1. Preface

Embedded programs included in this book were created based on the bootloader Bel8. Bootloader is called boot loader in embedded systems. The boot is a small program to run after power up or reset the system. This program leads the hardware environment of the system to appropriate state to prepare a good fit for the final calling to the application. The loader copied the software components to RAM and ultimately guides to running the applications.

An embedded system is a dedicated computer system designed for one or two specific functions. This system is embedded as a part of a complete device system that includes hardware, such as electrical and mechanical components. The embedded system is unlike the general-purpose computer, which is engineered to manage a wide range of processing tasks. An embedded system is a computer system with a dedicated function within a larger mechanical or electrical system, often with real-time computing constraints. An embedded system is an application that contains at least one programmable computer, typically in the form of a microcontroller. Embedded microcontrollers are everywhere today. In the average household you will find them far beyond the obvious places like cell phones, calculators, and MP3 players. Hardly any new appliance arrives in the home without at least one controller and, most likely, there will be several—one microcontroller for the user interface (buttons and display), another to control the motor, and perhaps even an overall system manager. This applies whether the appliance in question is a washing machine, garage door opener, curling iron, or toothbrush. If the product uses a rechargeable battery, modern high density battery chemistries require intelligent chargers. The PIC is a widely used microcontroller. This book is intended for use in college level courses teaching microcontrollers and embedded systems. It not only establishes a foundation of C language programming, but also provides a comprehensive treatment of PIC interfacing for engineering students. From this background, the design and interfacing of microcontroller based embedded systems can be explored. This book can also be used by practicing technicians, hardware engineers and computer scientist.
2. Bootloader Bel8

Bootloader operate on PIC18F46K80 microcontroller. The firmware communicate via UART1 on the PIC device using UART/USB Converter Prolific PL 2003.

Flash microcontroller enable firmware to program itself. This is done by a “bootloader” providing a firmware kernel, residing in the microcontroller. The kernel uses a small portion of program memory not normally used by the firmware’s main application. When the bootloader firmware is activated, a host PC can use a serial protocol to read, write and verify updates to the microcontroller's application firmware. Once the application firmware is programmed, the bootloader cedes control, allowing normal application execution until the bootloader is called. Small firmware code size (less than 450 instruction words on PIC microcontroller. The key features of the Bootloader BEL8 include:

• Automatic baud rate synchronization to the host
• Baud rate flexibility, from 1,200 bps to 3 Mbps for extremely fast programming
• A 16-bit CRC packet and Flash memory verification for quick verification of successful programming, even at low baud rates
• An advanced “write planner” that eliminates unnecessary erase/write transactions
• Optional application remapping that does not require linker script modifications
or remapping of interrupt service routines
• A forced bootloader re-entry mechanism requiring minimal start-up delay and no additional I/O pins or application firmware code to re-enter the bootloader
• Optional MCLR Reset control, allowing the host PC application to automatically reset the device for robust bootloader re-entry

2.1. Bootloader Firmware Operation

Data is received through the USART module and quickly stored to the RAM buffer to avoid overrun errors. After each packet is received, the integrity of the data is verified using a 16-bit CRC. Essential operating design of the bootloader summarizes Figure 2.

![Bootloader Bel8 operating design diagram](image)

Fig.2. Bootloader Bel8 operating design diagram

When a valid request packet has been received, the command interpreter evaluates the command number in the packet to determine what operation needs
to be done, such as: Erase, Write, Read or Verify. The request is fulfilled and a response is returned through the USART to either Acknowledge completion of the task, or on Read operations, to send back device memory data.

In the default configuration, bootloader firmware is stored at the end of Flash program memory space. Keeping the bootloader at the end of program memory space allows application firmware to handle interrupts at the normal hardware interrupt vector address. This configuration keeps interrupt latency to a minimum. The host PC bootloader application will write a “GOTO” as the first instruction at the Reset vector (address, 0000h). This GOTO jumps to the bootloader at start-up. The application firmware’s original Reset vector instruction is automatically moved to reside just before the bootloader block of memory. The bootloader firmware uses this application Reset vector to start the application when the Bootloader mode is not requested, see Figure 3.

![Fig.3. Bootloader Bel8 program Memory](image)

On PIC18F46k80 devices, this automatic relocation of the Reset vector requires the application firmware to provide a single GOTO instruction as the first instruction at address, 0000h.
The PCLATH register needs to be loaded before executing a far GOTO. The host PC bootloader application will write up to four instructions at the Reset vector (address, 0000h) to jump to the bootloader at start-up. To make room, the first four application firmware instructions are automatically moved by the host PC software to reside just before the bootloader firmware.

