8. Microcontroller

- Textbook

  "Programming Robot Controllers", Myke Predko, McGraw Hill.

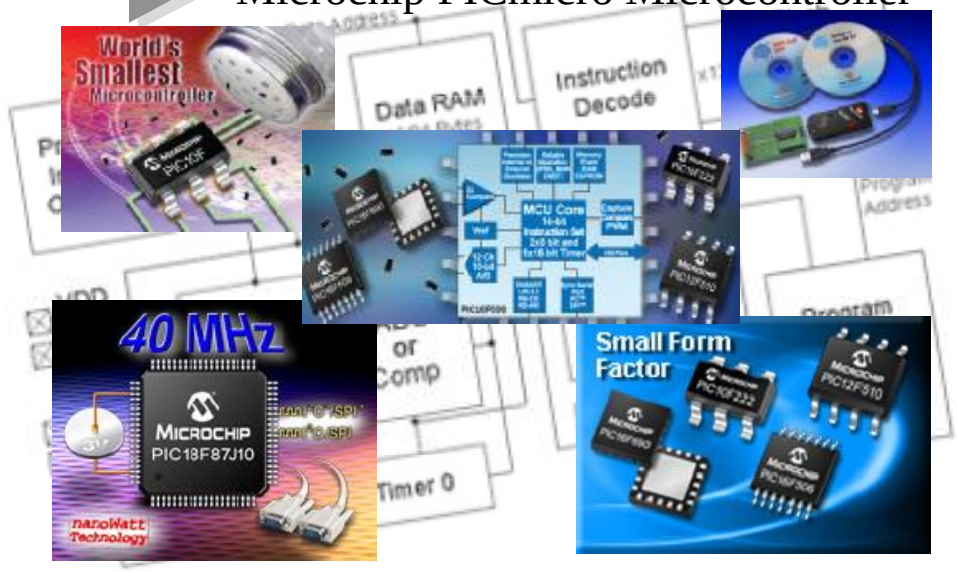
- Reference

  "PIC Robotics: A Beginner's Guide to Robotics Projects Using the PIC Micro", John Iovine, McGraw Hill.

  "Embedded C Programming and the Microchip PIC", Richard H. Barnet, Delmar Learning.

  "PIC Microcontroller: An Introduction to Software & Hardware Interfacing", Han-Way Huang, Delmar Learning.

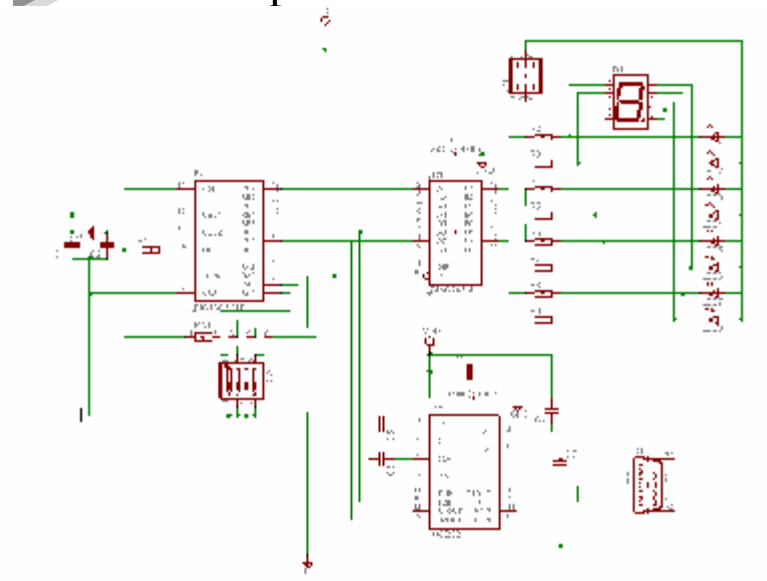
Microchip PICmicro Microcontroller



Microchip PICmicro Microcontroller

- Also known as PIC
- Advantages:
  - Easy to find parts from numerous distributors
  - Many version of PIC – function, size, package, I/O
- Free IDE tools
  - e.g. MPLAB
- Free C compiler
  - mikroElektronika mikroC (limited to 2K program)
  - HI-TECH PICC Lite (limited to 16F84 and 16F627)
  - Microchip / Burnon MPLAB C18 (time limited)

