPF	f,d Swap nibbles in f	1	00	1110	ffff	ffff	Z	1,2
XORWF	f,d Exclusive OR W with f	1	00	0110	ffff	ffff	Z	1,2
<b>BIT-ORIENTED FILE REGISTER OPERATIONS</b>								
BCF	f,b Bit Clear f	1	01	00bb	bfff	ffff		1,2
BSF	f,b Bit Set f	1	01	01bb	bfff	ffff		1,2
BTFSF	f,b Bit Test f, Skip if Clear	1(2)	01	10bb	bfff	ffff		3
BTFSF	f,b Bit Test f, Skip if Set	1(2)	01	11bb	bfff	ffff		3
<b>LITERAL AND CONTROL OPERATIONS</b>								
ADDLW	k Add Literal and W	1	11	11kk	kkkk	kkkk	C,DC,Z	
ANDLW	k AND Literal with W	1	11	1011	kkkk	kkkk	Z	
CALL	k Call subroutine	2	10	0kxx	kkkk	kkkk		
CLRWDT	- Clear Watchdog Timer	1	00	0000	0110	0100		TO,PD
GOTO	k Go to address	2	10	kkkk	kkkk	kkkk		
IORLW	k Inclusive OR Literal with W	1	11	1000	kkkk	kkkk	Z	
MOVLW	k Move Literal to W	1	11	00xx	kkkk	kkkk		
RETFIE	- Return from interrupt	2	00	0000	0000	1001		
RETlw	k Return with literal in W	2	11	01xx	kkkk	kkkk		
RETURN	- Return from Subroutine	2	00	0000	0000	1000		
SLEEP	- Go into standby mode	1	00	0000	0110	0011		TO,PD
SUBLW	k Subtract W from Literal	1	11	110x	kkkk	kkkk	C,DC,Z	
XORLW	k Exclusive OR Literal with W	1	11	1010	kkkk	kkkk	Z	

## Programming PIC 16F877

► Assembler (MPLAB)

► Basic (Pic Basic Pro)

► C (HITEC PICC)

```

main()
{
    set_tris_b(0x00);           0007    MOVLW    00
                               0008    TRIS     6

    while(true)
    {
        if (input(PIN_A0))     0009    BTFSF    05,0
                               000A    GOTO    00D

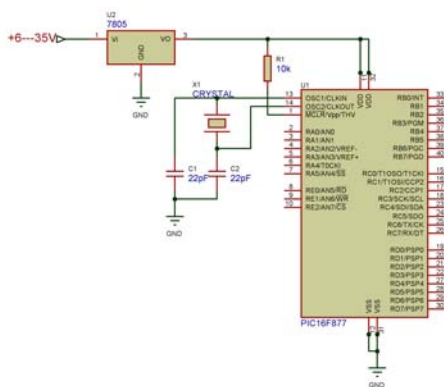
            output_high(PIN_B0); 000B    BSF     06,0

        else
            output_low(PIN_B0); 000C    GOTO    00E

    }
                               000D    BCF     06,0
                               000E    GOTO    009
}
    
```

## PIC BASIC PROGRAMMING

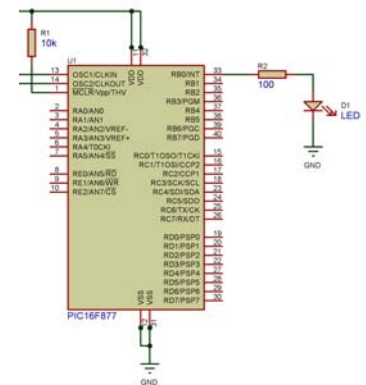
## Minimum circuitry for PIC16F877



## LED (light emitting diode) flasher

```

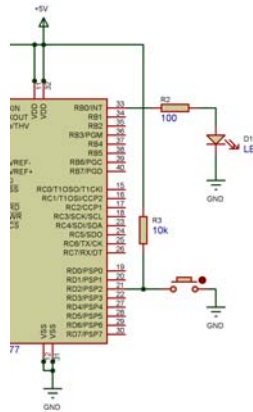
LOOP:
HIGH PORTB.0
PAUSE 500
LOW PORTB.0
PAUSE 500
GOTO LOOP
    
```