3. Operation PIC18F46K80 microcontroller

Microchip Inc. has developed the PIC18 series of microcontrollers for use in high-pin-count, high-density, and complex applications. The PIC18 microcontrollers offer cost-efficient solutions for general purpose applications written in C that use a Real Time Operating System RTOS and require a complex communication protocol stack such as TCP/IP, CAN, USB, or ZigBee. PIC18F devices provide flash program memory in sizes from 8 to 128Kbytes and data memory from 256 to 4Kbytes, operating at a range of 3.3 to 5.0 volts, at speeds from DC to 40MHz.

RISC and CISC core

RISC Reduced Instruction Set Computer and CISC Complex Instruction Computer refer to the instruction set of a microcontroller. In an 8-bit RISC microcontroller, data is 8 bits wide but the instruction words are more than 8 bits wide, usually 12, 14, or 16 bits and the instructions occupy one word in the program memory. Thus the instructions are fetched and executed in one cycle, which improves performance. In a CISC microcontroller, both data and instructions are 8 bits wide. CISC microcontrollers usually have over two hundred instructions. Data and code are on the same bus and cannot be fetched simultaneously. PIC18F46k80 applied in bootloader has technical features:

- Operating Voltage Range: 1.8V to 5.5V
- On-Chip 3.3V Regulator
- Operating Speed up to 64 MHz
- Up to 64 Kbytes On-Chip Flash Program Memory:
  - 10,000 erase/write cycle, typical
  - 20 years minimum retention, typical
- 1,024 Bytes of Data EEPROM:
  - 100,000 Erase/write cycle data EEPROM memory, typical
- 3.6 Kbytes of General Purpose Registers (SRAM)
- Three Internal Oscillators: LF-INTOSC (31 KHz), MF-INTOSC (500 kHz) and HF-INTOSC (16 MHz)
- Self-Programmable under Software Control
- Priority Levels for Interrupts
- 8 x 8 Single-Cycle Hardware Multiplier
- Extended Watchdog Timer (WDT):
- Programmable period from 4 ms to 4,194s
  • In-Circuit Serial Programming™ (ICSP™) via Two Pins
  • In-Circuit Debug via Two Pins
  • Programmable BOR
  • Programmable LVD

Microcontroller is 44-Lead Plastic ThinQuadFlatpack (PT) - [TQFP] and topology is shown on fig. 4.

Figure 5 shows the architecture of the PIC18F46k80 microcontroller. The CPU is at the center of the diagram and consists of an 8-bit ALU, an 8-bit working accumulator register WREG, and an $8 \times 8$ hardware multiplier. The higher byte and the lower byte of a multiplication are stored in two 8-bit registers called PRODH and PRODL respectively.
Rys. 5. Mikrokontroller PIC18F46K80 architektura

The program counter and program memory are shown in the upper left portion of the diagram. Program memory addresses consist of 21 bits, capable of accessing 2Mbytes of program memory locations. The PIC18F452 has only
32Kbytes of program memory, which requires only 15 bits. The remaining 6 address bits are redundant and not used. A table pointer provides access to tables and to the data stored in program memory. The program memory contains a 31-level stack which is normally used to store the interrupt and subroutine return addresses. The data memory can be seen at the top center of the diagram. The data memory bus is 12 bits wide, capable of accessing 4Kbytes of data memory locations. As we shall see later, the data memory consists of special function registers (SFR) and general purpose registers, all organized in banks.

3.1. Operation of PORT and LATCH on PIC18F

Output pins of a PIC Microcontroller is divided in to different PORTS containing a group of GPIO (General Purpose Input Output Pins) pins. In 16F PIC Microcontrollers, there are two registers associated with a port, TRIS and PORT. eg: TRISB, PORTB, TRISC, PORTC. TRIS stands for Tri-State, which determines the direction of each GPIO pin. Logic 1 at a particular bit of TRIS register makes the corresponding pin Input and Logic 0 at a particular bit of TRIS register makes the corresponding pin Output. An Input pin of PIC18 Microcontroller have very high input impedance, thus it may said to be in Hi-Impedance state. PORT register is used to read data from or write data to GPIO pins. Logic 1 at a particular bit of PORT register makes the corresponding pin at Logic High (VDD) and Logic 0 at a particular bit of PORT register makes the corresponding pin at Logic Low (VSS) if that pin is an Output pin (TRIS bit is 0). PORT register can be used to read digital data from an Input pin. Logic 1 indicates the pin is at Logic High and Logic 0 indicates that the pin is at Logic Low.
Fig 4. PORT, TRIS Register PIC Microcontroller.