PIC Experimental Board

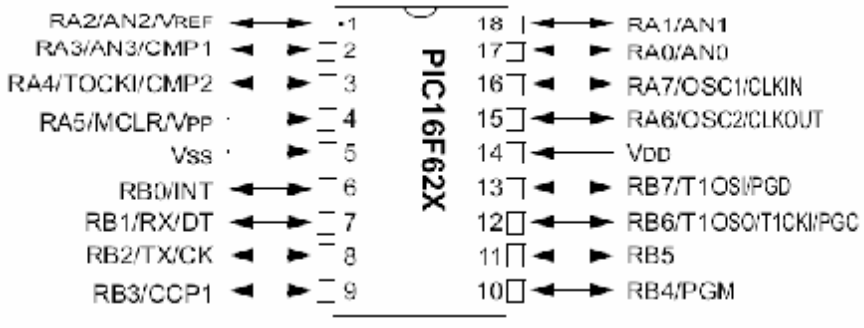


Basic Circuit Requirements

- Stable +5V supply
- Reset line input ($\overline{\text{MCLR}}$)
- Clock
  - Internal
  - Resistor/capacitor (RC)
  - Crystal oscillator

PIC 16F627

- Pin assignment



PIC16F62X PINOUT DESCRIPTION

RA0 / AN0	RA0 Bi-directional I / O port	AN0 Analog comparator input
RA1 / AN1	RA1 Bi-directional I / O port	AN1 Analog comparator input
RA2 / AN2 / VREF	RA2 Bi-directional I / O port VREF – AN VREF output	AN2 Analog comparator input
RA3 / AN3 / CMP1	RA3 Bi-directional I / O port CMP1 – Comparator 1 output	AN3 Analog comparator input
RA4 / T0CKI / CMP2	RA4 Bi-directional I / O port CMP2 – OD Comparator 2 output	T0CKI – Timer0 clock input
RA5 / MCLR / VPP	RA5 – Input port	MCLR – Master clear

VPP – Programming voltage input. When configured as MCLR, this pin is an active low RESET to the device. Voltage on MCLR / VPP must not exceed VDD during normal device operation.

RA6 / OSC2 / CLKOUT RA6 Bi-directional I / O port

OSC2 XTAL – Oscillator crystal output. Connects to crystal or resonator in Crystal Oscillator mode.

CLKOUT– In ER / INTRC mode, OSC2 pin can output CLKOUT, which has 1 / 4 the frequency of OSC1

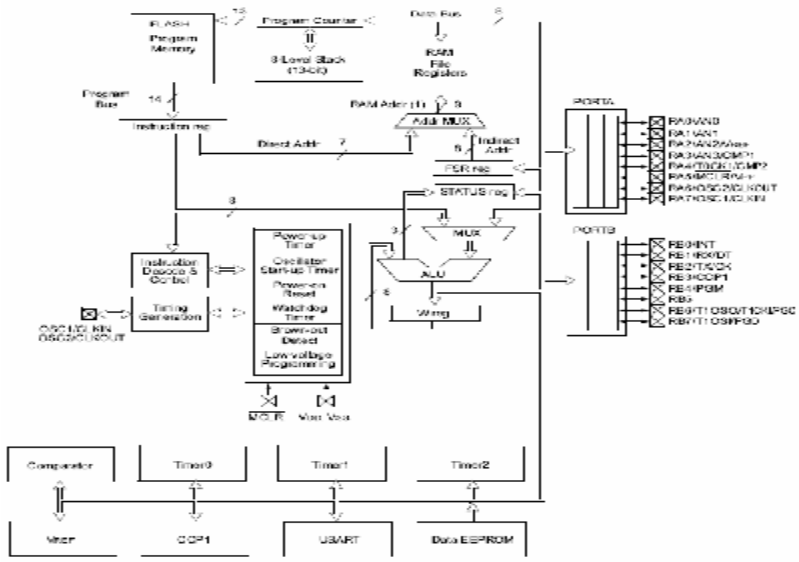
RA7 / OSC1 / CLKIN RA7 Bi-directional I / O port

OSC1 XTAL – Oscillator crystal input

CLKIN – External clock source input. ER biasing pin.