## BUTTON READ

```

INPUT    PORTD.2
LOOP:
IF PORTD.2=1 THEN
HIGH PORTB.0
ELSE
LOW PORTB.0
ENDIF
GOTO LOOP
    
```



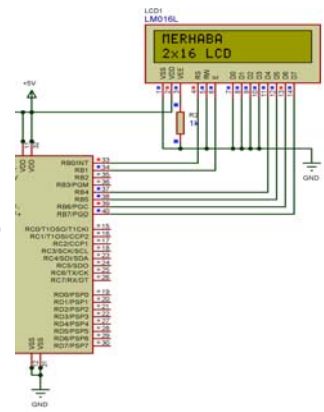
## LCD

```

DEFINE OSC 4
DEFINE LCD_DREG PORTB
DEFINE LCD_DBIT 4
DEFINE LCD_RSREG PORTB
DEFINE LCD_RSBIT 0
DEFINE LCD_EREG PORTB
DEFINE LCD_EBIT 1
DEFINE LCD_BITS 4
DEFINE LCD_LINES 2
DEFINE LCD_COMMANDUS 2000
DEFINE LCD_DATAUS 50

LCDOUT 254,1, "MERHABA"
LCDOUT 254,192, "2x16 LCD"

END
    
```



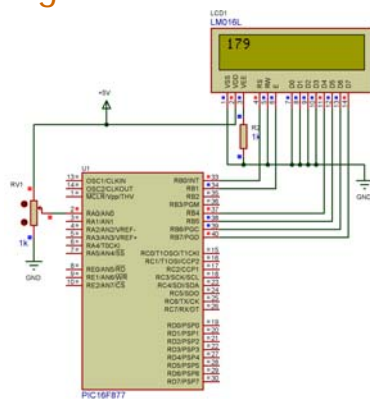
## Analog to Digital Conversion

Format: ADCIN  
Channel, Var

Sample Program:

```

ABC VAR BYTE
ADCON1=2 'PORTA is analog
INPUT PORTA.0
LOOP:
ADCIN PORTA.0,ABC
LCDOUT 254,1,#ABC
PAUSE 100
GOTO LOOP
    
```



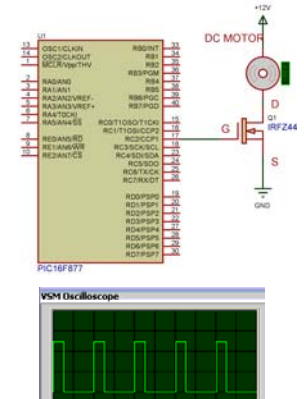
## Pulse Width Modulation

Format:

HPWM Channel, Duty cycle, Frequency  
Sample Program:

```

DEFINE CCP1_REG PORTC
'Hpwm 1 pin port
DEFINE CCP1_BIT 2
'Hpwm 1 pin bit
HPWM 1,64,1000
' Send a 25% duty cycle PWM signal at 1kHz
END
    
```



## Pulse Width Modulation

```

DEFINE CCP1_REG PORTC 'Hpwm 1 pin port
DEFINE CCP1_BIT 2 'Hpwm 1 pin bit
    
```

```

DUTY VAR BYTE
i VAR BYTE
    
```

DONGU:

```

FOR i=0 TO 255
HPWM 1,DUTY,1000
DUTY=DUTY+1
PAUSE 50
NEXT
GOTO DONGU
    
```

## Programming in C

- Programming the PIC in C offers several advantages:
  - Higher level language – developer is insulated from details of the chip
  - Library support for common tasks (string manipulation, serial communication)
- We use the CCS compiler (<http://www.ccsinfo.com/>) which don't suck. All examples will use CCS code