Difference between PORT and LATCH on PIC18F is shown on schematic diagram. Each port has three registers for its operation. These registers are:

- TRIS register (data direction register)
- PORT register (reads the levels on the pins of the device)
- LAT register (output latch)

So in most situations, you will write to the latch and read from the port.
The LAT\textsubscript{x} register associated with an I/O pin eliminates the problems that could occur with read-modify-write instructions. A read of the LAT\textsubscript{x} register returns the values held in the port output latches, instead of the values on the I/O pins. A read-modify-write operation on the LAT register, associated with an I/O port, avoids the possibility of writing the input pin values into the port latches. A write to the LAT\textsubscript{x} register has the same effect as a write to the PORT\textsubscript{x} register. The differences between the PORT and LAT registers can be summarized as follows:

- Write to the PORT\textsubscript{x} register writes the data value to the port latch.
- Write to the LAT\textsubscript{x} register writes the data value to the port latch.
- Read of the PORT\textsubscript{x} register reads the data value on the I/O pin.

Fig. 5 Generic I/O Port Operation
Read of the LATx register reads the data value held in the port latch. When you write a bit in a I/O pin, you're storing this bit from Data Bus to the Data Register D-FlipFlop. If TRISx of this bit is 0, so data from Q of the Data Register will be in the I/O pin. Write in LATx or PORTx is the same. See below in Red, on fig.6.

Fig.6 Write in LATx or PORTx

On the other hand, read from LATx is different of read from PORTx. When you're reading from LATx, you're reading what is in the Data Register D-FlipFlop. See picture below in green:
Fig. 7. Read from LATx is reading what is in the Data Register D-FlipFlop

And when you read from PORTx, you're reading the actual I/O pin value. See below in blue:
PIC uses read-modify-write to write operations and this can be a problem, so they use this shadow register to avoid it. Every port has its corresponding TRIS register: TRISA, TRISB, TRISC etc which determines performance, but not the contents of the port bits. By clearing some bit of the TRIS register (bit=0), the corresponding port pin is configured as output. Similarly, by setting some bit of the TRIS register (bit=1), the corresponding port pin is configured as input. This rule is easy to remember 0 = Output, 1 = Input. You can write to PORT and TRIS Registers entirely or bit by bit.

**Writing Bit by Bit:**
- TRISC0 = 1;  //Makes 0th bit of PORTC Input
- TRISC5 = 0;  //Makes 5th bit of PORTC Output
- RB3 = 1;  //Makes 3rd bit of PORTB at Logic High
- RB7 = 0;  //Makes 7th bit of PORTB at Logic Low
Writing Entire Register
You should be familiar with following C Programming concepts.

- A number with a prefix ‘0b’ indicates a binary number.
- A number with a prefix ‘0’ indicates an octal number.
- A number with a prefix ‘0x’ indicates a hexadecimal number.
- A number without prefix is a decimal number.

PORTB = 0xFF; //Makes all pins of PORTB Logic High
TRISC = 0x00; //Makes all pins of TRISC Output
PORTD = 128; //Makes 7th bit of PORTD Logic High

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Binary</th>
<th>Octal</th>
<th>Hexadecimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0b00000000</td>
<td>00</td>
<td>0x00</td>
</tr>
<tr>
<td>1</td>
<td>0b00000001</td>
<td>01</td>
<td>0x01</td>
</tr>
<tr>
<td>128</td>
<td>0b10000000</td>
<td>0200</td>
<td>0x80</td>
</tr>
<tr>
<td>255</td>
<td>0b11111111</td>
<td>0377</td>
<td>0xFF</td>
</tr>
</tbody>
</table>

PORTB = 0xFF; //Makes all pins of PORTB Logic High
TRISC = 0x00; //Makes all pins of TRISC Output
PORTD = 128; //Makes 7th bit of PORTD Logic High

In program: All Port Test I look as:

```c
// All ports as outputs
TRISA=0;
TRISB=0;
TRISC=0;
TRISD=0;
TRISE=0;

// All Line Low
LATA=0x00;
LATB=0x00;
LATC=0x00;
LATD=0x00;
LATE=0x00;
```

(......)
3.2. Program listing All_Port Test for bootLoader Bel8 PIC18F46K80

```c
#include <p18lf46k80.h>
#pragma config  RETEN = ON,INTOSCSEL = LOW, SOSCSEL = DIG, XINST = OFF //CONFIG1L
#pragma config  FOSC = INTIO2, PLLCFG = ON, FCMEN = OFF, IESO = ON //CONFIG1H
#pragma config PWRTEN = OFF, BOREN = OFF , BORV = 3, BORPWR = ZPBORMV // CONFIG2L
#pragma config WDTEN = OFF, WDTPS = 128 // CONFIG2H
#pragma config MCLRE = ON, MSSPMSK = MSK7, CANMX = PORTC // CONFIG3H
#pragma config BBSIZ = BB1K, STVREN = ON // CONFIG4L
#pragma config CP0 = OFF, CP1 = OFF // CONFIG5L
#pragma config CPB = OFF, CPD = OFF // CONFIG5H
#pragma config WRT0 = OFF, WRT1 = OFF, WRT2 = OFF, WRT3 = OFF // CONFIG6L
#pragma config WRTB = OFF, WRTC = OFF, WRTD = OFF // CONFIG6H
#pragma config EBTR0 = OFF, EBTR1 = OFF, EBTR2 = OFF, EBTR3 = OFF // CONFIG7L
#pragma config EBTRB = OFF // CONFIG7H
int i,j;
```
// Main code section. Execution starts here.