RB0 / INT	RB0 Bi-directional I / O port.	INT – External interrupt.
RB1 / RX / DT	RB1 Bi-directional I / O port. DT Synchronous data I / O.	RX – USART receive pin
RB2 / TX / CK	RB2 Bi-directional I / O port. CK Synchronous clock I / O.	TX – USART transmit pin
RB3 / CCP1	RB3 Bi-directional I / O port.	CCP1 Capture / Compare / PWM I / O
RB4 / PGM	RB4 Bi-directional I / O port. PGM Low voltage programming input pin.	
RB5	RB5 Bi-directional I / O port.	
RB6 / T1OSO / T1CKI / PGC	RB6 Bi-directional I / O port. T1OSO – XTAL Timer1 oscillator output. T1CKI ST – Timer1 clock input. PGC ST – ICSP™ Programming Clock.	
RB7 / T1OSI / PGD	RB7 Bi-directional I / O port. T1OSI XTAL – Timer1 oscillator input. Wake-up from SLEEP on pin change. PGD ST CMOS ICSP Data I / O	

Block Diagram



Block diagram

The PIC16F62X devices contain an 8-bit ALU and working register. The ALU is a general purpose arithmetic unit. It performs arithmetic and Boolean functions between data in the working register and any register file. Depending on the instruction executed, the ALU may affect the values of the Carry (C), Digit Carry (DC), and Zero (Z) bits in the STATUS register. The C and DC bits operate as a Borrow and Digit Borrow out bit, respectively, bit in subtraction.

Two types of data memory are provided on the PIC16F62X devices. Non-volatile EEPROM data memory is provided for long term storage of data such as calibration values, lookup table data, and any other data which may require periodic updating in the field. This data is not lost when power is removed. The other data memory provided is regular RAM data memory. Regular RAM data memory is provided for temporary storage of data during normal operation. It is lost when power is removed.

Configuration Register

- Give the application developer flexibility in how the PIC are used in application.
- If register are not set correctly, PIC will not run properly.
- Register is written when the PIC is being programmed.
- Register content is accessed in PIC bootup process to select the hardware options required for application.

Configuration register define for: ***[details refer to textbook Table3-5]***

- Code / data protection (CP / CPD) - normally set "OFF"
- Low voltage programming (LVP) - normally set "disable"
- Brown-out detect (BODEN) - normally set "enable"
- Reset parameters (MCLR) – use _MCLR as reset pin
- Power-up timer - normally set "enable"
- Watchdog timer (WDT) - normally set "disable"
- Oscillator mode used (FOSC0-2) – normally set "HS" (high speed)

- For mikroC, default setting is:
    - Oscillator mode = HS
    - WDT = disable
    - LVP = disable

Input/Output (I/O) Registers

- TRISx (where x = A or B)
  - To control the input/output of the I/O pin
  - Register load with "1", I/O pin set as input
  - Register load with "0", I/O pin set as output
  - e.g. TRISA.F0 = 1 will set port A pin 0 (i.e. RA0) as input
  - TRISB.F7 = 0 will set port B pin 7 (i.e. RB7) as output
- PORTx (where x = A or B)
  - Directly store data sending out or received from I/O pin
  - e.g. PORTA.F0 = 1 will send high to port A pin 0 (RA0)
  - temp = PORTB.F3 will assign data received from port B pin 3 (RB3) to temp