## PIC – Common Tasks with CCS

- Program Template. Starting point for just about everything

```
#define <16F877.h> // Define the type of chip you're using.
// Makes it easier to switch chips
#include delay(clock=2000000) // 20Mhz oscillator
void main()
{
    /* Initialization Code goes here */
    while (TRUE)
    {
        /* Program Code goes here */
    }
}
```

## PIC – Common Tasks with CCS

- Digital I/O

- Standard I/O vs. Fast I/O

- Using (standard I/O):

```
// Output a high on PIN_D1, low on PIN_D2
// Wait 50 us and invert
output_high(PIN_D1);
output_low(PIN_D2);
delay_us(50);
output_low(PIN_D1);
output_high(PIN_D2);
```

## PIC – Common Tasks with CCS

- Analog Input

- Initialization:

```
setup_adc_ports(ALL_ANALOG);
setup_adc(ADC_CLOCK_DIV_2);
```

- Picking a channel:

```
set_adc_channel(0); // Note: must wait
// between changing
// input channels (~ 10us)
```

- Inputting Data:

```
unsigned int16 data; // Declare a 16-bit
integer
data = read_adc(); // Read a 10-bit value
// from the selected channel
```

## PIC – Common Tasks with CCS

- Using PWM

- Initialization:

```
setup_timer_2(T2_DIV_BY_1,249,1); // Setup the
PWM period
setup_ccp1(CCP_PWM); // Set CCP1
for PWM
```

- Setting the Duty Cycle:

```
set_pwm1_duty(500); // See the CCS examples
for the formula // for setting the
PWM period and duty // cycle
```

## PIC – Tips for Software Design

- Design the program as a state machine

- A main() loop, with:

- A switch() statement that jumps to a function() which represents the actions that occur in that state
- Each state function() has an output section and a transition section (which can change the current state variable)

- Interrupts are very useful (for example: interrupt when data received on serial port), but can cause problems.

- I.e. if you change state during an interrupt (such as an E-stop), return from the interrupt service routine, then change the state variable again (during the transition section) the interrupt change is lost.

- Design with tuning and debugging in mind

- Programmer time is more important than machine time – the PIC16F877 is plenty fast

## PIC – Tips for Debugging

- Use a protoboard with RS232 support and lots of print statements. Example:

- program waits for a switch press

- reads an analog voltage

- changes the PWM cycle accordingly

## PIC – Tips for Debugging

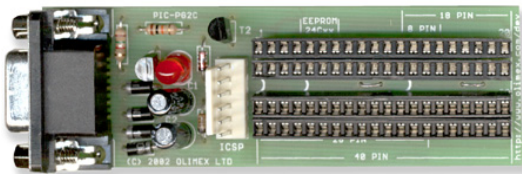
```

Some_function()
{
    int1 pushed = FALSE, last_pushed = FALSE;
    int16 analog_value;
    float volts;
    pushed = input(PIN_D3);
    if (pushed && !last_pushed) {
        puts("Button Pushed!");
        analog_value = read_adc(); /* 10-bit analog input value is
        * between 0-1023 0-5V range */
        volts = 5.0 * (analog_value / 1024.0);
        printf("Button pushed! Analog value is %f volts, PWM to %i\n, volts,
        analog_value);
        set_pwm1_duty(analog_value);
        /* We've pre-configured PWM channel 1 - the set_pwm1_duty cycle
        function accepts
        a 10-bit number and adjusts the cycle accordingly */
    }
}
    
```