void main(void)
{
    //unsigned int
    OSCCONbits.IRCF=0b000; //internal oscillator frequency 31kHz
    INTCON2bits.NOT_RBPU=0; //enable internal pullups resisors on PORTB
    WPUB=0b00000010; //ENABLE SELECABLE WEEK PUULUPS ON RB1 PIN
    ANCON1=0;
    ANCON0=0; // All pins digital
    TRISD=0; //PORTD as output;
    //TRISBbits.TRISB1=1; // PIN RB1 as input (0-output 1-input)

    // All ports as outputs
    TRISA=0;
    TRISB=0;
    TRISC=0;
    TRISD=0;
    TRISE=0;

    // All Line Low
    LATA=0x00;
    LATB=0x00;
    LATC=0x00;
    LATD=0x00;
    LATE=0x00;

    //All Line In A Blink
    for(j=0; j<800; ++j); //delay
    LATA=0xff;
    for(j=0; j<1200; ++j); //delay
    LATA=0x00;

    //All Line In B Blink
    LATB=0xff;
    for(j=0; j<800; ++j); //delay
    LATB=0x00;
    for(j=0; j<1200; ++j); //delay
    LATB=0x00;

    //All Line In C Blink
    LATC=0xff;
    for(j=0; j<800; ++j); //delay
    LATC=0x00;
    for(j=0; j<1200; ++j); //delay
LATC=0x00;
    //All Line In D Blink
    for(j=0; j<800; ++j); //delay
LATD=0xff;
    for(j=0; j<1200; ++j); //delay
LATD=0x00;
    //All Line In E Blink
    for(j=0; j<800; ++j); //delay
LATE=0xff;
    for(j=0; j<1200; ++j); //delay
LATE=0x00;
    // Every diode at once
while(1)
{
    //Port A
    for (i=0;i<8;++i)
    {
        PORTA=0b1<<i;       //put on every diode once
        for(j=0; j<300; ++j); //delay
    }
    //Port B
    for (i=0;i<8;++i)
    {
        PORTB=0b1<<i;       //put on every diode once
        for(j=0; j<300; ++j); //delay
    }
    //Port C
    for (i=0;i<8;++i)
    {
        PORTC=0b1<<i;       //put on every diode once
        for(j=0; j<300; ++j); //delay
    }
    //Port D
    for (i=0;i<8;++i)
    {
        PORTD=0b1<<i;       //put on every diode once
        for(j=0; j<300; ++j); //delay
```c
// Port E

for (i=0; i<8; ++i)
{
    PORTE=0b1<<i;          // put on every diode once
    for(j=0; j<300; ++j);   // delay
}

// PORTB as output;
TRISB=0;
LATB=0xff;

// Main loops
while(1){
    LATB=0xff;
    delay_s((unsigned)1);
    for(i=0; i<=8; i++){
        LATB&=~(1<<i);    // change led colour
        // delay_us((unsigned)1);   // delay
    }
    for(j=0; j<600; ++j);   // delay
}

// * * * End main.c * * *

// * * * delay.c * Copyright(C)2016 Bohdan Borowik
#include "plik.h"
void delay_us(int us)
{unsigned long i=((FCPU/100000000.0)*us)/4;
 for (;i>0;--i);
}
void delay_ms(int ms)
{unsigned i;
 for (i=0;i<ms;++i)delay_us(998);
}
void delay_s(int s)
{unsigned i;
 for (i=0;i<s;++i)delay_ms(1000);
}
void delay(unsigned ile)
```
{  
    unsigned int i;
    for (i=0; i<ile; i++);
}

// *** End delay. c ***

3.3. Port A and TRISA Register

Port A is an 8-bit wide, bidirectional port. Bits of the TRISA and ANSEL control the PORTA pins. All portA pins act as digital inputs/outputs. Besides, five of them can also be analog inputs (denoted as AN):

<table>
<thead>
<tr>
<th>Name</th>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>PORTA</td>
<td>RA7 Ø</td>
<td>RA6 Ø</td>
<td>RA5</td>
<td>—</td>
<td>RA3</td>
<td>RA2</td>
<td>RA1</td>
<td>RA0</td>
</tr>
<tr>
<td>LATA</td>
<td>LATA7 Ø</td>
<td>LATA6 Ø</td>
<td>LATA5</td>
<td>—</td>
<td>LATA3</td>
<td>LATA2</td>
<td>LATA1</td>
<td>LATA0</td>
</tr>
<tr>
<td>TRISA</td>
<td>TRISA7 Ø</td>
<td>TRISA6 Ø</td>
<td>TRISA5</td>
<td>—</td>
<td>TRISA3</td>
<td>TRISA2</td>
<td>TRISA1</td>
<td>TRISA0</td>
</tr>
<tr>
<td>ANCON0</td>
<td>ANSEL7</td>
<td>ANSEL6</td>
<td>ANSEL5</td>
<td>ANSEL4</td>
<td>ANSEL3</td>
<td>ANSEL2</td>
<td>ANSEL1</td>
<td>ANSEL0</td>
</tr>
</tbody>
</table>

