PC USB Projects PROGRAMMING GUIDE

Version 3.6

based on:

PIC32MZ2048ECH100 v2.9.2 firmware, PIC32MZ2048ECH144 v2.9.2 firmware

PIC32MX250F128B v2.9.1 firmware, PIC32MX270F256B v2.9.1 firmware

PIC18F24J50 v2.6.3 firmware, PIC18F26J50 v2.7.8 firmware,

PIC18F2550 firmware v2.5 and v2.6, PIC18F4550 firmware v2.7.1,

and

LIB_PCUSBProjects v6.2.NET4(x64).dll

dr. Simon Vavpotič, 2014 ©

http://sites.google.com/site/pcusbprojects
simonvav@gmail.com

INDEX

1. THE IDEA _____ 5

2. PIC18 PROGRAMMING GUIDE _____ 6

INTRODUCTION TO PROGRAMMING _____ 6

LIB_PCUSBProjects and SVLIB_PIC18F24J50 PROGRAM LIBRARIES _____ 6

a. Add DLL library to your project: _____ 7
b. Now you are ready to use the library in your VB program _____ 7
c. Basic functions _____ 9
d. Bit manipulation and port functions _____ 10
e. Thermostatic control functions _____ 10
f. Velleman compatibility functions and USB connection control functions _____ 11
g. Real time clock (RTC) support _____ 14
h. Asynchronous counter support _____ 15
i. Frequency and pulse width measurement functions _____ 15
j. JTAG programmer support: Basic functions _____ 16
k. JTAG programmer support: PE and PE loader functions _____ 18
l. Low frequency generator support _____ 19
m. EEPROM programming functions and microcontroller restart _____ 20
n. User functinos execution _____ 20
o. LCD control functions _____ 20

2. PIC32 PROGRAMMING GUIDE _____ 22

INTRODUCTION TO PROGRAMMING _____ 22

LIB_PCUSBProjects PROGRAM LIBRARY _____ 22

a. Add DLL library to your project: _____ 23
b. Now you are ready to use the library in your VB program _____ 24
c. Operation mode selection _____ 25
d. Direct PIC32 memory access _____ 26
e. Emulated direct PIC18 memory access _____ 26
f. I/O port access functions _____ 27
g. Emulated PIC18F2xJ50 port access functions _____ 29
h. Virtual port bit access to a memory location _____ 30
i. Transfering data arrays between DLL and VB.NET applications _____ 31
j. Software microcontroller restart _____ 31
k. Advanced A/D converter functions (PIC32MX only) _____ 31
l. I^2C bus master functions _____ 32
m. ADT7410 16-bit external temperature sensor functions _____ 33
n. Thermostatic and watchdog control functions _____ 33
o. EEPROM programming _____ 36
p. User RAM and EEPROM functions _____ 36
q. Setting USB configuration _____ 37
r. 8-channel low frequency PWM generator _____ 38
s. Velleman K8055 and K8055N experiment board functions _____ 39

t. Asynchomous counter support 40
u. PIC18F2xJ50 PROGRAMMER SUPPORT 40
v. JTAG PROGRAMMER SUPPORT 41

3. HIGH LEVEL FUNCTIONS FOR PIC18 AND PIC32 MICROCONTROLLERS 44

INTRODUCTION 44

Transfering data arrays between DLL and VB.NET applications 44

SPI1 support for PIC18 44
a. Setting up MSSPs 45
b. Additional control pins 46
c. Data transfer 46
d. Software SPI interfaces 46
EXAMPLE 1: How to use software functions? 46
EXAMPLE 2: Direct PC communication with an SPI device for testing purposes 47
You can also use only MemRead and MemWrite intstructions 47
e. Hardware SPI functions in PIC18F26J50 firmware v2.6.5 or later 49
EXAMPLE 1: OPEN CONNECTION AND READ DATA 49
EXAMPLE 2: OPEN CONNECTION AND WRITE DATA 49

Software I2C functionality in PIC18F26J50 firmware 50
EXAMPLE 1: Reading air pressure from HP03MA sensor 51

Weather Station Functions for PIC18 52
a. Weather Station Functions 54
b. Using raw data 55
EXAMPLE1: HOW TO READ HUMIDITY AND AIR PRESSURE SENSOR RAW VALUES 56
c. Reading temperature, air pressure and humidity from PIC18F26J50 57
EXAMPLE1: HOW TO READ HUMIDITY AND AIR PRESSURE SENSOR FINAL VALUES 58
d. Setting firmware constants 58
e. PIC18F26J50 remote GLCD (RGLCD) support 59

Graphics LCD (LGM12864B) with touchscreen functions for PIC18 60
a. Text and graphics functions 61
b. EXAMPLE 1: GLCD DEMO v3 APPLICATION WITH SOURCE CODE 63
c. EXAMPLE 2: GLCD HARDWARE TESTING APPLICATION 64
d. I^2C bus remote GLCD slave commands 64

SPI support for PIC32 66
a. SPI functions 66
EXAMPLE 1: OPEN CONNECTION AND READ DATA 67
EXAMPLE 2: OPEN CONNECTION AND WRITE DATA 67
EXAMPLE 3: You can use only MemRead and MemWrite intstructions 67

Hoperf RFC85 and RFC83C(L) simple 1-bit wireless communications 68

HopeRF RFM69CW wireless transceiver module 70
a. How it works? 70
b. Packet communications 70
c. RFM69CW functions 70
EXAMPLE 1: INITIALIZE RFM69CW MODULE, IF NECESSARY AND SEND STRING 71
EXAMPLE 2: INITIALIZE RFM69CW MODULE, IF NECESSARY AND RECEIVE STRING 72

d. Inbuilt wireless software responder module _____ 72
e. Setting Inbuilt wireless software responder module EEPROM defaults _____ 73

4. MICROCONTROLLER PROGRAMMERS _____ 75

@.WINDOWS APPLICATIONS for Microcontroller Programmers _____ 75

A. Simple USB microcontroller programmer _____ 76

B. Fast USB microcontroller programmer schematics _____ 76

C. Simple and fast microcontroller programmers based on Velleman K8055-1 and K8055N-2 boards _____ 77

D. High programming voltage microcontroller programmer adapter _____ 77

E. Super speed reconfigurable USB microcontroller programmer _____ 78

F. Super speed programmer software development kit (SDK) _____ 78

G. Super speed microcontroller programmers can now be used with PIC18F24J50, too! _____ 79

H. JTAG PROGRAMMER FOR PIC32 FAMILY _____ 81

5. STARTER KIT GUIDE _____ 82

INTRODUCTION TO STARTER KITS _____ 82

A. SV PIC18F2xJ50 STARTER KIT SCHEMATIC _____ 82

B. SV PIC18F2xJ50 STARTER KIT SCHEMATIC – alternative 1 _____ 84

C. SV PIC18F2xJ50 STARTER KIT SCHEMATIC – alternative 2 _____ 85

D. Basic circuit for PIC18F24J50 or PIC18F26J50 to work over USB _____ 86

E. Need more speed? Here is a basic circuit for PIC18F24J50 and PIC18F26J50 for high speed operation _____ 87

1. THE IDEA

Microcontrollers are computers in one chip and they are fun to play with. But… No PC has an inbuilt microcontroller programmer and many new PCs don't have printer (centronix) and modem ports (RS232) anymore. Therefore two major obstacles have to be overcome before you can even think of programming a microcontroller:

- You have to make or buy a microcontroller programmer that you can be used with your PC.

- You need a suitable programming environment to make the microcontroller communicate with the PC.

An USB microcontroller programmer is certainly the best option to solve the first problem. If you thing of its future use, it is also important that the enclosed Windows programming application is also a native 64-bit application, with a 32-bit version just as an option. This is important, because the operating systems for PCs will all get 64-bit only in a few years from now.

The programming environment includes all the necessary software for your PC to create programs that allow your PC to talk to a microcontroller. This involves a high programing language support and some kind of DLL (dynamic link library) and/or a suitable device driver. Newer versions of Microsoft Windows already have inbuilt USB HID (human device interface) driver; therefore DLL (dynamic link library) is sufficient to connect a microcontroller to PC.

It is also not a bad idea to have an SDK kit for you USB microcontroller Windows programming application available. This would allow you to program future microcontrollers with just some relatively simple changes to the programming application.

2. PIC18 PROGRAMMING GUIDE

INTRODUCTION TO PROGRAMMING

Programming microcontrollers from scratch is relatively difficult due to many open questions that may arise. The first task is to build a programmer circuit. Next you need to write a programming application that programs HEX files. But this is only a half of the story. You also need a programming environment, at least an assembler and a linker. However, it is more convenient to install a more complex development environment with a higher programming language like C, C++ or Basic. You should also consider using program libraries, if you have to solve a more complex problem.
The next task is to get as much documentation about the microcontroller you are about to program as possible. It is important to understand the microcontroller architecture and its functional units operation before you begin programming.

Though there are many microcontroller programmers on the market you can build one for just a fraction of the cost. There are two options: either you by a preprogrammed microcontroller, or build one yourself. Please, see Microcontroller programmers section if you would like to build a programmer for a few euros or dollars…

Now, when you have a programmed microcontroller that talks to your computer as a HID USB device, you just need interface software to control it from your PC. The easiest way is to use a suitable DLL that comes with the HEX file. DLL implements simple HID functions and provides for inter-thread and inter-process synchronization. The latter means that you will be able to access the microcontroller from different threads of one application, or from different threads within other applications.

Now, you just have to write your own application… See section "*How to start programming PIC18F24J50 or PIC18F26J50?*" for details.

LIB_PCUSBProjects and SVLIB_PIC18F24J50 PROGRAM LIBRARIES

There are many ways to use a PIC18F24J50, PIC18F26J50 or PIC18F2550 microcontroller in a custom circuit. You do not have to use assembly language to program it. C++ and especially Microsoft Visual Basic are much better options.

Most of the following samples are in VB.NET, but you can find VC.NET (C++) samples in Downloads section. Though programming in Visual C++.NET is very similar to programming in VB.NET there are some syntax differences and some differences in Visual Studio C++ development environment.

There are two important steps:

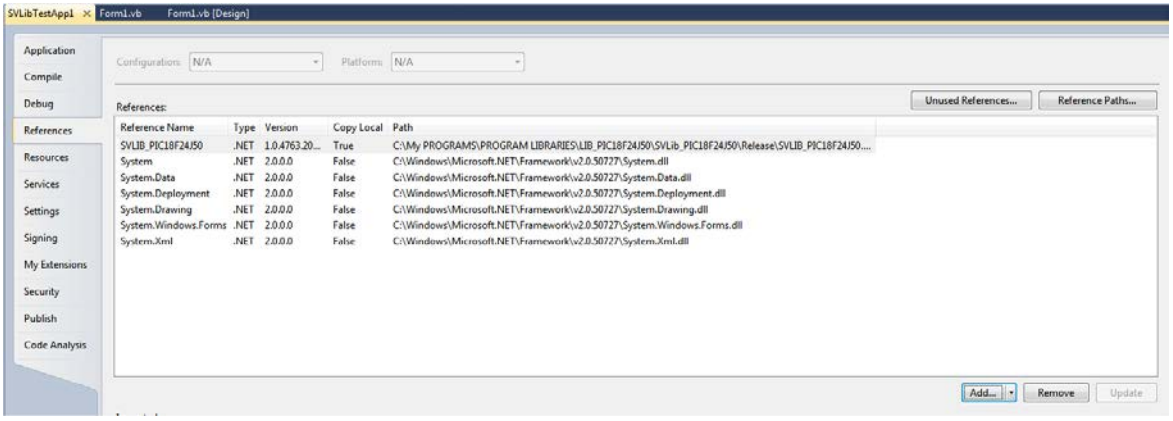
1. Obtain a preprogrammed microcontroller or program a microcontroller with a HEX file (ex. PIC18F24J50 firmware v2.x (xx MHz) for general use.hex from the Downloads section) that turns it into a HID (human interface device) USB device. The original PIC18F2550 firmware v2.6 (xx MHz) for general use *.hex and PIC18F2xJ50 firmware v2.x (xx MHz) for general use.hex (4 MHz, 12 MHz, 16 MHz version, or 20 MHz version) file have inbuilt: VID = 0x4D8 and PID = 0xF001 (0xE001 in version 2.4). Windows programming application enables you to change both values before start of programming. If you use more than one USB device on your computer it is important for each device to have its own VID & PIC combination. This is even more important if you plan to use more than one PIC18F24J50 or PIC18F26J50 microcontroller. Though there is no practical limit on the number of controllers that you can attach to your computer, each must have unique VID & PID. Therefore it is a good idea to program your microcontrollers with consecutive PIDs (ex. 0xF001, 0xF002, 0xF003, ...).

2. Next, you have to **create a .NET (ex. VB.NET or VC.NET) project and attach the DLL library** that enables communication between high programming language program and the microcontroller. Now, you can start programming...

   Here is a sample VB code on how to use ***LIB_PCUSBProjects vx.y.NETn.dll*** in your programs (**DLLs also works with PIC18F26J50).**
   Please, note: *SVLIB_PIC18F24J50 v4.0.dll* or ***LIB_PCUSBProjects v6.2.NET4x64.dll*** require PIC18F2xJ50 with *SVPIC18F2xJ50 firmware v2.6.5.hex* or later for full functionality. Microsoft Visual Studio.NET 4.0 applications reqiure a .NET 4.0 DLL. If you plan a .NET4.0 application, use ***LIB_PCUSBProjects v6.2.NET4.dll or LIB_PCUSBProjects v6.2.NET4x64.dll***. You may also use ***SVLIB_PIC18F24J50 v2.7c.st.NET4(x64).dll*** for single threaded applications.