I/O convention for mikroC and PICC-Lite:

mikroC	PICC-Lite
TRISA	TRISA
TRISA.F0	TRISA0
PORTB	PORTB or RB
PORTB.F0	PORTB.0 or PORTB0 or RB0

Input/Output Registers

- How to make RB2 I/O pin a digital output and drive a high value?

```
TRISB.F2 = 0;        // RB2 set as output pin
PORTB.F2 = 1;        // RB2 drives a "high" value out
```

- How to make RA0 I/O pin a digital output and drive a low value?

```
TRISA.F0 = 0;        // RA0 set as output pin
PORTA.F0 = 0;        // RA0 drives a "low" value out
```

- How to make RA7 I/O pin a digital input and receive a data?

```
TRIS_.F_ = ___ ;     // RA7 set as input pin
temp = PORT_.F_;     // Receive from RA7 and assign to temp
```

Input/Output Registers

- How to drive entire Port B to high values?

  TRISB = 0;
  PORTB = 0xFF;

- How to make RA I/O pin digital inputs and store the received data to temp?

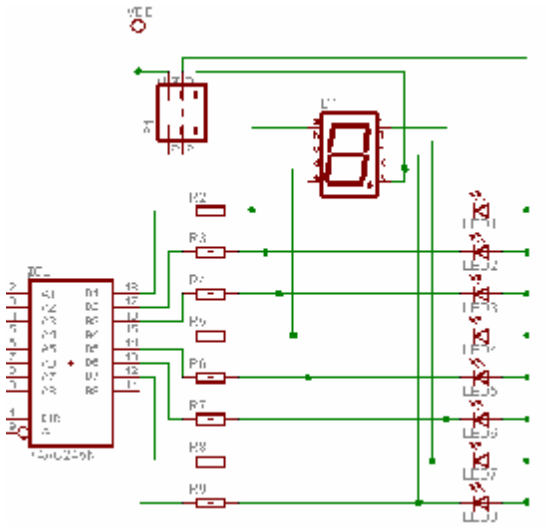
  TRISA = 0x___;
  temp = _____;

Workshop 1 – mikroC startup and burn PIC

- LED flashing

```
void main() {
  PORTB = 0;
  TRISB = 0;

  while(1) {
    PORTB =  ~PORTB;
    Delay_ms(1000);
  }
}
```



Workshop 2 - PIC input and output

```
void main()
{
  PORTB = 0;
  TRISB = 0;
  CMCON = 0x07;    /* enable digital I/O for PORTA */
  TRISA.F0 = 1;    /* set RA0 as input */

  while(1)
  {
    if (PORTA.F0)
        PORTB = 0xFF;
    else
        PORTB = 0;
    Delay_ms(1000);    /* set 1000ms delay    */
  }
}
```

7-segment LED display

Char	a	b	c	d	e	f	g	dp	value
0	0	0	0	0	0	0	1	0	02
1	1	0	0	1	1	1	1	0	9e
2	0	0	1	0	0	1	0	0	24
3	0	0	0	0	1	1	0	0	0c
4	1	0	0	1	1	0	0	0	98
5	0	1	0	0	1	0	0	0	48
6	0	1	0	0	0	0	0	0	40
7	0	0	0	1	1	1	1	0	1e
8	0	0	0	0	0	0	0	0	00
9	0	0	0	0	1	0	0	0	08

Workshop 3 - PIC output to 7-segment display

```
void main() {
  PORTB = 0;
  TRISB = 0;
  while(1)   {
    PORTB = 0x02;        /*0*/
    Delay_ms(1000);
    PORTB = 0x9e;        /*1*/
    Delay_ms(1000);
    PORTB = 0x24;        /*2*/
    Delay_ms(1000);
    PORTB = 0x0c;        /*3*/
    Delay_ms(1000);
    PORTB = 0x98;        /*4*/
    Delay_ms(1000);
    PORTB = 0x48;        /*5*/
    Delay_ms(1000);
    PORTB = 0x40;        /*6*/
    Delay_ms(1000);
    PORTB = 0x1e;        /*7*/
    Delay_ms(1000);
    PORTB = 0x00;        /*8*/
    Delay_ms(1000);
    PORTB = 0x08;        /*9*/
    Delay_ms(1000);
  }
}
```