Legend: — = unimplemented, read as '0'. Shaded cells are not used by PORTA.
Note 1: These bits are enabled depending on the oscillator mode selected. When not enabled as PORTA pins, they are disabled and read as 'x'.
<table>
<thead>
<tr>
<th>Pin Name</th>
<th>Function</th>
<th>TRIS Setting</th>
<th>I/O</th>
<th>I/O Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RA0/CVref/AN0/ULPWU</td>
<td>RA0</td>
<td>0</td>
<td>O</td>
<td>DIG</td>
<td>LATA&lt;0&gt; data output; not affected by analog input.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>I</td>
<td>ST</td>
<td>PORTA&lt;0&gt; data input; disabled when analog input is enabled.</td>
</tr>
<tr>
<td></td>
<td>CVref</td>
<td>x</td>
<td>O</td>
<td>ANA</td>
<td>Comparator voltage reference output. Enabling this feature disables digital I/O.</td>
</tr>
<tr>
<td></td>
<td>AN0</td>
<td>1</td>
<td>I</td>
<td>ANA</td>
<td>A/D Input Channel 0. Default input configuration on POR; does not affect digital output.</td>
</tr>
<tr>
<td>ULPWU</td>
<td>RA1</td>
<td>0</td>
<td>O</td>
<td>DIG</td>
<td>Ultra Low-Power Wake-up input.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>I</td>
<td>ST</td>
<td>PORTA&lt;1&gt; data input; disabled when analog input is enabled.</td>
</tr>
<tr>
<td></td>
<td>AN1</td>
<td>1</td>
<td>I</td>
<td>ANA</td>
<td>A/D Input Channel 1. Default input configuration on POR; does not affect digital output.</td>
</tr>
<tr>
<td>C1INC(1)</td>
<td>C1INC</td>
<td>x</td>
<td>I</td>
<td>ANA</td>
<td>Comparator 1 Input C.</td>
</tr>
<tr>
<td>RA2/Vref-/AN2/C2INC</td>
<td>RA2</td>
<td>0</td>
<td>O</td>
<td>DIG</td>
<td>LATA&lt;2&gt; data output; not affected by analog input.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>I</td>
<td>ST</td>
<td>PORTA&lt;2&gt; data input; disabled when analog functions are enabled.</td>
</tr>
<tr>
<td></td>
<td>Vref-</td>
<td>1</td>
<td>I</td>
<td>ANA</td>
<td>A/D and comparator low reference voltage input.</td>
</tr>
<tr>
<td></td>
<td>AN2</td>
<td>1</td>
<td>I</td>
<td>ANA</td>
<td>A/D Input Channel 2. Default input configuration on POR.</td>
</tr>
<tr>
<td>C2INC(1)</td>
<td>C2INC</td>
<td>x</td>
<td>I</td>
<td>ANA</td>
<td>Comparator 2 Input C.</td>
</tr>
<tr>
<td>RA3/Vref+/AN3</td>
<td>RA3</td>
<td>0</td>
<td>O</td>
<td>DIG</td>
<td>LATA&lt;3&gt; data output; not affected by analog input.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>I</td>
<td>ST</td>
<td>PORTA&lt;3&gt; data input; disabled when analog input is enabled.</td>
</tr>
<tr>
<td></td>
<td>Vref+</td>
<td>1</td>
<td>I</td>
<td>ANA</td>
<td>A/D Input Channel 3. Default input configuration on POR.</td>
</tr>
<tr>
<td>RA5/AN4/C2INB/HLVDIN/T1CKI/SS/CTMUI</td>
<td>RA5</td>
<td>0</td>
<td>O</td>
<td>DIG</td>
<td>LATA&lt;5&gt; data output; not affected by analog input.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>I</td>
<td>ST</td>
<td>PORTA&lt;5&gt; data input; disabled when analog input is enabled.</td>
</tr>
<tr>
<td></td>
<td>AN4</td>
<td>1</td>
<td>I</td>
<td>ANA</td>
<td>A/D Input Channel 4. Default configuration on POR.</td>
</tr>
<tr>
<td>C2INB(2)</td>
<td>C2INB</td>
<td>x</td>
<td>I</td>
<td>ANA</td>
<td>Comparator 2 Input B.</td>
</tr>
<tr>
<td></td>
<td>HLVDIN</td>
<td>1</td>
<td>I</td>
<td>ANA</td>
<td>High/Low-Voltage Detect external trip point input.</td>
</tr>
<tr>
<td></td>
<td>T1CKI</td>
<td>x</td>
<td>I</td>
<td>ST</td>
<td>Timer1 clock input.</td>
</tr>
<tr>
<td></td>
<td>SS</td>
<td>1</td>
<td>I</td>
<td>ST</td>
<td>Slave select input for MSSP module.</td>
</tr>
<tr>
<td></td>
<td>CTMUI(2)</td>
<td>x</td>
<td>O</td>
<td>—</td>
<td>CTMU pulse generator charger for the C2INB comparator input.</td>
</tr>
<tr>
<td>RA6/OSC2/CLKOUT</td>
<td>RA6</td>
<td>0</td>
<td>O</td>
<td>DIG</td>
<td>LATA&lt;6&gt; data output; disabled when FOSC2 Configuration bit is set.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>I</td>
<td>ST</td>
<td>PORTA&lt;6&gt; data input; disabled when FOSC2 Configuration bit is set.</td>
</tr>
<tr>
<td></td>
<td>OSC2(2)</td>
<td>x</td>
<td>O</td>
<td>ANA</td>
<td>Main oscillator feedback output connection (HS, XT and LP modes).</td>
</tr>
<tr>
<td>RA7/OSC1/CLKIN</td>
<td>RA7</td>
<td>0</td>
<td>O</td>
<td>DIG</td>
<td>System cycle clock output (Fosc/4) (EC and INTOSC modes).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>I</td>
<td>ST</td>
<td>PORTA&lt;7&gt; data input; disabled when FOSC2 Configuration bit is set.</td>
</tr>
<tr>
<td></td>
<td>OSC1</td>
<td>1</td>
<td>I</td>
<td>ANA</td>
<td>Main oscillator input connection (HS, XT, and LP modes).</td>
</tr>
<tr>
<td></td>
<td>CLKIN</td>
<td>1</td>
<td>I</td>
<td>ANA</td>
<td>Main external clock source input (EC modes).</td>
</tr>
</tbody>
</table>