   NOTE: *LIB_PCUSBProjects v6.2.NET4x64.dll* is the latest library that supports all the functions described below and also supports PIC18F24J50, PIC18F26J50, PIC18F2550 and PIC32MX250F128B/PIC32MX270F256B microcontrollers.

a. Add DLL library to your project:



Click on "Project" → "<project name> Properties".. and select "References". Then click the "Add" button and LIB_PCUSBProjects v6.2.NET4.dll from 32-bit applications or LIB_PCUSBProjects v6.2.NET4x64.dll for 64-bit applications.

NOTE: The libraries are downwards compatible, so you can always use a library with higher version number.

b. Now you are ready to use the library in your VB program

The library supports only class "SVPICAPI". You must create an object to use the class.

Next you must declare and create the object:

```
Dim PIC As LIB_PCUSBProjects.SVPICAPI
PIC = New LIB_PCUSBProjects.SVPICAPI
```

LIB_PCUSBProjects v6.2.NET4.dll or LIB_PCUSBProjects v6.2.NET4x64.dll libraries support the following operation modes:

There are four compatibility mode flags:

PIC32_mode
PIC18F2550_mode
K8055_legacy_mode
MCP2200_legacy_mode

All the flags are **originally set to false** to indicate native mode: **PIC18F24J50/PIC18F26J50**.

NOTE: Only one flag may be set **true** at a time, leave all flags **false**, if you are using PIC18F2xJ50 microcontroller.

NOTE: Existing programs must be adapted to set the appropriate compatibility mode before using other functions of **LIB_PCUSBProjects v6.2.NET4(x64).dll** program library.

Examples of applying compatibility mode settings for different micrcontrollers:

```
' How to set PIC32MX250F128B/PIC32MX270F256B compatibility mode:
Dim PIC As New LIB_PCUSBProjects.SVPICAPI
PIC.PIC32_mode = True
....

' How to set PIC18F2550 compatibility mode:
Dim PIC As New LIB_PCUSBProjects.SVPICAPI
PIC.PIC18F2550_mode = True
....

' How to set MCP2200 legacy compatibility mode:
Dim PIC As New LIB_PCUSBProjects.SVPICAPI
PIC.MCP2200_legacy_mode = True
....

' How to set K8055 legacy compatibility mode:
Dim PIC As New LIB_PCUSBProjects.SVPICAPI
PIC.K8055_legacy_mode = True
```

To establish connection to the PIC18F24J50 microcontroller you must first call function “Open” with PID as a parameter. VID is permanently set to “4D8”.

```
PIC.Open(&HF001)
```

If the function succeeds than the communication between PIC18F24J50 and PC is established and you may send commands. The PIC18F24J50 firmware is deliberately programmed the way that it does not use any of the port B outputs to indicate proper connection to USB. This behavior is deliberate because we usually want to use all 8 outputs for control and we do not want to send signals to the circuitry at PIC18F24J50 startup.

Here is an example on how to set one of the digital outputs on port B:

```
ListBox1.Items.Add(CStr(PIC.SetDigitalOutputs(data)))
```

Data is an 8-bit value that specifies the output values of all 8 channels. To clear or set only one of the channels a simple binary mathematics may be used. It is also possible to read back the value of the port B. Here is an example:

```
ListBox1.Items.Add("port B = " + CStr(Hex(PIC.ReadDigitalInputs(Asc("B")))))
```

ReadDigitalInputs function can in fact read from ports “A”,”B” and “C”. Port “A” and “C” are initially programmed as follows:

port A = all inputs (8-bits)

port B = all outputs (8-bits)

port C = all outputs, except for RC1 and RC2 which are outputs

RA0 and RA1 bits of the port A bits are initially connected to AN0 and AN1 and they are configured as analog inputs. A 10-bit A/D converter with multiplexor reads the voltage (0 .. Vdd) in them. The values of both inputs can be read as follows:

```
ListBox1.Items.Add(CStr(PIC.ReadAnalogInput(CByte(TbAddress.Text))))
```

RC0 and RC1 bits of the port C are programmed as PWM1 and PWM2 (PWM = pulse width modulator) outputs. You can build your own circuit to convert those outputs to D/A 1 and D/A 2 that convert impulses to the average voltage between +0 V and +5 V. If you use Velleman’s K8055 or K8055N board the D/A outputs are already implemented. PWM resolution is 10-bits, which also applies to D/A conversion outputs. Here is an example on how to use PWM modules:

```
ListBox1.Items.Add(CStr(Hex(PIC.SetPWM(1023, 500))))
```

Function SetPWM sets two 10-bit values (decimal range from 0 to 1023) for PW1 and PWM2 accordingly.

c. Basic functions

The list of the supported functions by the SVPICAPI class is the following:

```
Function Open(devID as Integer) as Integer   ' devID = 0..&hFFFF (default = &hF001)
Function SetDigitalOutputs(portB as Byte) as Integer   ' sets digital outputs
Function ReadAnalogInput(ADchannel as Byte) as Integer ' ADchannel=0..15
Function ReadDigitalInputs(port as Byte) as Integer    'port = asc('A'), asc('B') or
asc('C')
‘int SetDefaultOutputValue(char port, byte lat); 'port = asc('A'), asc('B') or
asc('C'), lat = ‘default value = 0..255
‘.. See Microchip documentation
Function GetDefaultOutputValues(port as Byte) as Integer ' reads default values
Function Close() as Integer                          ‘ closes communication
Function DataMemRead(word addr) as Integer            ' reads 1 byte from RAM (12-bit
addressing)
Function DataMemWrite(word addr, byte data) as Integer ' writes 1 byte to RAM (12-bit
addressing)
Function ProgMemRead(dword addr) as Integer      ' 1 byte of reads program memory in
EEPROM ‘(24-bit addressing)
```

Functions **DataMemRead** and **DataMemWrite** may be used to access PIC18F24J50, PIC18F26J50 or PIC18F2550 memory during operation. While most of the memory is not of interest of windows application, the last memory block is. It contains all PIC18 internal registers, so the functions give access to all the registers. If you want to write to port B you may also use the following code:

```
PIC.DataMemWrite(&hF81, <8-bit value>)
```

To read value from port B you can use instruction

```
<an 8-bit value or a negative value, if error> = PIC.DataMemRead(&hF81)
```

The function ProgMemRead enables to read whole program memory. This enables one to read program code and configuration register values. The type and version of PIC chip is written permanently on addresses &h3FFFFE and &h3FFFFF. To read these addresses you simply write:

```
<an 8-bit value or a negative value, if error> = PIC.ProgMemRead(&h3FFFFE)

<an 8-bit value or a negative value, if error> = PIC.ProgMemRead(&h3FFFFF)
```

d. Bit manipulation and port functions

```
GetPortMemAdr(port as String) as Uint16; // returns port memory address
```

The function returns 16-bit PIC18 memory address where port resides. There are three ports: “A”, “B” and “C”. Example:

```
Adr = GetPortMemAdr(Asc(“A”))

SetDataMemBit(adr as UInt16, bitNumber as Byte) as Integer
```

The function sets a bit on desired PIC18 memory address.

```
ClearDataMemBit(adr as UInt16, bitNumber as Byte) as Integer
```

The function clears a bit on desired PIC18 memory address.

```
SetPin(port as String, data as Byte) as Integer
ClearPin(port as String, data as Byte) as Integer
SetPin(data as Byte) as Integer
ClearPin(data as Byte) as Integer
```

All the functions above are derivatives of ***SetDataMemBit*** and ***ClearDataMemBit*** functions. The first two work on ports “A”, “B” and “C”. The following examples sets bit 0 on port A and B of PIC18F24J50:

```
SetPin(Asc(“A”),0)

SetPin(Asc(“B”),0)

SetPin(0)
```

It is clear that if port parameter is omitted the port B is presumed. This port is configured as output port on Velleman K8055.

e. Thermostatic control functions

There are 6 tunctions that support thermostatic control operation:

```
Function ReadBufferWORD(adr as Byte) as Integer
Function WriteBufferWORD(byte adr, WORD data) as Integer
Function LoadThermostaticControlValues() as Integer
Function StartThermostaticControl() as Integer
Function StopThermostaticControl()as Integer
Function ReadBackThermostaticControlValues() as Integer
```

ReadBufferWORD and **WriteBufferWORD** are used in conjunction with functions **ReadBuffer** and **WriteBuffer** to access read and write buffers for PIC18Fxxxxx and PC block data exchange. The new functions enable data word operations on the buffers. A data word of 16-bits is written to the write buffer or read from the read buffer low byte first.

Function **LoadThermostaticControlValues** loads thermostatic control configuration from buffer to the PIC18F26J50 microcontroller (16 bytes). The configuration is composed as follows:

1 byte ….. A/D A input for ambient temperature value
1 byte ….. A/D B input for internal refrigerator temperature value
1 byte ….. Relay for refrigerator power control (0 = on/ 1 = off)
1 byte ….. Relay for refrigerator heater control (0 = on/ 1 = off)
2 bytes … Ambient switch on threshold temperature A/D A value
2 bytes … Ambient switch off threshold temperature A/D A value
2 bytes … Internal: compressor switch off threshold temperature A/D B value
2 bytes … Internal: heating switch on threshold temperature A/D B value
2 bytes … Internal: heating switch off threshold temperature A/D B value

Here is also an example on how to load the thermostat configuration and start thermostatic control:

```
PIC.WriteBuffer(0, 1)  ' A/D A = A/D 0
PIC.WriteBuffer(1, 0)  ' A/D B = A/D 1
PIC.WriteBuffer(2, 0)  ' Relay A = bit = 0
PIC.WriteBuffer(3, 1)  ' Relay B = bit = 1
PIC.WriteBufferWORD(4, 550) ' Ambient switch on threshold temperature A/D A value
PIC.WriteBufferWORD(6, 500) ' Ambient switch off threshold temperature A/D A value
PIC.WriteBufferWORD(8, 400) ' Internal: compressor switch off threshold temperature A/D B
value
PIC.WriteBufferWORD(10, 450) ' Internal: compressor switch on threshold temperature A/D B
value
PIC.WriteBufferWORD(12, 250) ' Internal: heating switch on threshold temperature A/D B
value
PIC.WriteBufferWORD(14, 300) ' Internal: heating switch off threshold temperature A/D B
value
ListBox1.Items.Add(CStr(PIC.LoadThermostaticControlValues()))
ListBox1.Items.Add(CStr(PIC.StartThermostaticControl()))
```

Function **ReadBackThermostaticControlValues** retrieves thermostat configuration from the microcontroller and stores if into the read buffer. It can be read with ReadBuffer commands. Here is an example:

```
ListBox1.Items.Add("R = " + CStr(PIC.ReadBackThermostaticControlValues()))
For n = 0 To 3
    ListBox1.Items.Add(Hex(PIC.ReadBuffer(n)))
Next
For n = 4 To 14 Step 2
    ListBox1.Items.Add(CStr(PIC.ReadBufferWORD(n)))
Next
```

Functions **StartThermostaticControl** and **StopThermostaticControl** take no parameters. They only instruct the microcontroller to start or stop automatic thermostat control.

f. Velleman compatibility functions and USB connection control functions

This library requires a 32 or 64-bit Windows version with .NET4 installed. There are 7 new functions and an updated `IsConnected()` function:

```
Function IsConnected(devID as Integer) as Boolean  ' (DLL VERSION 2.6)
```

The above function enables detection of a connected device with an arbitrary PID (devID parameter). It the device is present on PC's USB, then the return value is **true** regardless the device is opened or not. Function:

```
Function IsConnected() as Boolean
```

… is enhanced and now it first checks weather the device is opened and then it checks it the device is present on the USB. It both is true than the result is **true**, otherwise the result is **false**.

SVLIB_PIC18F24J50 v2.7.st and v2.8 libraries introduce the legacy operating mode to support microcontrollers with original Velleman firmware. There is also an important upgrade that allows connecting to a USB device with arbitrary vendor and product IDs. The libraries are built in 32-bit and 64-bit subversions. SVLIB_PIC18F24J50 v2.7.st is intended for single thread applications while SVLIB_PIC18F24J50 v2.8 is intended for single and multiple thread applications. The SVLIB_PIC18F24J50 v2.8 also has a better control over legacy and advanced functions.

Functions **Open** and **IsConnected** now have the following variants:

```
Function Open(devID as Integer) as Integer      ' initiates communiction
Function Open(vendorID as Integer, devID as Integer) as Integer   ' initiates
communication
Function IsConnected() as Boolean               ' enhanced USB connection detection
Function IsConnected(devID as Integer) as Boolean
Function IsConnected(vendorID as Integer, devID as Integer) as Boolean
```

Open function can now take one or two 32-bit parameters. If a one parameter variant is used, devID defaults to 0x4D8, which is reserved for the Microchip devices domain. The vendor and product IDs are stored as internal variables of object that implements an interface to the microcontroller.

IsConnected function takes none, one or two parameters. The variant without parameters is intended for checking whether microcontroller is still connected, but the other two variants are also used do to check is other microcontrollers are present. IsConnected function with product ID (devID) only checks if there are any new devices present in the vendor's domain, while the function with two parameters performs a check to find an arbitrary USB device with specified vendor and product IDs.

There are also additional functions that ease the use of microcontrollers. Most of them operate are similarly or equally to original Velleman K8055D.DLL functions. Here is the list of the function definitions:

```
Function OpenDevice(int CardAddress) as Integer
Sub CloseDevice()
Sub WriteAllDigital(int Data)
Function SearchDevices() as UInt32
Function SetCurrentDevice(int lngCardAddress) as Integer
Function ReadAnalogChannel(int Channel) as Integer
Sub OutputAnalogChannel(int Channel, int Data)
Sub OutputAllAnalog(int Data1, int Data2)
Sub ClearAnalogChannel(int Channel)
Sub ClearAllAnalog()
Sub SetAnalogChannel(int Channel)
Sub SetAllAnalog()
Sub ClearDigitalChannel(int Channel)
Sub ClearAllDigital()
Sub SetDigitalChannel(int Channel)
Sub SetAllDigital()
Function ReadDigitalChannel(int Channel) as Boolean
Function ReadAllDigital() as Integer
Function ReadBackDigitalOut() as Integer
Sub ResetCounter(int CounterNr)
Function ReadCounter(int CounterNr) as Integer
Sub SetCounterDebounceTime(int CounterNr, int DebounceTime)
```

The most of the new functions work in both advanced (new firmware from this website) and legacy (Velleman) mode. **OpenDevice** function calls **Open** function with a different default vendor ID and base product ID depending on the selected operation mode. If the advanced mode is selected (see **SetK8055LegacyMode** function description below) the vendor ID is 0x4D8 and the base product ID is 0xF001. If legacy mode is selected then vendor ID is 0x10CF and base product ID is 0x5500.