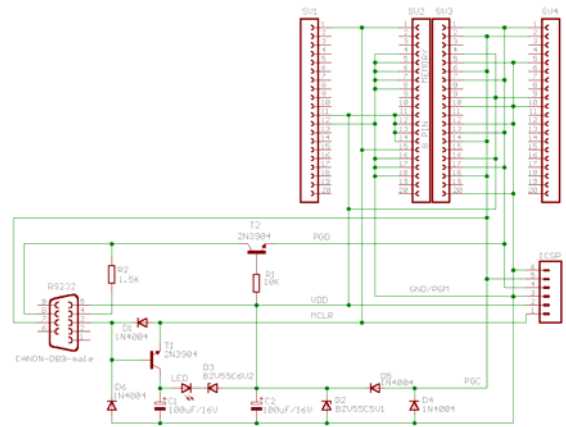
## CHAPTER 4

# PIC PROGRAMMERS

### PIC-PG2 - SERIAL PORT PROGRAMMER FOR 8/18/28/40 PIN PIC MICROCONTROLLERS AND I2C EEPROMS + ICSP connector and cable



<http://www.olimex.com/dev/pic-pg2.html>

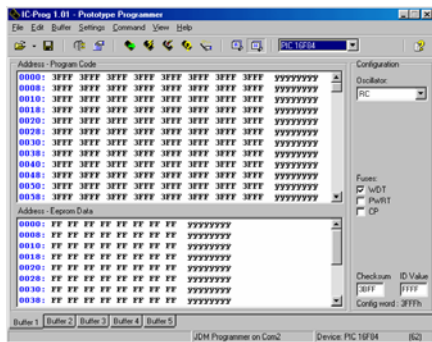


Copyright © 2002, OLIMEX Ltd  
<http://www.olimex.com/dev>

<http://www.ic-prog.com/>

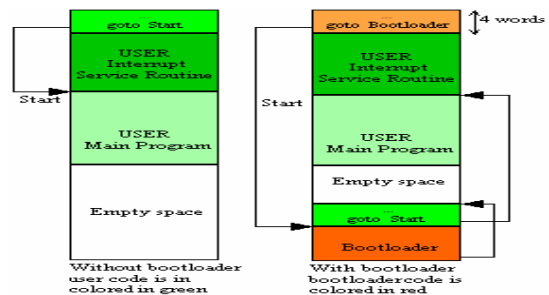
### IC-Prog Prototype Programmer

Programs :  
 12Cxx,  
 16Cxxx, 16Fxx,  
 16F87x,  
 18Fxxx, 16F7x,  
 24Cxx, 93Cxx,  
 90Sxxx, 59Cxx,  
 89Cx051,  
 89S53, 250x0,  
 PIC, AVR,  
 80C51 etc.

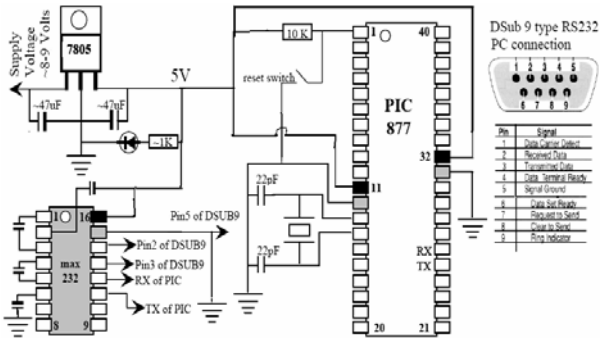


## BOOTLOADER

A bootloader is a program that stays in the microcontroller and communicates with the PC (usually through the serial interface). The bootloader receives a user program from the PC and writes it in the flash memory, then launches this program in execution

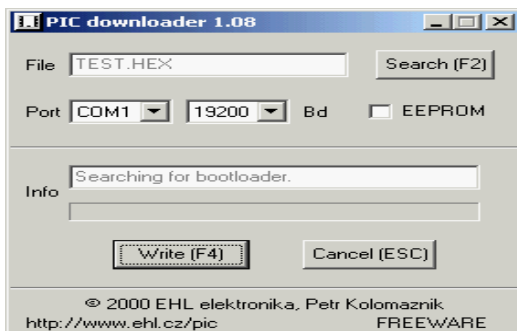


## BOOTLOADER



- The PIC16F877 has on-board FLASH memory
  - No burner needed to reprogram the PIC
  - No need to remove PIC from circuit
- Using a bootloader on the PIC, and a bootload utility on the PC the PIC can be reprogrammed in seconds over a serial link.
  - Burn the bootloader code onto the PIC
  - When writing your program in C tell the compiler not to use the top 255 bytes of flash memory
  - Connect the PIC circuit to the PC via a serial link. Run the bootloader code from the PC and download your code to the circuit in seconds.