Legend:
- O = Output; I = Input; ANA = Analog Signal; DIG = CMOS Output; ST = Schmitt Trigger Buffer Input;
- x = Don’t care (TRIS bit does not affect port direction or is overridden for this option)

Note 1: This pin assignment is unavailable for 28-pin devices (PIC18F2XX00).
Note 2: This pin assignment is only available for 28-pin devices (PIC18F2XX00).
The power-on reset is generated automatically when power supply voltage is applied to the chip. The MCLR pin should be tied to the supply voltage directly or, preferably, through a 10K resistor. Figure 8 shows a typical reset circuit.
Bohdan Borowik,  
**Digital Electronics Projects**

Fig. 8. Typical reset circuit

**MCLR pin**

Logic zero (0) on the MCLR pin causes immediate and regular reset. It is recommended to be connected as shown in figure below. The function of additional components is to sustain logic one (1) during normal operation. If their values are such to provide high logic level on the pin only upon T reset is over, the microcontroller will immediately start operating. This feature may be very useful when it is necessary to synchronize the operation of the microcontroller with additional electronics or the operation of several microcontrollers. For applications where the rise time of the voltage is slow, it is recommended to use a capacitor, and a series resistor as shown in Figure 9.

Fig. 9: Reset circuit used a capacitor, and a series resistor.
In some applications the microcontroller may have to be reset externally by pressing a button. Figure 10 shows the circuit that can be used to reset the microcontroller externally. Normally the MCLR input is at logic 1. When the RESET button is pressed, this pin goes to logic 0 and resets the microcontroller.

![External Reset](image)

**Fig. 10 External Reset application.**

5. External oscillator

The external clock EC mode uses the system clock source configured from external oscillator. The frequency of this clock source is limited to 40MHz. This mode has the following advantages:

- The external clock source is connected to the OSC1 input and the OSC2 is available for general purpose I/O.
- It is possible to synchronize the operation of the microcontroller with the rest of on board electronics.
- In this mode the microcontroller starts operating immediately after the power is on. There is no delay required for frequency stabilization.
- Temporary stopping the external clock input has the effect of halting the device while leaving all data intact. Upon restarting the external clock, the device resumes operation as if nothing has happened.

An external clock source can be connected to the OSC1 input of the
microcontroller in EC and ECIO modes. No oscillator start-up time is required after a power-on reset. Figure 11 shows the operation with the external clock in EC mode.