SearchDevices function is also dependent on the selected operation mode and the base product ID. It uses **IsConnected** function to find a maximum of 32 devices in advanced mode and 4 devices in legacy mode with product IDs starting from the base product ID.

SetCurrentDevice only works in legacy mode. It opens a connection to a microcontroller with 0x1CF vendor ID and product ID which is a sum of base product ID and **lngCardAddress**.

Functions **ResetCounter**, **ReadCounter** and **SetCounterDebounceTime** are only supported in legacy mode.

There are two additional functions to set and to check the currently set operating mode:

```
Function SetK8055LegacyMode(legacy_FW_mode as Boolean) as Integer
Function K8055LegacyMode() as Boolean
```

SetK8055LegacyMode function sets legacy mode, if the **legacy_FW_mode** parameter is **true** and advanced mode, if the parameter is **false**. The legacy mode mimics the communication of K8055D.DLL to the microcontroller. However, the advanced mode offers much greater functionality.

The remaining functions that also work in legacy mode are:

```
Function Open(devID as Integer) as Integer   ‘ initiates communiction
Function Open(vendorID as Integer, devID as Integer) as Integer   ‘ initiates
communiction ((NEW with v2.7: includes VID)
Function SetDigitalOutputs(byte portB as Byte) as Integer ‘ sets digital outputs
Function ReadAnalogInput(byte ADchannel as Byte) as Integer ‘, * value) as Integer   ‘
read analog channel
Function ReadDigitalInputs(port as Byte) as Integer ‘ byte* portA, byte* portB, byte*
portC) as Integer  ‘ reads all ports
Sub Close()  ‘ closes communication
Function SetPWM(dutyCycle1 as UInt16, dutyCycle2 as UInt16) as Integer ‘ sets PWM1 and
PWM2 and consequetially DAC1 and DAC2 values
Function SetPWM1(dutyCycle1 as UInt16) as Integer ' sets PWM1 and consequetially DAC1
values
Function SetPWM2(dutyCycle2 as UInt16) as Integer ' sets PWM2 and consequetially DAC2
values
Function DataMemRead(addr as UInt16) as Integer ' Reads PIC data memory Function
DataMemWrite(addr as UInt16, byte data as Byte) as Integer ' Writes to PIC Data
memory)
Function GetPortMemAdr(port as Byte) as UInt16’ returns port memory address
Function SetDigitalOutputs( port as Byte, byte portB as Byte) as Integer ‘ sets
digital outputs
Function SetDataMemBit(adr as UInt16, byte bitNumber as Byte) as Integer
Function ClearDataMemBit(adr as UInt16, bitNumber as Byte) as Integer
Function SetPin(port as Byte,byte data as Byte) as Integer
Function ClearPin( port as Byte,byte data as Byte) as Integer
Function SetPin(byte data as Byte) as Integer   ‘ default port = B
Function ClearPin(byte data as Byte) as Integer ‘ default port == B
Function FirmwareVersion() as Integer
Function Version() as Integer
Function GetPWMDutyCycle(byte PWMnum as Byte) as Integer
```

FirmwareVersion function returns 0 in legacy mode, otherwise it returns a 32-bit integer that contains the main version (2 bytes) followed by 1-byte subversion and 1-byte subsubversion.

DataMemRead and **DataMemWrite** functions only work if the address is 0xf81, which is the address of port B in a PIC18F2xJ50 microcontroller. **DataMemWrite** calls **SetDigitalOutputs** function and **DataMemRead** returns the current value of port B, stored in PC memory. PIC16C745 offers has way of reading back the output value.

If a function is not supported or a parameter value is not supported by legacy more the function would return a "not supported" error code (decimal value of "-99").

g. Real time clock (RTC) support

PIC18F2xJ50 microcontrollers have an inbuilt RTC module. This is an important functionality that enables on-time execution of different tasks. It has also an inbuilt alarm function that may automatically set the PB1 (RP11) output when the alarm is triggered.

RTC may run in many oscillator configurations. Configuration 0 enables it to operate without an additional crystal resonator. Configuration 1 requires an external 32.768 kHz crystal resonator, but assures more accurate time measurement.

```
Function RTC_Init(OscMode as Byte) as Integer
```

These functions enable starting, enabling and stopping (disabling) RTC. When RTC is disabled its register values are not updated:

```
Function RTC_Start() as Integer
Function RTC_Stop() as Integer
Function RTC_Enabled() as Boolean
```

It is important to know, whether the RTC is running. This function return **true**, if RTC is running, and **false**, it is not.

```
Function RTC_RB1OUT_Enable() as Integer
Function RTC_RB1OUT_Disable() as Integer
Function RTC_AL_RB1OUT_Enabled() as Boolean
```

These three functions enable or disable RTC to take control over RB1 (RP11) output and return current alarm module status.

```
Function RTC_Calibration(Data as Byte) as Integer
```

The operating frequency of RTC should be around 32768 Hz, but it may vary. The data parameter enables positive and negative adjustments (from -128 to +127). One bit increment instructs RTC to add four clock pulses every minute. This is equivalent to the input frequency of 0,7 Hz frequency increase. The maximum increase is therefore 127 * 0,7 Hz = 8,467 Hz. Similarly, one bit decrement means -0,7 Hz, which implies -8,533 Hz.

```
Function RTC_GetCalibration() as Integer
```

The function returns calibration value between -128 and +127.

```
Function RTC_PAD_Configuration(data as Byte) as Integer
Function RTC_PAD_GetConfiguration() as Integer
```

The above functions enable selection of the RTC output function on pin RB1 (RP11) and retrieval of the value. It is set to 0 by default. Please, see Microchip documentation for other values.

```
Function RTC_Year() as Integer
Function RTC_Month() as Integer
Function RTC_Day() as Integer
Function RTC_Weekday() as Integer
Function RTC_Hours() as Integer
Function RTC_Minutes() as Integer
Function RTC_Seconds() as Integer
Function RTC_AL_Month() as Integer
Function RTC_AL_Day() as Integer
```

```
Function RTC_AL_Weekday() as Integer
Function RTC_AL_Hours() as Integer
Function RTC_AL_Minutes() as Integer
Function RTC_AL_Seconds() as Integer
```

The above functions enable reading current date & time as well as alarm date & time. The following functions are similar to current date & time functions. They enable one to enter a date and time of an alarm.

```
Function RTC_AL_Enable() as Integer
Function RTC_AL_Enabled() as Boolean
Function RTC_SetDate(year as Integer, month as Byte, day as Byte, weekday as Byte) as
Integer
Function RTC_SetTime(hours as Byte, minutes as Byte, seconds as Byte) as Integer
Function RTC_AL_Disable() as Integer
Function RTC_AL_SetInterval(AL_mask as Byte) as Integer
Function RTC_AL_SetNumRepetitions(NRep as Byte) as Integer
Function RTC_AL_InfiniteRepetitionsEnabled(InfiniteRepetitions as Boolean) as Integer
Function RTC_AL_NumRepetitions() as Integer
Function RTC_AL_InfiniteRepetitions() as Boolean
Function RTC_AL_SetDate(month as Byte, day as Byte, weekday as Byte) as Integer
Function RTC_AL_SetTime(hours as Byte, minutes as Byte, seconds as Byte) as Integer
```

h. Asynchronous counter support

PIC18FxxJ50 microcontrollers have five counters T0 through T4. T0 and T3 timers may also be used as hardware counters. The following functions are supported:

```
Function SetCounter(CounterNr as Integer, val as Integer) as Integer
Function ResetCounter(CounterNr as Integer) as Integer
Function ReadCounter(CounterNr as Integer) as Integer
Sub SetCounterDebounceTime(CounterNr as Integer, DebounceTime as Integer)
```

SetCounter function allows you to load an arbitrary 16-bit into a counter (counter number 0 or 1). This function is only supported with firmware 2.x. **ResetCounter** sets the selected counter to 0. **Read counter** reads the 16-bit value from counter. **SetCounterDebounceTime** is only supported with the original Velleman firmware, which implements counters in software.

i. Frequency and pulse width measurement functions

There are two sets of functions that enable you to measure signal frequency and phase. Timer 1 and be used directly or it can be a clock source for an ECCP unit that operates in capture mode.

ECCP SIGNAL LENGTH MEASUREMENTS:

There are a number of functions that enable you to control ECCP2 unit in capture mode. The basic capture functionality is provided through **CCP2CaptureRisingEdge** function. PIC18F2550 firmware v2.6 has limited support and only supports the function below that return positive signal length:

```
Function CCP2CaptureRisingEdge() as Int32
```

However, firmware v2.6.2 for PIC18F24J50 and PIC18F26J50 also support the following capture control functions:

```
Function CCP2enable() as Integer
Function CCP2enable(mode as Integer) as Integer
Function CCP2disable() as Integer
Function CCP2suspend() as Integer
Function CCP2restart() as Integer
```

`CCP2enable` enables configures Timer 1 and ECCP2 to operate in capture mode. CCP2 input is set to RP13 pin that corresponds to PWM2 output on Velleman K8055-1 and K8055N-2 board. It is also possible to relocate CCP2 input by writing a different value into **RPINR8 register** of Preripheral Pin Select unit on PIC18F2xJ50. PIC18F2550 has no such unit and CCP2 input can only be relocated to RB3 pin through CCP2MX bit in configuration register 3. PC USB Project PIC18F2550 firmware sets CCP2 to RC1.

RPINR8 register can be accessed with memory read and write functions:

```
PIC.DataMemWrite(&hEEE,<number of RP pin>)
```

PIC18F24J50 and PIC18F26J50 firmwares also support CCP2enable function with operation mode parameter. Operation mode 0 instructs function CCP2CaptureRisingEdge to capture length of the high period the signal, but operation more 1 cases the function capture of the whole length of the signal. The latter can be used to calculate the signal frequency from the Timer 1 frequency. The Timer 1 frequency is initially set to Fosc/4, where Fosc it the frequency of PIC clock source (external or internal main oscillator).

The remaining functions disable, suspend and restart ECCP and Timer 1. Disable function changes operation mode of ECCP2 back to PWM, so PR13 pin becomes PWM2 output. Suspend function only temporary disables signal length capturing. Signal length capturing resumes, when CCP2restart function is called.

TIMER 1 SIGNAL LENGTH MEASUREMENTS

There are also four functions as follows that provide frequency and phase measurement directly through Timer 1:

```
Function Timer1Init(RPpin as Byte, prescaler as Byte) as Integer
Function Timer1Frequency() as Integer
Function Timer1HighImpulseWidth() as Integer
Function Timer1LowImpulseWidth() as Integer
```

The **Time1Init** function initializes PIC18F2xJ50 internal Timer1 to measure frequency and pulse width. It has two parameters. **RPpin** specifies the pin of PIC18F2xJ50 microcontroller chip, which will be used for measurements. **See subsection 5.b. for details on RPx pins**. The **prescaler** parameters may have one of three values that determine the frequency divider (1:1, 1:2, 1:4 or 1:8) used to measure frequency or impulse width. Extremely low frequencies require the use of a higher frequency divider.

Timer1Frequency, **Timer1HighImpulseWidth** and **Timer1LowImpulseWidth** all return a 16-bit integer value, which holds the number of processor core clock ticks counted during the one impulse time. The processor core runs at 48 MHz (for all firmware versions published on this website). Therefore the following commands in VB.NET must be executed in application software to obtain and display the actual frequency and actual pulse widths values:

```
cnt = PIC.Timer1Frequency()
lowcnt = PIC.Timer1LowImpulseWidth()
highcnt = PIC.Timer1HighImpulseWidth()
impulse_time = cnt / CPU_core_frequency * 4
freq = 1 / impulse_time
impulse_HighWidth = highcnt / CPU_core_frequency * 4
impulse_LowWidth = lowcnt / CPU_core_frequency * 4
Lb_Frequency.Text = CStr(freq) + " Hz"
Lb_HighImpulseWidth.Text = CStr(impulse_HighWidth * 1000) + " ms"
Lb_LowImpulseWidth.Text = CStr(impulse_LowWidth * 1000) + " ms"
```

j. JTAG programmer support: Basic functions

JTAG programmer transfers data to a microcontroller via 4 wires. The protocol is message oriented and supports communication in both directions (IN and OUT) at the same time. Each output message is composed of a multibit TMS header and a multibit input value and the output value of the same length. Most messages for PIC24 and PIC32 programming are unidirectional, which means that only an input or an output value is relevant. The other value has no meaning. TMS header and the value are given by two parameters: bit length and a 16-bit or a 64-bit variable. The maximum length is determined by data type length. The maximum TMS length is therefore 16-bits and the maximum value length is 64-bits. The length parameter defines a number of bits (starting from bit 0) from the variable to be sent via JTAG interface. If the number for bits is 5, then only the first 5 bits of the variable will be sent.