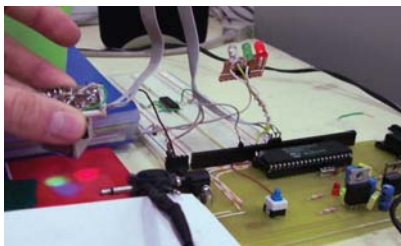
<http://www.microchip.com/PIC16bootload/>



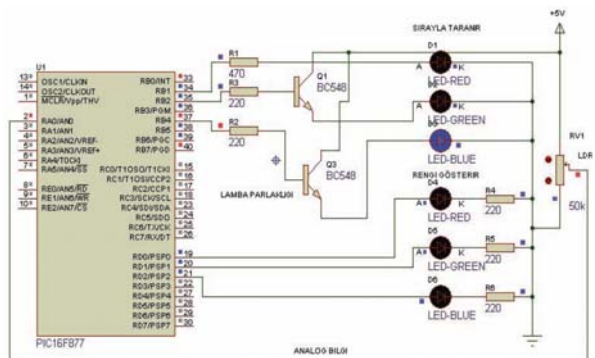
## CHAPTER 5

# PIC ISIS SIMULATION APPLICATIONS

## APPLICATION 1 PIC COLOR SENSOR

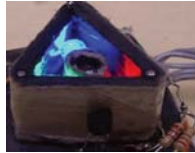


## ISIS CIRCUIT SCHEME



## DETAILS OF THE CIRCUIT

- Reads the color of the surface from the distance between 5-40mm. Three colors (red,blue,green) can be detected and shown on red,blue,green indicator LEDs.
- Red,Blue,Green LED diodes emit light, and the reflected light is read with LDR. LDR output is converted to a digital value using ADC channel 0 of PIC.



```

DEFINE OSC 4           '4 MHz clock
DEFINE ADC_BITS 8     '8 bit A2D
DEFINE ADC_CLOCK 3   'ADC Clock Adjust (rc = 3)
DEFINE ADC_SAMPLEUS 50
ADCON1 = 2           'PORTA is analog input
INPUT PORTA.0
TRISB = 0
SYMBOL VERI = PORTA.0 'Analog Data
SYMBOL REDIND= PORTB.3 'Indicator REDLED
SYMBOL GREENIND = PORTB.5 'Indicator GREENLED
SYMBOL BLUEIND = PORTB.4 'Indicator BLUELED

SYMBOL RED = PORTB.0   'Scanner REDLED
SYMBOL GREEN = PORTB.1 'Scanner GREENLED
SYMBOL BLUE = PORTB.2 'Scanner BLUELED
DAT VAR BYTE
DAT_R VAR BYTE
DAT_G VAR BYTE
DAT_B VAR BYTE
    
```

```

CLEAR
MAIN:
CALL READ_VAL
GOTO MAIN
READVAL:
HIGH RED
PAUSE 5
CALL ADC_RD
DAT_R = DAT
LOW RED
PAUSE 5
HIGH GREEN
PAUSE 5
CALL ADC_RD
DAT_Y = DAT
LOW GREEN
PAUSE 5
HIGH BLUE
PAUSE 5

CALL ADC_RD
DAT_B= DAT
LOW BLUE
PAUSE 5
IF DAT_R < BILGI_B && DAT_R < DAT_Y THEN
HIGH KIRMLD
LOW MAVILED
LOW YESILLED
ENDIF
IF DAT_G < DAT_B && DAT_G < DAT_R THEN
HIGH YESILLED
LOW MAVILED
LOW KIRMLD
ENDIF
IF DAT_B < DAT_G && DAT_B < DAT_R THEN
HIGH MAVILED
LOW KIRMLD
LOW YESILLED
PAUSE 20
ENDIF
RETURN
    
```

```

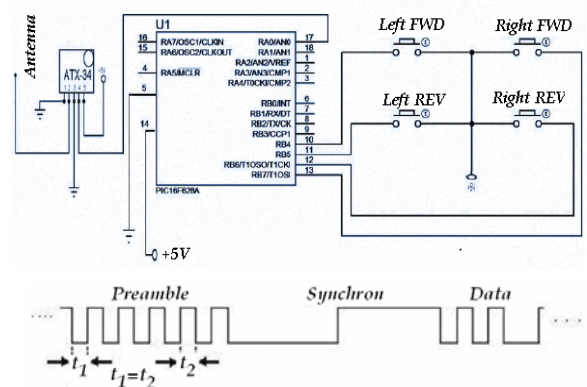
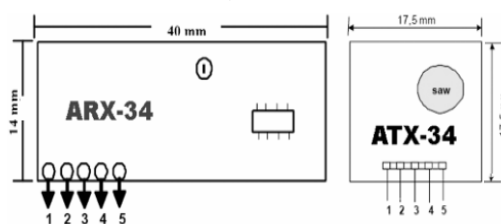
ADC_READ:
ADCIN VERI, BILGI
RETURN
    
```