Workshop 4 - PIC interactive I/O

```
void main() {
int  seg7[10] = {0x02, 0x9e, 0x24, 0x0c, 0x98, 0x48, 0x40, 0x1e, 0x00,
                 0x08};    /* 0 to 9*/
int  index = 0;     /* for store the current display digit */
  PORTB = seg7[0];
  TRISB = 0;
  CMCON = 0x07;    /* enable digital I/O for PORTA */
  TRISA.F0 = 1;
  TRISA.F1 = 1;
  while (1)  {
        if (PORTA.F0 == 0)   /* when RA0 is pressed */
        {
              index = index + 1;
              if (index == 10)
                 index = 0;
              PORTB = seg7[index];
              Delay_ms(1000);
        }
  }
}
```

Workshop 5 - PIC input decode

```
void main() {
int  seg7[10] = {0x02, 0x9e, 0x24, 0x0c, 0x98, 0x48, 0x40, 0x1e, 0x00, 0x08}; /* 0 to 9*/
int  index = 0;     /* for store the current display digit */
  PORTB = seg7[0];
  TRISB = 0;
  CMCON = 0x07;    /* enable digital I/O for PORTA */
  TRISA.F0 = 1;    /* set all 4 inputs for PORTA   */
  TRISA.F1 = 1;
  TRISA.F2 = 1;
  TRISA.F3 = 1;
  while (1)   {
        index = 0;
        if (PORTA.F0 == 1)    index = index + 1;  /* calculate weighting */
        if (PORTA.F1 == 1)    index = index + 2;
        if (PORTA.F2 == 1)    index = index + 4;
        if (PORTA.F3 == 1)    index = index + 8;
        if (index > 9)        /* Display 0 to 9   */
              index = 0;
        PORTB = seg7[index];
        Delay_ms(1000);
  }
}
```

Music Frequency

Tone	Frequency	Period / 2
do	1046.5	478
re	1174.7	426
me	1318.5	379
fa	1396.9	358
so	1568	319
la	1760	284
ti	1975.5	253

Workshop 6 - PIC output to speaker

```
void main() {
  int       i;
  PORTB = 0xff;
  TRISB = 0;
  CMCON = 0x07;    /* enable digital I/O for PORTA */
  TRISA.F0 = 1;          TRISA.F1 = 1;          TRISA.F2 = 1;          TRISA.F3 = 1;
  while (1)
  {
      if (PORTA.F0 == 0) {
        for (i=0; i<500; i++) {
           PORTB.F0 = 1;
           Delay_us(478);   /* do */
           PORTB.F0 = 0;
           Delay_us(478);
        }
      }
   }
}
```

Interrupt setting

```
void main() {
  TRISB = 0x01;      /* RB0 as interrupt */
  INTCON.GIE = 1;    /* enable global interrupt */
  INTCON.INTE = 1;   /* enable RB0/INT pin interrupt */
  while(1) {
  }
}

void interrupt () {
  INTCON.INTE = 0;     /* disable interrupt–avoid interrupt again */
  if (INTCON.INTF) {   /* check interrupt from RB0   */
     INTCON.INTF = 0; /* reset interrupt flag */
  }
  INTCON.INTE = 1;     /* enable interrupt again     */
}
```

Workshop 7 - PIC interrupt

```
void interrupt () {
int  i, j;
  INTCON.INTE = 0;
  if (INTCON.INTF) {
       PORTB = 0x55;
       Delay_ms(1000);
       PORTB = 0xaa;
       Delay_ms(1000);
       PORTB = 0x55;
       Delay_ms(1000);
       PORTB = 0xaa;
       Delay_ms(1000);
       INTCON.INTF = 0;     /* reset interrupt flag */
  }
  PORTB = 0;         /* reset pattern */
  INTCON.INTE = 1;
}

void main() {
  PORTB = 0;
  TRISB = 0x01;      /* RB0 as interrupt */
  INTCON.GIE = 1;    /* enable global interrupt */
  INTCON.INTE = 1;   /* enable RB0/INT pin interrupt */

  while(1) {
    PORTB = ~PORTB;
    Delay_ms(1000);
  }
}
```