![External Oscillator Diagram](image)

**Fig. 11 External Oscilator.**

The LP, XT and HS modes support the usage of external oscillator for configuring clock source. The frequency of this source is determined by quartz crystal or ceramic resonators connected to the OSC1 and OSC2 pins. Depending on features of the component in use, select one of the following modes:

- **LP mode** (Low Power) is used for low-frequency quartz crystal only. This mode is designed to drive only 32.768 kHz crystals usually embedded in quartz watches. It is easy to recognize them by small size and specific cylindrical shape. The current consumption is the least of the three modes.
- **XT mode** is used for intermediate-frequency quartz crystals up to 8 MHz. The current consumption is the medium of the three modes.
- **HS mode** (High Speed) is used for high-frequency quartz crystals over 8 MHz. The current consumption is the highest of the three modes.

The first several clock sources listed use an external crystal or ceramic resonator that is connected to the OSC1 and OSC2 pins. For applications where
accuracy of timing is important, a crystal should be used. And if a crystal is used, a parallel resonant crystal must be chosen, since series resonant crystals do not oscillate when the system is first powered. The capacitor values depend on the mode of the crystal and the selected frequency. Table 1 gives the recommended values. For example, for a 4MHz crystal frequency, use 15pF capacitors. Higher capacitance increases the oscillator stability but also increases the start-up time.

Table 1: Capacitor values

<table>
<thead>
<tr>
<th>Mode</th>
<th>Frequency</th>
<th>C1,C2 (pF)</th>
</tr>
</thead>
<tbody>
<tr>
<td>LP</td>
<td>32 KHz</td>
<td>33</td>
</tr>
<tr>
<td></td>
<td>200 KHz</td>
<td>15</td>
</tr>
<tr>
<td>XT</td>
<td>200 KHz</td>
<td>22–68</td>
</tr>
<tr>
<td></td>
<td>1.0 MHz</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>4.0 MHz</td>
<td>15</td>
</tr>
<tr>
<td>HS</td>
<td>4.0 MHz</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>8.0 MHz</td>
<td>15–33</td>
</tr>
<tr>
<td></td>
<td>20.0 MHz</td>
<td>15–33</td>
</tr>
<tr>
<td></td>
<td>25.0 MHz</td>
<td>15–33</td>
</tr>
</tbody>
</table>

The LP (low-power) oscillator mode is advised in applications to up to 200KHz clock. The XT mode is advised to up to 4MHz, and the HS (high-speed) mode is advised in applications where the clock frequency is between 4MHz to 25MHz. An external clock source may also be connected to the OSC1 pin in the LP, XT, or HS modes as shown in Figure 11.

5.1. Internal oscillator in RC and RCIO mode

There are certainly many advantages in using elements for frequency stabilization, but sometimes they are really not necessary. It is mostly enough the oscillator operates at frequency not precisely defined so that embedding of such expensive elements means a waste of money. The simplest and cheapest solution in these situations is to use one resistor and one capacitor for the operation of oscillator. There are two modes:
**RC mode.** In RC mode, the RC circuit is connected to the OSC1 pin as shown in figure. The OSC2 pin outputs the RC oscillator frequency divided by 4. This signal may be used for calibration, synchronization or other application requirements.

![Diagram of RC mode](image)

**Fig. 12:** Adjustable clock in RC mode

**RCIO mode.** Similar to the previous case, the RC circuit is connected to the OSC1 pin. This time, the available OSC2 pin is used as additional general purpose I/O pin. In both cases, it is recommended to use components as shown in figure. The frequency of such oscillator is calculated according to the formula \( f = \frac{1}{T} \)

*in which:*

- \( f = \text{frequency [Hz]} \)
- \( T = R \times C = \text{time constant [s]} \)
- \( R = \text{resistor resistance [Ω]} \)
- \( C = \text{capacitor capacity [F]} \)

Figure 13 shows how a crystal is connected to the microcontroller.
5.2 External source Clock Operation

An external source clock source can be connected to the OSC1 input of the microcontroller in EC and ECIO modes. No oscillator start-up time is required after a power-on reset. Figure 14 shows the operation with the external clock in EC mode. Timing pulses at the frequency $F_{\text{OSC}/4}$ are available on the OSC2 pin. These pulses can be used for test purposes or to provide pulses to external devices. The ECIO mode is similar to the EC mode, except that the OSC2 pin can be used as a general purpose digital I/O pin. As shown in Figure 15, this pin becomes bit 6 of PORTA (i.e., pin RA6).
5.3. Clock Switching

It is possible to switch the clock from the main oscillator to a low-frequency clock source. For example, the clock can be allowed to run fast in periods of intense activity and slower when there is less activity. In the PIC18F46K80 microcontroller this is controlled by bit SCS of the OSCCON register. In microcontrollers of the PIC18F family that do support an internal clock, clock switching is controlled by bits SCS0 and SCS1 of OSCCON. It is important to ensure that during clock switching unwanted glitches do not occur in the clock signal. PIC18F microcontrollers contain circuitry to ensure error-free switching from one frequency to another.