## RGB VALUES

COLOR	RED	GREEN	BLUE
GLOSSY RED	3.14 V	1.23 V	1.11 V
DARK RED	2.99 V	1.45 V	1.36 V
DARK MATTE GREEN	2.24 V	2.89 V	3.11 V
LIGHT GLOSSY GREEN	1.27 V	3.78 V	3.37 V
MATTE GREEN	1.51 V	3.44 V	2.39 V
LIGHT MATTE BLUE	1.11 V	2.84 V	3.24 V
MATTE BLUE	1.23 V	2.87 V	3.17 V
DARK GLOSSY BLUE	2.22 V	2.57 V	3.36 V

## APPLICATION 2 RF CONTROLLED MOTOR

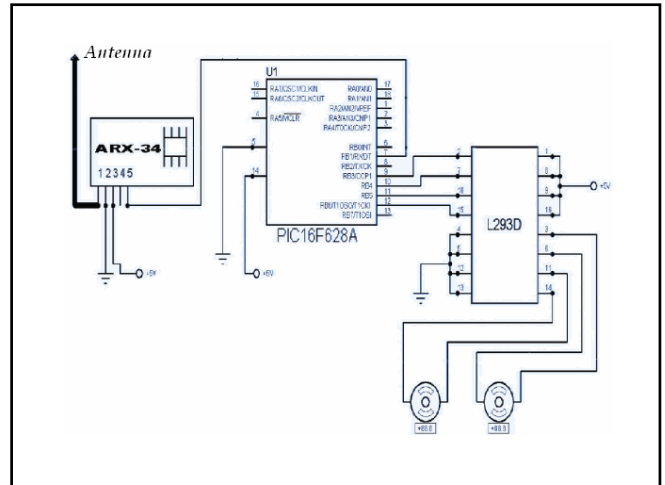
433,92 MHZ



```

CMCON=07 'PortA
INCLUDE "modedefs.bas"
OPTION_REG.7 = 1 'PortB Pull_Up Active
TRISB = %11110000
TRISA = %00000000
K VAR BYTE
K=0
PAUSE 500
Serout2 PORTA,0,16780,[REP$AA\5,REP$00\REP$FF\5]
; preamble + Synchron Sending
MAINLOOP:
IF PORTB.4=1 then K.BIT0=1
IF PORTB.5=1 then K.BIT1=1
IF PORTB.6=1 then K.BIT2=1
IF PORTB.7=1 then K.BIT3=1
SEROUT PORTB.7,N2400, [254]
SEROUT PORTB.7,N2400, [K]
SEROUT PORTB.7,N2400, [192]
PAUSE 16
K=0
GOTO MAINLOOP