Program the appropriate PIC18F26J50 firmware HEX file to your PIC18F26J50 microcontroller, include the PC USB Projects DLL in your project and add the following function to your JTAG programming application to able to quickly send and receive JTAG messages:

```
Function jtag_SendReceive(PIC As SVLib_PIC18F24J50.SVPICAPI, tms_header_len As Byte, tms As
UInt16, val_len As Byte, val As UInt64, ByRef out As Int64) As Boolean
On Error GoTo abort
        out = PIC.jtag_SendReceive(tms_header_len, tms, val_len, val)
         If out < 0 Then GoTo abort
        Return True
        Exit Function
       abort:
        Return False
End Function

Function jtag_SetMode(PIC As SVLib_PIC18F24J50.SVPICAPI, len As Byte, val As UInt64) As
Boolean' Set mode
    On Error GoTo abort
    Dim out As UInt64
    Return jtag_SendReceive(PIC, len, val, 0, 0, out) ' out is not used
    Exit Function
 abort:
   Return False
End Function

Function jtag_SendCommand(PIC As SVLib_PIC18F24J50.SVPICAPI, len As Byte, val As UInt64) As
Boolean' Send command
    On Error GoTo abort
    Dim out As UInt64
    Return jtag_SendReceive(PIC, 4, &H3, len, val, out) ' out is not used
   Exit Function
 abort:
   Return False
End Function

Function jtag_XferData(PIC As SVLib_PIC18F24J50.SVPICAPI, len As Byte, val As UInt64, ByRef
out As UInt64) As Boolean' XferData
    On Error GoTo abort
    Return jtag_SendReceive(PIC, 3, &H1, len, val, out) ' out is not used
   Exit Function
abort:
   Return False
End Function

Function jtag_XferFastData(PIC As SVLib_PIC18F24J50.SVPICAPI, len As Byte, val As UInt64,
ByRef out As UInt64) As Boolean' XferFastData
   On Error GoTo abort
   Dim res As Boolean
   Dim controlVal As UInt64
   Dim n As Integer = 100
   val <<= 1
   ' add PrAcc input bit 0
    res = jtag_SendReceive(PIC, 3, &H1, len + 1, val, out)
```

```
   ' read data
   out >>= 1
   ' drop of PrAcc out bit
    Return res
    Exit Function
 abort:
   Return False
End Function

Function jtag_XferInstruction(PIC As SVLib_PIC18F24J50.SVPICAPI, inst As UInt64) As Boolean
  On Error GoTo abort
  Dim controlVal As UInt64
  Dim out As UInt64
  If Not jtag_SendCommand(PIC, ETAP_CMD_LEN, ETAP_CONTROL) Then GoTo abort
  Do
      If Not jtag_XferData(PIC, 32, &H4C000, controlVal) Then GoTo abort
   Loop While (controlVal And &H40000) = 0 ' 2^18

   If Not jtag_SendCommand(PIC, ETAP_CMD_LEN, ETAP_DATA) Then GoTo abort
   If Not jtag_XferData(PIC, 32, inst, out) Then GoTo abort
   If Not jtag_SendCommand(PIC, ETAP_CMD_LEN, ETAP_CONTROL) Then GoTo abort
   Return jtag_XferData(PIC, 32, &HC000, out)
    Exit Function
 abort:
    Return False
 End Function
```

k. JTAG programmer support: PE and PE loader functions

PE and PE loader:

Function **ReadPIC32PE**(pos as integer) as integer // Reads PE value at desired location, which contains a 32-bit word.
Function **GetPIC32PE_Size**() as integer // Returns PE size in 32-bit words.
Function **ReadPIC32PELoader**(pos as integer) as UInt32 // Reads PE Loader halfword (16-bits)
Function **GetPIC32PELoader_Size**() as UInt32 // Returns PE Loader size in 16-bit halfwords.
To download PE to a PIC32 microcontroller add the following function to your VB application:

```
    Function PIC32_DownloadPE(PIC As SVLib_PIC18F24J50.SVPICAPI) As Boolean

        ' Serial execution mode must already be entered!
        On Error GoTo abort
        Dim n As Integer
        Dim data As UInt64
        Dim out As UInt64
        If Not jtag_XferInstruction(PIC, &H3C04BF88) Then GoTo abort
        If Not jtag_XferInstruction(PIC, &H34842000) Then GoTo abort
        If Not jtag_XferInstruction(PIC, &H3C05001F) Then GoTo abort
        If Not jtag_XferInstruction(PIC, &H34A50040) Then GoTo abort
        data = &HAC85
        data <<= 16
        If Not jtag_XferInstruction(PIC, data) Then GoTo abort
        If Not jtag_XferInstruction(PIC, &H34050800) Then GoTo abort
        data = &HAC85
        data <<= 16
        data = data Or &H10
        If Not jtag_XferInstruction(PIC, data) Then GoTo abort
        ' LW A1,0x40(A0)
        data = &H8C85
        data <<= 16
        data = data Or &H40
        If Not jtag_XferInstruction(PIC, data) Then GoTo abort
        ' ORI $A1,$ZERO,0x8000 (32k)
```

```
        'If Not jtag_XferInstruction(PIC, &H34058000) Then GoTo abort
        data = &HAC85
        data <<= 16
        data = data Or &H20
        If Not jtag_XferInstruction(PIC, data) Then GoTo abort
        data = &HAC85
        data <<= 16

        data = data Or &H30
        If Not jtag_XferInstruction(PIC, data) Then GoTo abort
        If Not jtag_XferInstruction(PIC, &H3C04A000) Then GoTo abort
        If Not jtag_XferInstruction(PIC, &H34840800) Then GoTo abort

        ' Step 6
        For n = 0 To PIC.GetPIC32PELoader_Size() - 1 Step 2
            data = &H3C060000 Or PIC.ReadPIC32PELoader(n)
            If Not jtag_XferInstruction(PIC, data) Then GoTo abort
            data = &H34C60000 Or PIC.ReadPIC32PELoader(n + 1)
            If Not jtag_XferInstruction(PIC, data) Then GoTo abort
            data = &HAC86
            data <<= 16
            If Not jtag_XferInstruction(PIC, data) Then GoTo abort
            If Not jtag_XferInstruction(PIC, &H24840004) Then GoTo abort
        Next

        ' execute

        If Not jtag_XferInstruction(PIC, &H3C19A000) Then GoTo abort
        If Not jtag_XferInstruction(PIC, &H37390800) Then GoTo abort
        If Not jtag_XferInstruction(PIC, &H3200008) Then GoTo abort
        If Not jtag_XferInstruction(PIC, &H0) Then GoTo abort

        ' Step 7 - A: Start PE loader

        Delay(100)  ' Allow loader to start
        If Not jtag_SendCommand(PIC, ETAP_CMD_LEN, ETAP_FASTDATA) Then GoTo abort
        data = &HA000
        data <<= 16
        data = data Or &H900
        If Not jtag_XferFastData(PIC, 32, data, out) Then GoTo abort ' Starting Address
        If Not jtag_XferFastData(PIC, 32, PIC.GetPIC32PE_Size, out) Then GoTo abort 'Size

        ' Step 7 - B:

        For n = 0 To PIC.GetPIC32PE_Size() - 1
            If Not jtag_XferFastData(PIC, 32, PIC.ReadPIC32PE(n), out) Then GoTo abort '
transfer 1 CPU command
        Next

        If Not jtag_XferFastData(PIC, 32, 0, out) Then GoTo abort ' 0
        data = &HDEAD  ' magic word
        data <<= 16
        If Not jtag_XferFastData(PIC, 32, data, out) Then GoTo abort ' Send Magic sign to
stop loading
        Delay(100)
        PEDownloaded = True
        Return True
        Exit Function
abort:
        Return False

    End Function
```

I. Low frequency generator support

http://sites.google.com/site/pcusbprojects
simonvav@gmail.com

```
Function PIC.LowFreqGenConfiguration(<low phase length> as integer, <high phase
length> as integer, <number for RBx output x={0,1,..7}> as byte) as Integer
```

Configures the low frequency generator

```
Function PIC.LowFreqGen_Enable() as Integer
```

Enables and starts the generator.

```
Function PIC.LowFreqGen_Disable() as Integer
```

Stops and disables the generator.

m. EEPROM programming functions and microcontroller restart

```
Function ProgMemWrite(addr as UInt32, dta as Uint16) as Integer
```

Writes a 16-bit word to PIC internal program memory (EEPROM) location addr. Only lower 24-bits of addr variable are used, because PIC only provides 24-bit addressing for program memory. addr must be an even address (ex. &hE000, &hE002 ...)

```
Function ProgMemPageErase(addr as UInt32) as Integer
```

' Program memory of PIC18F26J50 must be erased prior to programming. The memory can only be erased in 1024-byte pages that are aligned to 1024-byte (1k byte) boundaries.

```
Function Restart() as Integer
```

Restarts the microcontroller.

n. User functinos execution

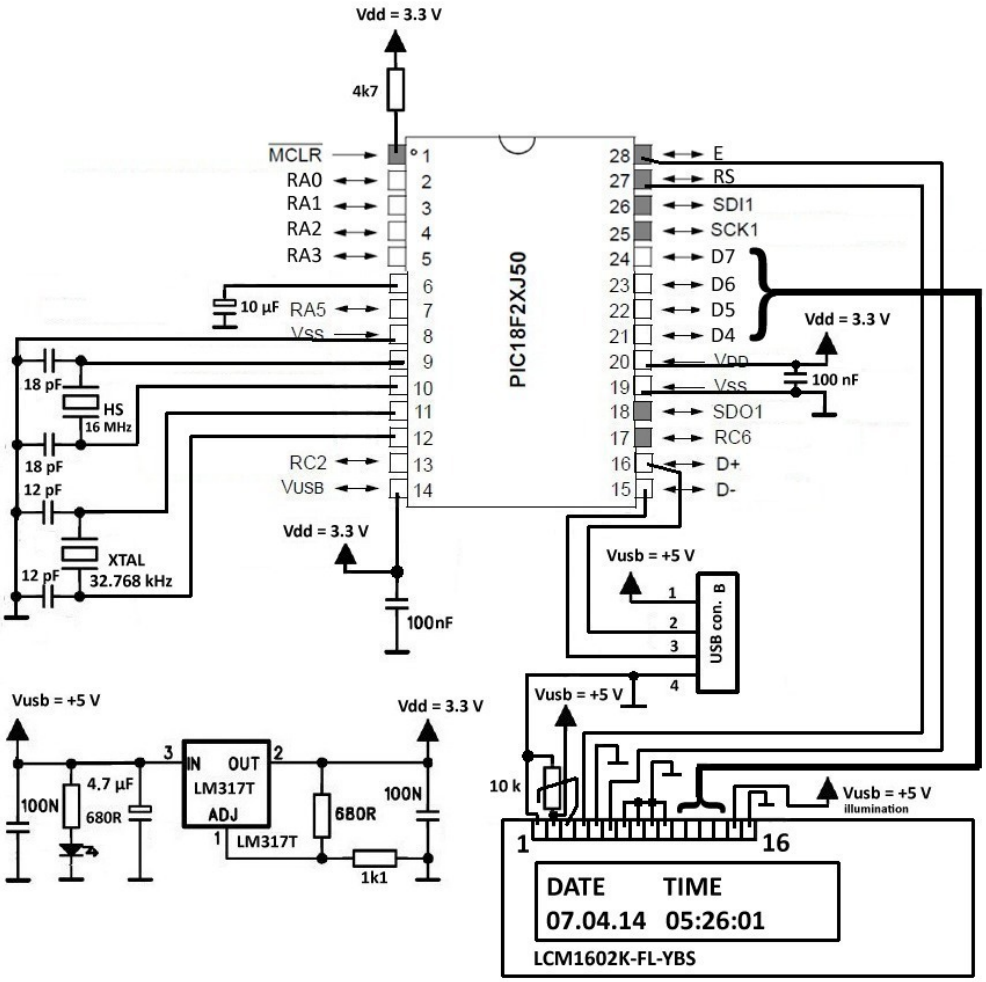
ExecuteE000 function starts a preprogrammed user function execution from the address &hE000. The function is only available in PIC18F26J50 firmware v2.6.1. **ProgMemPageErase** and **ProgMemWrite** must be used prior to this command to upload a user function. A user function examples (***PIC18F26J50 firmware v2.6.1 - ADD-ON5.zip*** and ***PIC18F26J50 firmware v2.6.1 - ADD-ON.zip***) are provided from Downloads section on PC USB Projects website (https://sites.google.com/site/pcusbprojects/6-downloads). Up to 62 bytes of parameters may be passed to a use function and up to 63 bytes result may be received. The parameters are passed as an byte array with the help of **WriteBuffer** (8-bit parameters) and/or **WriteBufferWORD** (16-bit parameters) functions. When the user function finishes the resulting parameter array is returned. ReadBuffer and ReadBufferWORD functions may be used to read 8-bit and 16-bit the results.

Here is an example of call of **ExecuteE000** function from Visual Basic.NET for ***PIC18F26J50 firmware v2.6.1 - ADD-ON5.zip*** user function sample:

```
Private Sub BuExecuteE000_Click(sender As System.Object, e As System.EventArgs) Handles
BuExecuteE000.Click
        PIC.WriteBuffer(0, CByte(TbData.Text)) ' Write Input parameter 1
        PIC.ExecuteE000()
        ListBox1.Items.Add(CStr(Hex(PIC.ReadBuffer(1))))  ' Read output parameter 1
        ListBox1.Items.Add(CStr(Hex(PIC.ReadBuffer(2))))  ' Read output parameter 2
End Sub
```

o. LCD control functions

SPLC780D can operate on 2.7 V or more, but LCM1602K-FL-YBS LCD would only work with 5 V power supply, but it has 3.3 V compatible inputs. So it can be directly connected to a 3.3 V microcontroller outputs, a 3.3 V microcontroller can only receive data from LCM1602K-FL-YBS LCD panel via a voltage converter. One may avoid a voltage converter by connecting the LCD R/W signal to the ground. This prevents the display panel to send data and also spared one of the microcontrollers I/O pins. This way only 6 PIC18F2xJ50 pins are needed to connect the display. But proper delays must be used not to exceed the LCD response times for longer operations.