Baud Rate Generator

- In Asynchronous mode bit Baud Rate Generator controls the baud rate.
- Given the desired baud rate and Fosc, the nearest integer value for the SPBRG register can be calculated using the following formula:

When BRGH = 0, SYNC = 0,

Desired Baud rate = Fosc / (64 (X + 1))

Fosc = 16 MHz, Desired Baud Rate = 9600

9600 = 16000000 / (64(X+1))

X = 25.042

Calculated Baud Rate = 16000000 / (64(25 + 1)) = 9615

Error = (Calculated Baud Rate - Desired Baud Rate) / Desired Baud Rate

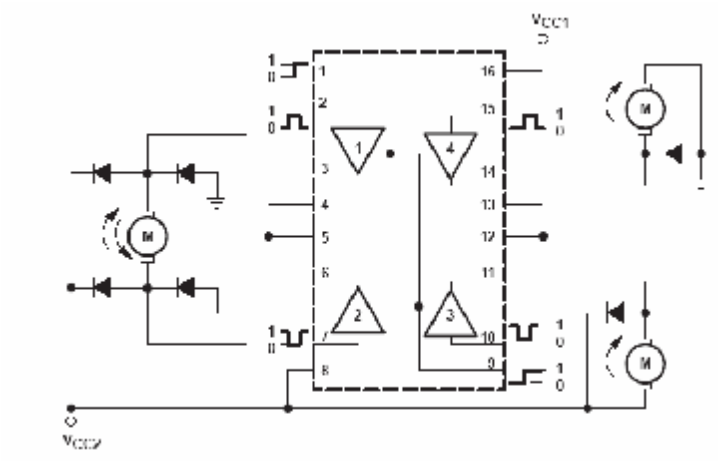
= (9615 - 9600)/ 9600

= 0.16%

Workshop 8 - Serial Comm.

```
void main() {
unsigned char temp;
     OPTION_REG = 0xd1; /* assign prescalar to TMR0 */
     TMR0 = 0;        /* reset timer for start   */
     INTCON.T0IE = 1; /* enable timer interrupt  */
     INTCON.GIE = 1;  /* enable global interrupt */
     SPBRG = 12;      /* 51 : baud rate = 1200 @ 4MHz */
                      /* 12 : baud rate = 4800        */
     TXSTA.TXEN = 1;  /* enable usart            */
     RCSTA.CREN = 1;  /* enable continuous receive*/
     RCSTA.SPEN = 1;  /* serial port enable      */
     PIR1.RCIF = 0;   /* clear receive flag    */
     while (1) {
         if (PIR1.RCIF) {    /* wait char received  */
            temp = RCREG;    /* get char            */
            TXREG = temp -1; /* modify char        */
            PIR1.RCIF = 0;   /* clear flag         */
         }
     }
}
```

L293 Motor Driver



Workshop 9 - Motor control

```
void main() {
  PORTB = 0;
  TRISB.F5 = 0;    /* motor enable    */
  TRISB.F6 = 0;    /* motor direction */
  TRISB.F7 = 0;    /* RB6, RB7 = 0,0 or 1,1 = no motion  */
                   /* 1,0 one direction    */
                   /* 0,1 other direction  */
  CMCON = 0x07;    /* enable digital I/O for PORTA */
  TRISA.F0 = 1;
  TRISA.F1 = 1;
  PORTB.F5 = 1;    /* enable motor   */
    while (1)  {
        if (PORTA.F0 == 0) {   /* RA0 press */
              PORTB.F6 = 1;
              PORTB.F7 = 0;
              Delay_ms(1000);
        }
        if (PORTA.F1 == 0) {   /* RA1 press */
              PORTB.F6 = 0;
              PORTB.F7 = 1;
              Delay_ms(1000);
        }
        PORTB.F6 = 0;
        PORTB.F7 = 0;
  }
```