5.4. OSCON register

The OSCON register controls the main aspects of the device clock’s operation. It selects the oscillator type to be used, which of the power-managed modes to invoke and the output frequency of the INTOSC source. It also provides status on the oscillators. The OSCTUNE register controls the tuning and operation of the internal oscillator block. It also implements the
PLLEN bit which controls the operation of the Phase Locked Loop (PLL) for Frequency Multiplier.

**REGISTER 3-1: OSCCON: OSCILLATOR CONTROL REGISTER**

<table>
<thead>
<tr>
<th>Bit 7</th>
<th>IDLEN: Idle Enable bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Device enters an Idle mode when a SLEEP instruction is executed</td>
</tr>
<tr>
<td>0</td>
<td>Device enters Sleep mode when a SLEEP instruction is executed</td>
</tr>
</tbody>
</table>

**bit 6-4 IRCF<2:0>: Internal Oscillator Frequency Select bits**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>111</td>
<td>HF-INTOSC output frequency is used (16 MHz)</td>
</tr>
<tr>
<td>110</td>
<td>HF-INTOSC/2 output frequency is used (8 MHz, default)</td>
</tr>
<tr>
<td>101</td>
<td>HF-INTOSC/4 output frequency is used (4 MHz)</td>
</tr>
<tr>
<td>100</td>
<td>HF-INTOSC/8 output frequency is used (2 MHz)</td>
</tr>
<tr>
<td>011</td>
<td>HF-INTOSC/16 output frequency is used (1 MHz)</td>
</tr>
<tr>
<td>010</td>
<td>HF-INTOSC/32 output frequency is used (500 kHz)</td>
</tr>
<tr>
<td>001</td>
<td>HF-INTOSC/64 output frequency is used (250 kHz)</td>
</tr>
<tr>
<td>000</td>
<td>LF-INTOSC output frequency is used (31.25 kHz)</td>
</tr>
</tbody>
</table>

If INTSRC = 0 and MFIOSEL = 0:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>110</td>
<td>HF-INTOSC/32 output frequency is used (500 kHz)</td>
</tr>
<tr>
<td>101</td>
<td>HF-INTOSC/64 output frequency is used (250 kHz)</td>
</tr>
<tr>
<td>100</td>
<td>HF-INTOSC/512 output frequency is used (31.25 kHz)</td>
</tr>
</tbody>
</table>

If INTSRC = 1 and MFIOSEL = 0:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>110</td>
<td>MF-INTOSC output frequency is used (500 kHz)</td>
</tr>
<tr>
<td>101</td>
<td>MF-INTOSC/2 output frequency is used (250 kHz)</td>
</tr>
<tr>
<td>100</td>
<td>LF-INTOSC output frequency is used (31.25 kHz)</td>
</tr>
</tbody>
</table>

If INTSRC = 1 and MFIOSEL = 1:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>110</td>
<td>MF-INTOSC output frequency is used (500 kHz)</td>
</tr>
<tr>
<td>101</td>
<td>MF-INTOSC/2 output frequency is used (250 kHz)</td>
</tr>
<tr>
<td>100</td>
<td>MF-INTOSC/16 output frequency is used (31.25 kHz)</td>
</tr>
</tbody>
</table>

**bit 3 OSTS: Oscillator Start-up Timer Time-out Status bit**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Oscillator Start-up Timer (OST) time-out has expired; primary oscillator is running, as defined by FOSC&lt;3:0&gt;</td>
</tr>
<tr>
<td>0</td>
<td>Oscillator Start-up Timer (OST) time-out is running; primary oscillator is not ready – device is running from internal oscillator (HF-INTOSC, MF-INTOSC or LF-INTOSC)</td>
</tr>
</tbody>
</table>

**bit 2 HFIOFS: HF-INTOSC Frequency Stable bit**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>HF-INTOSC oscillator frequency is stable</td>
</tr>
</tbody>
</table>
0 = HF-INTOSC oscillator frequency is not stable
bit 1-0 **SCS<1:0>: System Clock Select bits** *(4)*

1x = Internal oscillator block (LF-INTOSC, MF-INTOSC or HF-INTOSC)
01 = SOSC oscillator
00 = Default primary oscillator (OSC1/OSC2 or HF-INTOSC with or without PLL; defined by the FOSC<3:0> Configuration bits, CONFIG1H<3:0>)

**Note 1:** The Reset state depends on the state of the IESO Configuration bit (CONFIG1H<7>).

2: Modifying these bits will cause an immediate clock frequency switch if the internal oscillator is providing the device clocks.

3: The source is selected by the INTSRC bit (OSCTUNE<7>).

4: Modifying these bits will cause an immediate clock source switch.

5: INTSRC = OSCTUNE<7> and MFIOSEL = OSCCON2<0>.

6: This is the lowest power option for an internal source.