The following functions are supported:

```
Sub LCDinit()
Sub LCDwrite8(dta as Byte)
Sub LCDwrite(dta as Byte, rs as Byte)
Sub LCDdatetimeRefresh(interval as Byte)
```

LCD control firmware functions are currently only supported by **PIC18F24J50 firmware v2.6.3** and **PIC18F26J50 firmware v2.6.3**. LCDinit function initializes LCD display with SPLC780D controller. The schematic is shown on the picture above. LCDwrite8 function is used during initialization to send commands through 8-bit bus (D0 through D7) with trailing bits D0 through D3 set to zero. LCDwrite function sends commands and data through a 4-bit data bus (D4 through D7).

LCDdatetimaRefresh function is used to set display refresh interval for independent date and time display. Only the first LCD row is used to display date and time.

Get the Digital Clock with LCD example for PIC18F2xJ50 (PIC18F2xJ50 Digital Clock with LCD example.zip) from Downloads section for more information. The example works with PIC18F2xJ50 formware v2.5 or newer.

Get PIC18F2xJ50 RTC Example with display v3 from Downloads section for more information.

2. PIC32 PROGRAMMING GUIDE

INTRODUCTION TO PROGRAMMING

Programming microcontrollers from scratch is relatively difficult due to many open questions that may arise. The first task is to build a programmer circuit. Next you need to write a programming application that programs HEX files. But this is only a half of the story. You also need a programming environment, at least an assembler and a linker. However, it is more convenient to install a more complex development environment with a higher programming language like C, C++ or Basic. You should also consider using program libraries, if you have to solve a more complex problem.
The next task is to get as much documentation about the microcontroller you are about to program as possible. It is important to understand the microcontroller architecture and its functional units operation before you begin programming.

Though there are many microcontroller programmers on the market you can build one for just a fraction of the cost. There are two options: either you by a preprogrammed microcontroller, or build one yourself. Please, see Microcontroller programmers section if you would like to build a programmer for a few euros or dollars…

Now, when you have a programmed microcontroller that talks to your computer as a HID USB device, you just need interface software to control it from your PC. The easiest way is to use a suitable DLL that comes with the HEX file. DLL implements simple HID functions and provides for inter-thread and inter-process synchronization. The latter means that you will be able to access the microcontroller from different threads of one application, or from different threads within other applications.

Now, you just have to write your own application…

LIB_PCUSBProjects PROGRAM LIBRARY

There are many ways to use a PIC32MX (ex. PIC32MX250F128B/PIC32MX270F256B) or a PIC32MZ (ex. PIC32MZ2048ECH100/PIC32MZ2048ECH144) microcontroller in a custom circuit. You do not have to use assembly language to program it. C++ and especially Microsoft Visual Basic are much better options.

NOTE: PIC32 means any of PIC32MX250F128B, PIC32MX270F256B, PIC32MZ2048ECH100 and PIC32MZ2048ECH144 microcontrollers.

There are two important steps:

1. Obtain a preprogrammed microcontroller or program a microcontroller with a HEX file (*PIC32 ….hex* from the Downloads section) that turns it into a HID (human interface device) USB device. The original HEX file has inbuilt: VID = 0x4D8 and PID = 0xD001 (for PIC32MX microcontrollers) or PID=0xF100 (for PIC32MZ microcontrollers). If you use more than one USB device on your computer it is important for each device to

have its own VID & PID combination. This is even more important, if you plan to use more than one PIC32 microcontroller.

Use **PIC32MX250F128B_setup_utility v2** (from Downloads section) to set your own VID & PID combination via USB after the microcontroller is programmed for PIC32MX microcontrollers. Or use **PC USB Projects HEX Editor v3.0** with a wider range of setup options for PIC32MX and PIC32MZ microcontrollers.

Though there is no practical limit on the number of controllers that you can attach to your computer, each must have unique VID & PID. Therefore it is a good idea to program your microcontrollers with consecutive PIDs (ex. 0xD001, 0xD002, 0xD003, ...).

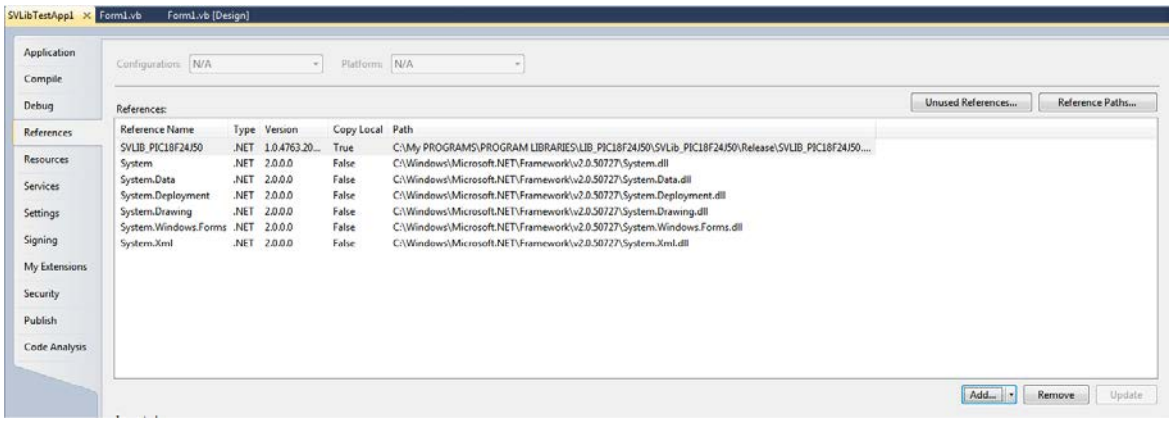
2. Next, You have to **create a .NET (ex. VB.NET or VC.NET) project and attach the DLL library** that enables communication between high programming language program and the microcontroller. Now, you can start programming...

There are in fact many ways to use the PIC32 microcontroller in a custom circuit. You do not have to use assembly language to program it. C++ and especially Microsoft Visual Basic are much better options. Here is a sample VB code on how to use of **LIB_PCUSBProjects v6.2.NET4(x64).dll** in your programs.

a. Add DLL library to your project:

A universal DLL (dynamic program library) for Windows simplifies the use of supported PIC microcontrollers. 32-bit or 64-bit versions of **LIB_PCUSBProjects v6.2.NET4(x64).dll (or later)** are available from Downloads section. The library is preconfigured to start in PIC18F24J50 and PIC18F26J50 compatibility mode. All other microcontroller compatibility modes require setting of a compatibility flag after the "PIC" object creation. See an example on the right hand side.
The library supports PIC18F2550, PIC32MX250F128B/PIC32MX270F256B and PIC32MZ2048EC100/PIC32MZ2048ECH144 compatibility modes that enables you to use these microcontrollers without workarounds. It also supports MCP2200, PIC16C745 and PIC18F24J50 original firmwares through legacy compatibility mode flags.
LIB_PCUSBProjects v6.2.NET4(x64).dll adds support for wireless communication protocol for data transfer. Read more on wireless communication modules here. PIC32MX250F128B DATA TRANSFER PROTOCOL (x64).zip example is available from Downloads section.

LIB_PCUSBProjects.h (LIB_PCUSBProjects v6.2.NET4 - header file.zip) header file is also available from Downloads section.



Click on “Project” → “<project name> Properties”.. and select “References”. Then click the “Add” button and add **LIB_PCUSBProjects v6.2.NET4.dll or LIB_PCUSBProjects v6.2.NET4x64.dll** (if your application is 64-bit).

b. Now you are ready to use the library in your VB program

The library supports only class “SVPICAPI”. You must create an object to use the class.

Next you must declare and create the object:

```
Dim PIC As LIB_PCUSBProjects.SVPICAPI
PIC = New LIB_PCUSBProjects.SVPICAPI
PIC.PIC32_mode = 1   ‘ ← for PIC32MX
```

or

```
Dim PIC As LIB_PCUSBProjects.SVPICAPI
PIC = New LIB_PCUSBProjects.SVPICAPI
PIC.PIC32_mode = 2   ‘ ← for PIC32MZ
```

See the next chapter (2.c) for details on operation mode selection.

To establish connection to the microcontroller you must first call function “Open” with PID as a parameter.

PIC.Open(VID,PID)

Example: PIC.Open(&H4D8,&HD001)

or

PIC.Open(PID)

Example: PIC.Open(&H4D8,&HD001) → VID is permanently set to “4D8”.

If the function succeeds than the communication between PIC32 and PC is established and you may send commands. The microcontroller firmware is deliberately programmed the way that it does not use any of the port B outputs (8-port B pins on PIC32 are used as outputs, the rest are unused or inputs) to indicate proper connection to USB. This behavior is deliberate because we usually want to use all 16 outputs for control and we do not want to send signals to the circuitry at PIC32 startup.

IsConnected function takes none, one or two parameters. The variant without parameters is intended for checking whether microcontroller is still connected, but the other two variants are also used do to check is other microcontrollers are present. IsConnected function with product ID (devID) only checks if there are any new devices present in the vendor’s domain, while the function with two parameters performs a check to find an arbitrary USB device with specified vendor and product IDs.

IsConnected(vendorID as Uint32, devID as Uint32) as Boolean

IsConnected(devID as Uint32) as Boolean ' vendor ID of &H4D8 is presumed.

The above function enables detection of a connected device with an arbitrary PID (devID parameter). It the device is present on PC’s USB, then the return value is **true** regardless the device is opened or not.
Function:

IsConnected() as Boolean

… first checks weather the device is opened and then it checks, if the device is present on the USB. If both are true the answer is **true**, otherwise **false** returned.

Summary

Functions **Open** and **IsConnected** now have the following variants:

Open(devID as UInt16) as Integer
Open(vendorID as UInt16, devID as UInt16) as Integer
IsConnected() as Boolean '
IsConnected(devID as UInt16) as Boolean
IsConnected(vendorID as UInt16, devID as UInt16) as Boolean

c. Operation mode selection

LIB_PCUSBProjects v6.2.NET4.dll or LIB_PCUSBProjects v6.2.NET4x64.dll libraries support the following operation modes:

There are four compatibility mode flags:

PIC32_mode
PIC18F2550_mode
K8055_legacy_mode
MCP2200_legacy_mode

All the flags are originaly set to **false** to indicate native compatibility mode: PIC18F24J50/PIC18F26J50.

NOTE: Only one flag may be set **true** at a time.
NOTE: Existing programs must be adapted to set the appropriate compatibility mode before using other functions of LIB_PCUSBProjects v6.2.NET4(x64).dll program library.

Examples of applying compatibility mode settings for different micrcontrollers:

' How to set PIC32MX250F128B/PIC32MX270F256B compatibility mode:
```
Dim PIC As New LIB_PCUSBProjects.SVPICAPI
PIC.PIC32_mode = 1
....
```

' How to set PIC32MZ2048ECH100/PIC32MZ2048ECH144 compatibility mode:
```
Dim PIC As New LIB_PCUSBProjects.SVPICAPI
PIC.PIC32_mode = 2
....
```

' How to set PIC18F2550 compatibility mode:
```
Dim PIC As New LIB_PCUSBProjects.SVPICAPI
PIC.PIC18F2550_mode = True
....
```

' How to set MCP2200 legacy compatibility mode:
```
Dim PIC As New LIB_PCUSBProjects.SVPICAPI
PIC.MCP2200_legacy_mode = True
....
```

' How to set K8055 legacy compatibility mode:

```
Dim PIC As New LIB_PCUSBProjects.SVPICAPI
PIC.K8055_legacy_mode = True
....
```

d. Direct PIC32 memory access

Direct memory access functions allow a host PC to access all PIC32 internal functional units (all registers) and all its memories (boot and program EEPROM and static RAM). The functions literal expose all the PIC32 internal functionality to the host. This enables one to experiment and test various procedures to be programmed directly to a PIC32 microcontroller, or to offer a comprehensive I/O extension to a newer PC without any useful I/O ports (like printer port or serial port) to connect simple external devices.

PIC32 microcontrollers have four memory access modes regarding the length and type of the memory location being read or written:

- byte read/write (8-bit access)
- word read/write (16-bit access)
- dword (double word) read/write (32-bit access)
- dword write to microcontroller EEPROM (32-bit access) and EEPROM page erase functionality

The program library has 8 functions to support these functionalities:

MemRead(addr as UInt32) as Int64 // Read double word
MemWrite(addr as UInt32, data as UInt32) as Int // Write double word
MemReadWord(addr as UInt32) as Int
MemWriteWord(addr as Uint32, data as Uint16) as Int
MemReadByte(addr as Uint32) as Int16;
MemWriteByte(addr as Uint32, data as byte) as Int
PIC32_WriteEEPROM(addr as Uint32, data as Uint32) as Int // Writes a double word to the desired EEPROM location
PIC32_ErasePageEEPROM(addr as UInt32) as Int // Erases one EEPROM page (pages are 1024 bytes long and they aligned to 1 kB boundaries (lower 10 bits of 32-bit address are neglected)

All functions return an integer result. A negative integer is error code. A positive integer result is a success code ("Write" functions return 0 upon successful write operations), or an output value ("read" functions). Example: **MemReadByte** function returns a positive value from range 0..255, or a negative error value. **MemWriteByte** returns 0 after a successful write, or a negative value in case of failure.

WARNING: The functions above are intended for expert users. Address parameter **addr** must always be within the allowed address ranges of a chosen PIC32 microcontroller. It is also important that the values entered to the PIC32 internal registers are correct. The address ranges are described in detail in Microchip PIC32 microcontroller datasheet (book code DS61168E). Access to a non-existing memory location may cause undesired microcontroller operation and the microcontroller may end-up in a deadlock and stop to communicate via USB. The same is true for incorrect values entered to the registers.