```



```

CMCON=07 'PortA
TRISB = %00000010

DEFINE HSER_RCSTA 90h
DEFINE HSER_TXSTA 20h
DEFINE HSER_BAUD 2400
DEFINE HSER_CLROERR 1

LEFTFWD VAR PORTB.3
LEFTREV VAR PORTB.4
RIGHTFWD VAR PORTB.5
RIGHTREV VAR PORTB.6
K VAR BYTE
ERRCHK VAR BYTE
PAUSE 250

MAINLOOP:
HSERIN [WAIT(254),K,ERRCHK]
LEFTREV=0; RIGHTREV=0;
RIGHTFWD=0; LEFTFWD=0;

```

```

IF ERRCHK=192
THEN
IF K.BIT0=1 THEN
LEFTFWD=1
ELSE
LEFTFWD=0
ENDIF
ENDIF

IF K.BIT1=1 THEN
LEFTREV=1
ELSE
LEFTREV=0
ENDIF

IF K.BIT2=1 THEN
RIGHTFWD=1
ELSE
RIGHTFWD=0
ENDIF

IF K.BIT3=1 THEN
RIGHTREV=1
ELSE
RIGHTREV=0
ENDIF
ENDIF

PAUSE 10
GOTO ANADONGU

```

## REFERENCES

- ❑ [www.microchip.com](http://www.microchip.com) (Official website of the PIC manufacturer, PIC16F877 datasheet & some application notes are available)
- ❑ <http://www.microchip.com/ParamChartSearch/chart.aspx?branchID=1002&mid=10&lang=en&pagelD=74>
- ❑ [http://en.wikipedia.org/wiki/PIC\\_microcontroller](http://en.wikipedia.org/wiki/PIC_microcontroller)
- ❑ <http://www.jpixton.dircon.co.uk/pic/>
- ❑ <http://www.oopic.com/>
- ❑ [www.antrak.org](http://www.antrak.org) (Ankara amateur radio society website)
- ❑ [www.eproje.com](http://www.eproje.com) (Some applications in Turkish)
- ❑ [www.picproje.net](http://www.picproje.net) (A discussion forum on PIC in Turkish)
- ❑ [www.elektroda.pl](http://www.elektroda.pl) (A discussion forum on PIC)
- ❑ [robot.metu.edu.tr](http://robot.metu.edu.tr) (METU Robot Society website)
- ❑ <http://www.engr.mun.ca/~msimms/pic/> (code archive)
- ❑ <http://www.workingtex.com/htpic/> (examples)

## REFERENCES-2

- ❑ <http://www.olimex.com/dev/pic-pg2.html>
- ❑ [http://sourceforge.net/project/download.php?group\\_id=599&use\\_mirror=heanet&filename=sdcc-2.6.0-setup.exe&40589420](http://sourceforge.net/project/download.php?group_id=599&use_mirror=heanet&filename=sdcc-2.6.0-setup.exe&40589420)
- ❑ <http://robot.metu.edu.tr/index.php?link=5>
- ❑ <http://www.tomshardware.com.tr/howto/20050909/index.html>

## C and bootloader links

- CCS PIC C Compiler
  - <http://www.ccsinfo.com/>
- CCS PIC C Compiler Manual
  - <http://www.ccsinfo.com/piccmanual3.zip>
- Bootloader Code (that resides on the PIC)
  - [http://www.workingtex.com/htpic/PIC16F87x\\_and\\_PIC16F7x\\_bootloader\\_v7-40.zip](http://www.workingtex.com/htpic/PIC16F87x_and_PIC16F7x_bootloader_v7-40.zip)
- Bootloader Program (that resides on the PC)
  - <http://www.ehl.cz/pic/>