Use **PIC32_WriteEEPROM** function only on previously erased locations, or you might cause permanent damage to the microcontroller internal EEPROM. Use **PIC32_ErasePageEEPROM** function to erase desires EEPROM pages.

e. Emulated direct PIC18 memory access

Functions **DataMemRead** and **DataMemWrite** were originally developed to access PIC18F2xJ50 memory during operation. They also allowed direct access to all PIC18F2xJ50 internal registers. The following code enabled access to port B:

PIC.DataMemWrite(&hF81, <8-bit value>)

To read value from port B you could use instruction

<8-bit variable> = PIC.DataMemRead(&hF81)

But **&hF81** is the only address that gets translated to the PC32 port B address. PIC32 DataMemRead and DataMemWrite functions also support virtual PIC18F2xJ50 port B by converting 8-bit value to 16-bit value for port B on PIC32 and vice versa on virtual PIC18F2xJ50 port B read access.

DataMemRead and DataMemWrite functions only support 16-bit memory addresses, so PIC32MX FW has an inbuilt base address of &hBF880000 and PIC32MZ FW has an inbuilt base address of &hBF860000. The 16-bit address is used as an offset value to the base address for all addresses except &HF81.

ProgMemRead function is not supported by PIC32 firmware.

f. I/O port access functions

PIC32 microcontrollers have two general 16-bit I/O ports: A and B. PIC18 have on the contrary three 8-bit ports. Though it is obvious that two 16-bit ports may control more I/O pins there are less available pins on a PIC32 microcontroller. PIC32 advanced A/D converter requires special voltage reference inputs. The mapping of external pins to port bits resembles the evolution of PIC32 microcontrollers. PIC32 microcontrollers without USB support have more I/O pins available, while their more advanced cousins PIC32 lost general I/O pins to make room for dedicated control inputs.

PIC18F2xJ50 microcontroller firmware programming (*.HEX files) on this website initially program port B as 8 relay control outputs and use 5 bits of port A and port C for inputs and the remaining bits for control functions. This configuration originates from a popular Velleman K8055N board and assures PIC18F2xJ50 microcontroller compatibility, should it be used to replace the original microcontroller on the board.

PIC32 microcontroller also follows this idea. The firmware programming from this website allows for easy implementation of K8055 and K8055N adapters, which allow you to fully exploit the board features. However, PIC32 microcontroller does not have a dedicated port for relay control outputs. It also uses some of the port B bits as digital inputs. In fact, +5.5 V tolerant pins are used as inputs. This simplifies connection of the microcontroller to K8055 board, which was designed for a 5V PIC16C745 microcontroller.

RA pins 0 through 4 are not mapped to port A. They are routed to A/D converter and PWM.

PIC32MX250F128B/PIC32MX270F256B port A pin assignment:

PIC32 pin name	port A bit number	Direction	Description
RA4	4	output	used as PWM 1 output and D/A channel 0
RA3	3	output	used as PWM 2 output and D/A channel 1
RA1	1	input	used as A/D input channel 1
RA0	0	input	used as A/D input channel 0

PIC32MX250F128B/PIC32MX270F256B port B pin assignment:

PIC32 pin name	port B bit number	direction	description
RB15	15	output	relay output 8 on K8055(N)
RB14	14	output	relay output 7 on K8055(N)
RB13	13	output	relay output 6 on K8055(N)
RB9	9	input (5.5 V tolerant)	
RB8	8	input (5.5 V tolerant)	
RB7	7	input (5.5 V tolerant)	
RB5	5	input (5.5 V tolerant)	
RB4	4	output	relay output 5 on K8055(N)
RB3	3	output	relay output 4 on K8055(N)

RB2	2	output	relay output 3 on K8055(N)
RB1	1	output	relay output 2 on K8055(N)
RB0	0	output	relay output 1 on K8055(N)

PIC32MZ2048ECH100 port A pin assignment:

PIC32 pin name	port A bit number	Direction	Description
RA1	1	input	used as A/D input channel 1
RA0	0	input	used as A/D input channel 0

PIC32MZ2048ECH100 port B pin assignment:

PIC32 pin name	port B bit number	direction	description
RB15	15	Input	~SS1 (SPI)
RB14	14	input	SDO1 (SPI)
RB13	13	input (5.5 V tolerant)	I3 on K8055(N)
RB12	12	input	NSS (SPI)
RB11	11	input (5.5 V tolerant)	I1 on K8055(N)
RB7	7	output	relay output 7 on K8055(N)
RB5	5	output	relay output 6 on K8055(N)
RB4	4	output	relay output 5 on K8055(N)
RB3	3	output	relay output 4 on K8055(N)
RB2	2	output	relay output 3 on K8055(N)
RB1	1	output	relay output 2 on K8055(N)
RB0	0	output	relay output 1 on K8055(N)

PIC32MZ2048ECH100 port D pin assignment:

PIC32 pin name	port B bit number	direction	description
RD15	15	output	SDO1 (SPI)
RD14	14	Input	SDI1 (SPI)
RD13	13	input (5.5 V tolerant)	I3 on K8055(N)
RD12	12	input	NSS (SPI)
RD10	10	input (5.5 V tolerant)	I5 on K8055(N)
RD9	9	input (5.5 V tolerant)	I4 on K8055(N)
RD5	5	input (5.5 V tolerant)	I2 on K8055(N)
RD1	1	output	SCK1 (SPI)
RD0	0	output	relay output 1 on K8055(N)

PIC32MZ2048ECH144 port A pin assignment:

PIC32 pin name	port A bit number	Direction	Description
RA1	1	input	used as A/D input channel 1
RA0	0	input	used as A/D input channel 0

PIC32MZ2048ECH144 port B pin assignment:

PIC32 pin name	port B bit number	direction	description
RB14	14	Input with internal PULL-UP	micro switch 3 on PIC32
RB13	13	input with internal PULL-UP	micro switch 2 on PIC32
RB12	12	input with internal PULL-UP	micro switch 1 on PIC32
RB7	7	output	
RB5	5	output	
RB4	4	output	
RB3	3	output	
RB2	2	output	
RB1	1	output	
RB0	0	output	

PIC32MZ2048ECH144 port H pin assignment:

PIC32 pin name	port B bit number	direction	description
RH2	2	output	Green LED
RH1	1	output	Yellow LED
RH0	0	output	Red LED

g. Emulated PIC18F2xJ50 port access functions

PIC32 FW and PIC32MX2X0 programming library support PIC18F2xJ50 port access functions emulation. These functions are intended to ease conversion of existing VB.NET applications to PIC32 microcontrollers.

Here is an example on how to set one of the digital outputs on port B:

```
ListBox1.Items.Add(CStr(PIC.SetDigitalOutputs(data)))
```

Data is an 8-bit value that specifies the output values of all 8 channels. To clear or set only one of the channels a simple binary mathematics may be used. It is also possible to read back the value of the port B. Here is an example:

```
ListBox1.Items.Add("port B = " + CStr(Hex(PIC.ReadDigitalInputs(Asc("B")))))
```

ReadDigitalInputs function can in fact read from ports “A”,”B” and “C”. Port “A” and “C” are initially programmed as follows:

simulated port A = all inputs (8-bits)

simulated port B = all outputs (8-bits)

simulated port C = all outputs, except for RC1 and RC2 which are outputs

RA0 and RA1 bits of the port A bits are initially connected to AN0 and AN1 and they are configured as analog inputs. A 10-bit A/D converter with multiplexor reads the voltage (0 .. Vdd) in them. The values of both inputs can be read as follows:

```
ListBox1.Items.Add(CStr(PIC.ReadAnalogInput(CByte(TbAddress.Text))))
```

RC0 and RC1 bits of the port C are programmed as PWM1 and PWM2 (PWM = pulse width modulator) outputs. You can build your own circuit to convert those outputs to D/A 1 and D/A 2 that convert impulses to the average voltage between +0 V and +5 V. If you use Velleman's K8055 or K8055N board the D/A outputs are already implemented. PWM resolution is 10-bits, which also applies to D/A conversion outputs. Here is an example on how to use PWM modules:

```
ListBox1.Items.Add(CStr(Hex(PIC.SetPWM(1023, 500))))
```

Function SetPWM sets two 10-bit values (decimal range from 0 to 1023) for PW1 and PWM2 accordingly.

The list of the supported functions by the SVPICAPI class is the following:

```
Open(devID as Integer) as Integer                    ' devID = 0..&hFFFF (default = &hD001)
SetDigitalOutputs(portB as Byte) as Integer   ' sets digital outputs
ReadAnalogInput(ADchannel as Byte) as Integer ' ADchannel=0..15
ReadDigitalInputs(port as String) as Integer    'port = asc('A'), asc('B') or asc('C')
'int SetDefaultOutputValue(char port, byte lat); 'port = asc('A'), asc('B') or asc('C'), lat = 'default value = 0..255
'.. See Microchip documentation
GetDefaultOutputValues(port as String) as Integer ' reads default values
Close() as Integer                                    ' closes communication
DataMemRead(addr as UInt16) as Integer          ' reads 1 byte from RAM (16-bit addressing), &hF81 emulation
DataMemWrite(addr as_UInt16, data as Byte) as Integer ' writes 1 byte to RAM (16-bit addressing) , &hF81
emulation
```

h. Virtual port bit access to a memory location

PIC32 microcontrollers no longer support bit access operations on an arbitrary memory location. Instead they use "**set"** and "**clear"** shadow registers for each control register. Writing an 8-bit, 16-bit or 32-bit word to a "set" shadow register is some kind of a logical OR operation. Only the bits that have value "1" get written to the original register. "**clear**" shadow register works the opposite way. Only the bits with value "0" get reset in the original register, other bits retain their values.

LIB_PCUSBProjects library v6.2.NET4 and PIC32 FW nevertheless support some bit operations that are intended to ease the transition from a PIC18 to a PIC32 microcontroller.

```
GetPortMemAdr(port as Byte) as Uint16;       // returns PIC18F2xJ50 virtual port memory address
```

The function returns 16-bit PIC18F24J50 memory address where port resides. There are three ports: "A", "B" and "C". Example:

```
Adr = GetPortMemAdr(Asc("B"))
```

NOTE: Adr can only have value of &hF81, since only port B is supported. All other values will return error core -99 (not supported).

SetDataMemBit(uint16 adr, byte bitNumber) as Integer

The function sets a bit on desired PIC18F2xJ50 memory address.

ClearDataMemBit(uint16 adr, byte bitNumber) as Integer

The function clears a bit on desired PIC18F24J50 memory address.

```
SetPin(port as Byte, data as Byte) as Integer
ClearPin(port as Byte, data as Byte) as Integer
```

NOTE: Only port B is supported. Any value other than Asc("B") will produce error code -99 (not supported).

If you omit port letter, port B is presumed.

int SetPin(byte data);
int ClearPin(byte data);

Examples:

SetPin(Asc("B"),0)

SetPin(0)

It is clear that if port parameter is omitted the port B is presumed. This port is configured as output port on Velleman K8055.

i. Transfering data arrays between DLL and VB.NET applications

This section has been moved to chapter 3: PIC18 and PIC32 Commnon Functions, section: **Transfering data arrays between DLL and VB.NET applications**.

j. Software microcontroller restart

PIC32 microcontroller can be restarted by an external input, or programmatically by calling **PIC32_Restart** function from a VB.NET application. Restarting the microcontroller is required, if it's basic configuration should be reloaded.

PIC32_Restart() as Integer

The restart command is successfully issued, if the return value is zero. After the restart the microcontroller will automatically reconnect to USB 2.0.

k. Advanced A/D converter functions (PIC32MX only)

PIC32_ADC_CaptureBlock() as Integer
PIC32_ADC_ReadBlockLocation(addr as UInt16, cnt as UInt16) as Integer
PIC32_ADC_GetConfiguration(void) as Integer ' Reads PIC32 ADC configuration to buffer
PIC32_ADC_SetConfiguration(void) as Integer ' Writes a new ADC configuration from buffer to PIC32

PIC32 A/D converter enables sampling frequencies up to 1 MHz. It has a 10 input analog multiplexer. Therefore the maximum number of input channels is 10. Sampling at a high rate requires the use of the microcontroller's RAM for buffering. PIC32 firmware communicated with a VB.NET application through exchange of data arrays. **PIC32_ADC_SetConfiguration** function can be done through the following procedure:

PIC.WriteBufferWORD(0, &HE4) ' Value of ADCON1 register (see Microchip Data Sheet DS61168E)
PIC.WriteBufferWORD(2, &H41E) ' Value of ADCON2 register (see Microchip Data Sheet DS61168E)
PIC.WriteBufferWORD(4, &H400) ' Value of ADCON3 register (see Microchip Data Sheet DS61168E)
PIC.WriteBufferWORD(6, &HFA0) ' 16-bit size of A/D capture buffer
PIC.WriteBufferWORD(8, &HFA0) ' 16-bit initial value of the buffer counter, must be less than A/D capture buffer size
PIC.PIC32_ADC_SetConfiguration()' Set configuration from the write buffer…

WriteBufferWORD function is described in chapter i. Transfering data arrays between DLL and VB.NET applications. **PIC32_ADC_GetConfiguration** returns the stored configuration from PC32MX microcontroller to the PC read buffer. Use **ReadBufferWORD** commands to read data.

After the setup is done, we are ready to capture a block of samples to PIC32 internal RAM 8 kB buffer. Each 10-bit sample is represented by 16-bits for easier handling. Use **PIC32_ADC_CaptureBlock** function to instruct PIC32 microcontroller to capture an 8 kB (or less) block of samples to its 8 kB buffer. The last step is transferring the captured data to the PC, which is accomplished with **PIC32_ADC_ReadBlockLocation** function. The function enables PC to read the contents of the PIC32 internal buffer with HID command. **PIC32_ADC_ReadBlockLocation** function enables reading parts of A/C converter capture buffer in blocks that contain a maximum of 30 16-bit words, which can be transferred to the PC in a single USB request reply.

NOTE: PIC32_ADC_ReadBlockLocation function transferee data PC internal buffer. Use **ReadBufferWORD** function described in chapter i. Transferring data arrays between DLL and VB.NET to read a block of data to your VB.NET application. Here is a sample function that downloads the PIC32 capture buffer contents to the drawing buffer of the oscilloscope sample application that you can fine in the PC USB Projects website.

```
Function Load_Draw_Buf() As Boolean
    On Error GoTo err1
    Dim n As Int16
    Dim m As Int16 = 0
    PIC.PIC32_ADC_CaptureBlock()
    For m = 0 To ADC_blk_cnt - 1
      PIC.PIC32_ADC_ReadBlockLocation(m * ADC_max_blksz + ADC_blk_start_offset, ADC_max_blksz)
      For n = 1 To ADC_max_blksz
        drawBuf((n - 1) + m * ADC_max_blksz) = PIC.ReadBufferWORD(n * 2 + 1)
      Next
    Next
    Return True
    Exit Function
err1:
    Return False
  End Function
```

l. I²C bus master functions

PIC32 firmware v2.9.1 includes the following functions to enable I²C bus master mode communication with slave devices for I²C bus 0:

PIC32_I2C_Read(adr as Byte, REGadr as Byte) as Integer
PIC32_I2C_ReadWord(adr as Byte, REGadr as Byte) as Integer
PIC32_I2C_Write(adr as Byte, REGadr as Byte, dta as Byte) as Integer
PIC32_I2C_GeneralReset() as Integer

PIC32 firmware v2.9.1 includes the following functions to enable I²C bus master mode communication with slave devices for I²C bus 0 (pins RB8 and RB9) and I²C bus 1 (pins RB2 and RB3):

PIC32_I2C_Read(bus_num as Byte, adr as Byte, REGadr as Byte) as Integer
PIC32_I2C_ReadWord(bus_num as Byte, adr as Byte, REGadr as Byte) as Integer
PIC32_I2C_Write(bus_num as Byte, adr as Byte, REGadr as Byte, dta as Byte) as Integer
PIC32_I2C_GeneralReset(bus_num as Byte) as Integer

PIC32_I2C_Read function takes two mandatory parameters: **external I²C device address** and **I²C device internal register address** and an optional parameter **I²C bus number**. If I_2C bus number is omitted, I²C bus 0 is presumed. The device address must be unique for each device on the bus, whereas internal register address is a device specific. If the function returns a positive value including zero, the value represents the I²C device internal register value. Negative values are error codes that indicate failed communication.

PIC32_I2C_Write function writes a new value to the chosen I^2C device internal register. If the writing succeeds the zero value is returned, otherwise a negative error represents an error code. PIC32_I2C_Write function also has the optional bus number parameter.

PIC32_I2C_GeneralReset sends a 0x06 command to address 0, which causes all devices on I^2C bus to reset. The reset operation waits for 200 µs for the reset operation to complete. If a device needs a longer reset time, user application should wait for additional time until the device is ready to process I^2C commands. PIC32_I2C_Write function also has the optional bus number parameter.

NOTE: The functions automatically determine, if PIC32 I^2C bus master have already been configured. The configuration is set, if not previously configured.

m. ADT7410 16-bit external temperature sensor functions

The following functions support ADT7410 temperature sensor operation:

PIC32_ADT7410_SingleSampleMode(adr as Byte) as Integer
PIC32_ADT7410_GetTemperature(adr as Byte) as Double
PIC32_ADT7410_SetConfig(adr as Byte, cnt as Byte, mode as Byte) as Integer
PIC32_ADT7410_GetConfig() as Integer
PIC32_ADT7410_Enable() as Integer
PIC32_ADT7410_Disable() as Integer
PIC32_ADT7410_GetTemperatures(cnt as Byte) as Integer
PIC32_ADT7410_StartSampling() as Integer
PIC32_ADT7410_Reset(adr as Byte) as Integer

Three parameters are used with the listed functions: **Adr** parameter represents a chosen temperature sensor number, which is also its address offset. The actual sensor address on I^2C bus is calculated from the base address. **Cnt** parameter contains the number of ADT7410 sensors on the I^2C bus. The preset base address is 48 (hex). And the preset number of attached sensors is one. Both presets may be altered by **PIC32_ADT7410_SetConfig** function, which also enables one to set ADT7410 configuration by setting the **mode** parameter. The latter is transferred directly to the configuration register of ADT7410 sensor. The preset mode parameter value is A0 (hex). Read the sensor information sheet for more information. The configuration may be read back to a PC buffer by **PIC32_ADT7410_GetConfig** function**.**

PIC32_ADT7410_SingleSampleMode executes a single temperature sampling. The temperature value remains stored in ADT7410 temperature registers. **PIC32_ADT7410_GetTemperature** function reads the temperature value from ADT7410 to PIC32 and then relays it to the PC via USB communication.

The remaining ADT7410 functions are used in double buffering mode. **Double buffering mode** supports up to 4 temperature sensors on I^2C bus 1 (PIC32 has two I2C buses, but currently only the first one is supported). Double buffering mode requires **PIC32_ADT7410_StartSampling** function to be called first to acquire the initial temperature reading with **PIC32_ADT7410_GetTemperature** function, or prior to enabling thermostatic control with **PIC32_ADT7410_Enable** function.

Thermostatic control also requires a valid I^2C configuration. The configuration is automatically set as needed at the start of PIC32 communication to ADT7410 temperature sensor.

PIC32_ADT7410_Reset enables reset of a particular I^2C device. This function operation resets internal registers of ADT7410 to default values. It is particularly useful when the ADT7410 stops responding to commands via I^2C bus due to a hardware or software error.

n. Thermostatic and watchdog control functions

PIC32 firmware supports virtually the same thermostatic functions as the PIC18F2xJ50 firmware. Its additional feature is a hardware watchdog support which enables restarting the microcontroller, if it becomes unresponsive. Watchdog is triggered after approximately 4 seconds (4096 miliseconds), if its counter is not reset by the microcontroller firmware. There is also an option of a visual LED indicator that blinks when the microcontroller operates property. See Default thermostatic and watchdog control values in the end of this subsection for details.

The thermostatic functions operate on virtual port B and they use A/D converter channels to read the temperature values or stored temperature values from 16-bit ADT7410 digital sensors.

NOTE: Values of A/D converter channels higher than 127 (i.e. 7F (hex)) represent ADT7410 the numbers of ADT7410 sensors. If a number higher than 127 is entered then <number-128> ADT7410 sensor value is used. However, ADT7410 sensor must be set to use double buffering. Read ADT7410 16-bit external temperature sensor functions section for more information.

There are also 6 functions that support thermostatic control operation:

ReadBufferWORD(adr as Byte) as Integer
WriteBufferWORD(adr as Byte, data as UInt16) as Integer
LoadThermostaticControlValues() as Integer
StartThermostaticControl() as Integer
StopThermostaticControl() as Integer
ReadBackThermostaticControlValues() as Integer

ReadBufferWORD and **WriteBufferWORD** are used in conjunction with functions **ReadBuffer** and **WriteBuffer** to access read and write buffers for PIC18Fxxxxx and PC block data exchange. The new functions enable data word operations on the buffers. A data word of 16-bits is written to the write buffer or read from the read buffer low byte first.

Function **LoadThermostaticControlValues** loads thermostatic control configuration from buffer to the PIC18F26J50 microcontroller (16 bytes). The configuration is composed as follows:

1 byte ….. A/D A input for ambient temperature value
1 byte ….. A/D B input for internal refrigerator temperature value
1 byte ….. Relay for refrigerator power control (0 = on/ 1 = off)
1 byte ….. Relay for refrigerator heater control (0 = on/ 1 = off)
2 bytes … Ambient switch on threshold temperature A/D A value
2 bytes … Ambient switch off threshold temperature A/D A value
2 bytes … Internal: compressor switch off threshold temperature A/D B value
2 bytes … Internal: heating switch on threshold temperature A/D B value
2 bytes … Internal: heating switch off threshold temperature A/D B value

Here is also an example on how to load the thermostat configuration and start thermostatic control:

```
PIC.WriteBuffer(0, 1)  ' sensor A = A/D 1
PIC.WriteBuffer(1, 0)  ' sensor B = A/D 0
PIC.WriteBuffer(2, 0)  ' Relay A = bit = 0
PIC.WriteBuffer(3, 1)  ' Relay B = bit = 1
PIC.WriteBufferWORD(4, 550) ' Thermostat enable threshold temperature A/D A value – sensor A
PIC.WriteBufferWORD(6, 500) ' Thermostat disable threshold temperature A/D A value – sensor A (Relays off)
PIC.WriteBufferWORD(8, 400) ' Relay 0 on temperature – sensor B
PIC.WriteBufferWORD(10, 450) ' Relay 0 off temperature – sensor B
PIC.WriteBufferWORD(12, 250) ' Relay 1 on temperature – sensor B
PIC.WriteBufferWORD(14, 300) ' Relay 1 off temperature – sensor B
ListBox1.Items.Add(CStr(PIC.LoadThermostaticControlValues()))
ListBox1.Items.Add(CStr(PIC.StartThermostaticControl()))
```

Function **ReadBackThermostaticControlValues** retrieves thermostat configuration from the microcontroller and stores if into the read buffer. It can be read with ReadBuffer commands. Here is an example:

```
ListBox1.Items.Add("R = " + CStr(PIC.ReadBackThermostaticControlValues()))
For n = 0 To 3
   ListBox1.Items.Add(Hex(PIC.ReadBuffer(n)))
Next
For n = 4 To 14 Step 2
   ListBox1.Items.Add(CStr(PIC.ReadBufferWORD(n)))
Next
```

Functions **StartThermostaticControl** and **StopThermostaticControl** take no parameters. They only instruct the microcontroller to start or stop automatic thermostat control.

Default thermostatic and watchdog control values are also stored in PIC32 EEPROM on address &h**BD01E800** in the following structure:

PIC32MX:

Address (hex)	Description
BD01E800	Startup mode = { 0 = themostatic controll is off, 1 = thermostatic controll start starts in ADC mode, 2 = thermostatic controll starts in ADT7410 mode}
BD01E802	Thermal sensor 0 ={ 0…1F = ADC input, 0x80 … 0x83 = ADT7410 at I2C addresses 0x48 to 0x52}
BD01E804	Thermal sensor 1 ={ 0…1F = ADC input, 0x80 … 0x83 = ADT7410 at I2C addresses 0x48 to 0x52}
BD01E806	Relay 0 output select = {0...7}. The pouts correspond to Q1..Q8 outputs on K8055(N) board and RB0..RB4 and RB13, RB14 and RB15 pins on the microcontroller chip.
BD01E808	Relay 1 output select = {0...7}. The pouts correspond to Q1..Q8 outputs on K8055(N) board and RB0..RB4 and RB13, RB14 and RB15 pins on the microcontroller chip.
BD01E80A	Thermostat enable threshold temperature A/D A value – sensor A
BD01E80C	Thermostat disabled threshold temperature A/D A value – sensor A (Relays off)
BD01E80E	Relay 0 on temperature – sensor B
BD01E810	Relay 0 off temperature – sensor B
BD01E812	Relay 1 on temperature – sensor B
BD01E814	Relay 1 off temperature – sensor B
BD01E816	Buffer 0 location initial temperature value for sensor A (thermostat enabled)
BD01E818	Buffer 1 location initial temperature value for sensor A (thermostat enabled)
BD01E81A	Buffer 0 location initial temperature value for sensor A (thermostat disabled)
BD01E81C	Buffer 1 location initial temperature value for sensor A (thermostat disabled)
BD01E81E	Number of temperature sensors (1 = default, max. = 4)
BD01E820	Proper operation indication enable (0 = disable, 1 = enable, default = 0)
BD01E822	Watchdog enable (0 = disable, 1 = enable, default = 0)

PIC32MZ:

Address (hex)	Description
BD07E800	Startup mode = { 0 = themostatic controll is off, 1 = thermostatic controll start starts in ADC mode, 2 = thermostatic controll starts in ADT7410 mode}
BD07E802	Thermal sensor 0 ={ 0…1F = ADC input, 0x80 … 0x83 = ADT7410 at I2C addresses 0x48 to 0x52}
BD07E804	Thermal sensor 1 ={ 0…1F = ADC input, 0x80 … 0x83 = ADT7410 at I2C addresses 0x48 to 0x52}
BD07E806	Relay 0 output select = {0...7}. The pouts correspond to Q1..Q8 outputs on K8055(N) board and RB0..RB4 and RB13, RB14 and RB15 pins on the microcontroller chip.
BD07E808	Relay 1 output select = {0...7}. The pouts correspond to Q1..Q8 outputs on K8055(N) board and RB0..RB4 and RB13, RB14 and RB15 pins on the microcontroller chip.
BD07E80A	Thermostat enable threshold temperature A/D A value – sensor A
BD07E80C	Thermostat disabled threshold temperature A/D A value – sensor A (Relays off)

BD07E80E Relay 0 on temperature – sensor B
BD07E810 Relay 0 off temperature – sensor B
BD07E812 Relay 1 on temperature – sensor B
BD07E814 Relay 1 off temperature – sensor B
BD07E816 Buffer 0 location initial temperature value for sensor A (thermostat enabled)
BD07E818 Buffer 1 location initial temperature value for sensor A (thermostat enabled)
BD07E81A Buffer 0 location initial temperature value for sensor A (thermostat disabled)
BD07E81C Buffer 1 location initial temperature value for sensor A (thermostat disabled)
BD07E81E Number of temperature sensors (1 = default, max. = 4)
BD07E820 Proper operation indication enable (0 = disable, 1 = enable, default = 0)
BD07E822 Watchdog enable (0 = disable, 1 = enable, default = 0)

The default configuration can be altered with EEPROM write functions that are described in the next chapter.

o. EEPROM programming

PIC32MC250F128B v2.9.1 firmware supports the following two functions for writing data directly to its internal EEPROM and one function that starts execution of a new code:

```
PIC32_WriteEEPROM(addr as UInt32, data as UInt32) as integer
PIC32_ErasePageEEPROM(addr as UInt32) as integer
```

The function **PIC32_WriteEEPROM** enables writing 32-bits of data (**data** parameter) to an arbitrary empty EEPROM location (**addr** parameter). An empty 32-bit EEPROM location always holds unprogrammed value of &h**FFFFFFFF**. If the location holds a different value, it must be first erased with **PIC32_ErasePageEEPROM** function. However, it is important to stress that **PIC32_ErasePageEEPROM** function erases 1024 bytes at a time. It the desired data location is located in an area where other data is stored, the whole EEPROM page must be read with **MemRead** command describe in chapter d. Direct PIC32 memory access. Then the desired data must be altered in the PC RAM and then the whole page must be transferred back to the PIC32 microcontroller with a series of **PIC32_WriteEEPROM** functions.

IMPORTANT: ADDRESS CONVERSION MUST BE USED!

PIC32 microcontroller CPU sees from 128 kB to 2 MB (depending on microcontroller family and type) internal EEPROM in the virtual address range from &h**BD000000** up to &h**BD0FFFFF**, while the EEPROM controller sees EEPROM in physical address space from &h**1D000000** to &h**1D0FFFFF**. Therefore **MemRead** function address for the EEPROM location must be converted to the address for the EEPROM controller prior to using **PIC32_WriteEEPROM** and **PIC32_ErasePageEEPROM** function. The conversion can be done with this simple function:

<EEPROM controller address> = < microcontroller CPU address> and &h**1FFFFFFF**

Function **PIC32_Jump** is unique, because it can start execution of firmware code on an arbitrary memory location (RAM or EEPROM). This enables testing of new firmware in RAM or reprogramming the whole EEPROM including Boot Flash from an USB compatible firmware that temporarily resides in RAM. All interrupts are disabled as a jump to new code is performed. This prevents malfunctions until new communication interrupts handles are ready to operate. The target firmware code must re-enable interrupts.

p. User RAM and EEPROM functions

PIC32MC250F128B v2.9.1 firmware supports the following code execution functions:

```
PIC32_Jump(address as UInt32) as Integer
PIC32_Call(address as UInt32) as Integer
```

```
PIC32_Call_b(address as UInt32, data as Byte) as Integer
PIC32_Call_w(address as UInt32, data as UInt16) as Integer
PIC32_Call(address as UInt32, data as UInt32) as Integer
```

PIC32_Jump function is unique, because it can start execution of firmware code on an arbitrary memory location (RAM or EEPROM). This enables testing of new firmware in RAM or reprogramming the whole EEPROM including Boot Flash from an USB compatible firmware that temporarily resides in RAM. All interrupts are disabled as a jump to new code is performed. This prevents malfunctions until new communication interrupts handles are ready to operate. The target firmware code must re-enable interrupts. The Jump function is intended to provide means to start an autonomous user code that never returns.

PIC32_Call functions are meant to run user functions. A **PIC32_Call** function call starts execution on a 32-bit address and makes an input parameter array (or a single parameter) and an output parameter array available to the user function. The function than loads parameters, executes and returns results in the output parameter array. Then it reverts back to PIC32 firmware. Function parameters are transferred both ways: form the firmware to the user function and backwards as byte arrays. Optionally you can also send a value to a scalar function directly in one parameter.

Use **WriteBuffer** and **WriteBufferWORD** functions (see section i. Transfering data arrays between DLL and VB.NET applications in this document for details) to enter the input parameter array before **PIC32_Call** function call. Use **ReadBuffer** and **ReadBufferWORD** functions upon returning from **PIC32_Call** function to read output parameter array (see section i. Transfering data arrays between DLL and VB.NET applications in this document for details).

User function definition

Microchip MPLAB (X) development environment enables you to write additional user functions for PIC32 to be executed from its firmware. Basic versions of MPLAB and MPLAB X environments and XC32 compiler are free. You can Download and install them from Microchip website (www.microchip.com).

A special **PIC32 LOADER v1.0 tool** (with source code) is provided to enable you to easily load a user function to PIC32 RAM. Later versions will also enable you to store a function to EEPROM. The tool source code is also provided. See EEPROM programming section (o) on details how to program your function to EEPROM. Only a few changes are needed to the original PIC32 LOADER v1.0 tool code.

There is also a **sample user function MPLAB project** (**PIC32 RAM function example.zip**) available from **Downloads section**. The starting RAM or EEPROM address is defined in linker definition file *.ld. The preset address is **A0005000**. Modify linker definition file and PIC32 LOADER v1.0 tool to use a different user function starting address.

If you need to program more than one user function, just place the functions on different memory locations. PIC32 firmware v2.9.1 only uses RAM above **A0003000** for A/D converter storage when block conversion functions are called. You should only place your functions in empty RAM or EEPROM locations.

q. Setting USB configuration

USB configuration is store in seven descriptors (the descriptors are described in detail on www.usb.org website):

PIC32MX:

Address	Description
BD01E000	Device descriptor
BD01E100	Configuration descriptor, Interface descriptor, HID Class-Specific Descriptor, Endpoint Descriptor 1, Endpoint Descriptor 2

BD01E200 Language code string descriptor
BD01E300 Manufacturer string descriptor
BD01E400 Product string descriptor
BD01E500 Class specific descriptor – HID

PIC32MZ:

Address	Description
BD07E000	Device descriptor
BD07E100	Configuration descriptor, Interface descriptor, HID Class-Specific Descriptor, Endpoint Descriptor 1, Endpoint Descriptor 2
BD07E200	Language code string descriptor
BD07E300	Manufacturer string descriptor
BD07E400	Product string descriptor
BD07E500	Class specific descriptor - HID

PIC32MX ONLY:

VID and PID values are stored in one 32-bit word on EEPROM address &h**BD01E008**. To alter them first the EEPROM page located on address range &h**BD01E000** through &h**BD01E3FF** must be read (**MemRead**) to the PC. Then location &h**BD01E008** has to be altered. The third step is erasing the whole EEPROM page (**PIC32_ErasePageEEPROM**) on the physical address range &h**1D01E000** through &h**1D01E3FF**, which is followed by programing the altered EEPROM page contents back (**PIC32_WriteEEPROM**).

See **o. EEPROM programming functions** chapter for details.

NOTE: When the USB configuration has been altered, it will only come into effect after the restart of the microcontroller. The easiest way to programmatically restart the microcontroller is to call the PIC32 restart function:

PIC32_Restart() as Integer

The other way is to disconnect the microcontroller and then plug it again. The restart command is successfully issued, if the return value is zero. After the restart the microcontroller will automatically reconnect to USB 2.0.

r. 8-channel low frequency PWM generator

Low frequency pulse width modulated signal (PWM) can be used for various purposes form lighting up the Christmas tree to driving robot servo motors. PIC32 firmware has special set of functions that enable up to 8 independent pulse width modulated channels. Each PWM channel is controlled by four parameters: length of low signal period, length of high signal period, the number of emulated port B output channel: values 0 through 7 (see chapter e. Emulated direct PIC18 memory access) and a Boolean variable that determine whether the channel is used.

Here are the functions prototypes:

PIC32_LowFreqGenConfiguration(channel as Byte) as Integer
PIC32_LowFreqGenConfiguration(cnt0_max as UInt32, cnt1_max as UInt32, relay_select as Byte, channel as Byte)
PIC32_LowFreqGen_Enable(channel as Byte) as Integer
PIC32_LowFreqGen_Disable(channel as Byte) as Integer

PIC32_LowFreqGenConfiguration function has two forms: The form with just one parameter read the selected PWM channel configuration to read buffer. Contains 10 bytes with four parameters as follows:

Location	Size	Description

0	4 bytes	selected PWM channel low phase length (**cnt0_max** parameter)
4	4 bytes	selected PWM channel high phase length (**cnt1_max** parameter)
8	1 byte	selected PWM channel output (**relay_select** parameter)
9	1 byte	selected PWM channel status (0 = disabled , 1 = enabled)

PIC32_LowFreqGenConfiguration function in four parameter form sets a new configuration for the chosen channel. The input data is of the same type as the data returned from the function **PIC32_LowFreqGenConfiguration** in one channel number parameter form.
The remaining pair for functions **PIC32_LowFreqGen_Enable** and **PIC32_LowFreqGen_Disable** enables or disables the selected channel.

Note: Consider chapter **i. Transfering data arrays between DLL and VB.NET applications** for more information on how to read or write data arrays to and from PIC32 in VB.NET.

s. Velleman K8055 and K8055N experiment board functions

There are also additional functions that ease the use of microcontrollers. Most of them operate are similarly or equally to original Velleman K8055D.DLL functions. Here is the list of the function definitions:

```
OpenDevice(CardAddress as Integer) as Integer
CloseDevice()
WriteAllDigital(Data as Integer)
SearchDevices() as UInt32
SetCurrentDevice(lngCardAddress as Integer) as Integer
ReadAnalogChannel(Channel as Integer) as Integer
OutputAnalogChannel(Channel as Integer, Data as Integer)
OutputAllAnalog(Data1 as Integer, Data2 as Integer)
ClearAnalogChannel(Channel as Integer)
ClearAllAnalog()
SetAnalogChannel(Channel as Integer)
SetAllAnalog()
ClearDigitalChannel(Channel as Integer)
ClearAllDigital()
SetDigitalChannel(Channel as Integer)
SetAllDigital()
ReadDigitalChannel(Channel as Integer) as Boolean;
ReadAllDigital() as Integer
ReadBackDigitalOut() as Integer
ResetCounter(CounterNr as Integer)
ReadCounter(CounterNr as Integer) as Integer
```

The most of the functions work in both advanced (new firmware from this website) and legacy (Velleman) mode. **OpenDevice** function calls **Open** function with a different default vendor ID and base product ID depending on the selected operation mode. If the advanced mode is selected (see **SetK8055LegacyMode** function description below) the vendor ID is 0x4D8 and the base product ID is 0xF001. If legacy mode is selected then vendor ID is 0x10CF and base product ID is 0x5500.

SearchDevices function is also dependent on the selected operation mode and the base product ID. It uses **IsConnected** function to find a maximum of 32 devices in advanced mode and 4 devices in legacy mode with product IDs starting from the base product ID.

SetCurrentDevice only works in legacy mode. It opens a connection to a microcontroller with 0x1CF vendor ID and product ID which is a sum of base product ID and **lngCardAddress**.

Functions **ResetCounter**, **ReadCounter** and **SetCounterDebounceTime** are only supported in legacy mode.

There are two additional functions to set and to check the currently set operating mode:

SetK8055LegacyMode(legacy_FW_mode as Boolean) as Integer
K8055LegacyMode() as Boolean

SetK8055LegacyMode function sets legacy mode, if the **legacy_FW_mode** parameter is **true** and advanced mode, if the parameter is **false**. The legacy mode mimics the communication of K8055D.DLL to the microcontroller. However, the advanced mode offers much greater functionality.

Some functions are also supported in K8055 legacy mode, but with some limitation. See SVLIB_PIC18F24J50 programming guide.

t. Asynchomous counter support

PIC32 microcontrollers have five counters T0 through T5. T3 and T5 timers may also be used as hardware asynchronous counters. The following functions are supported:

```
SetCounter(CounterNr as Integer, val as Integer) as Integer  ' Set counter initial
value
ResetCounter(CounterNr as Integer) as Integer         ' Resets counter to 0
ReadCounter(CounterNr as Integer) as Integer          ' Reads 16-bit counter value
```

u. PIC18F2xJ50 PROGRAMMER SUPPORT

The following PIC18F2xJ50 programmer functions are supported:

```
EnterICSPEEPROM(ICSP_entry_code as UInt32) as Integer
CommandEEPROM(cmd as Byte, data as UInt16, delay as Byte) as Integer
CommandResultEEPROM(cmd as Byte, delay as Byte) as Integer
WriteEEPROM(cmd as Byte, offset as Byte, cnt as Byte, delay as Byte) as Integer
ReadEEPROM(cmd as Byte, cnt as Byte) as Integer
ReadBuffer(adr as Byte) as Integer
WriteBuffer(adr as Byte, data as Byte) as Integer
```

Here are descriptions of EEPROM and flash RAM programming functions:

EnterICSPEEPROM(ICSP_entry_code as UInt32) as Integer

This function enables ICSP mode entry for Microchip PIC18FxxJxx family microcontrollers. These microcontrollers do not use the high voltage programming instead they enter program/verify mode while operating on normal power supply voltage. They use a special 32-bit code to prevent accidental entry into ICSP mode. PIC18FxxJ50 family ICSP entry code is: 4D434850 (hex).

CommandEEPROM(cmd as Byte, data as UInt16, delay as Byte) as Integer

This function contains all the necessary logic to send a command to a Microchip PIC microcontroller. Please, see Microchip programming documentation (www.microchip.com) for further information on programming the Microchip microcontrollers.

CommandEEPROM(cmd as Byte, data as UInt16, delay as Byte) as Integer

This function contains a standard procedure enables reading data from a microcontroller flash RAM location. It is used for the verification purpose after the programming or to read a microcontroller configuration word that identifies the microcontroller.

WriteEEPROM(cmd as Byte, offset as Byte, cnt as Byte, delay as Byte) as Integer

Data is written to the microcontroller flash RAM in blocks. The length of the blocks depends on the length of a microcontroller write buffer (ex. PIC18F2xJ50 microcontrollers have 64 bytes of buffer, while PIC18F14K50

microcontroller has only 16 bytes). There is also a post write delay parameter that is used to initiate the write buffer to flash RAM programming sequence. The PIC18Fxxxxx write buffer is accessed through the USB control and status report operations. Each operation had a limited data space and two or more USB operations are needed to transfer 64 bytes to the microcontroller. Therefore *offset* and *cnt* parameters are used to send a specified number of byte from the PC memory buffer to the PIC write buffer. The o*ffset* parameter enables correct positioning in the PC memory buffer.

ReadEEPROM(cmd as Byte, cnt as Byte) as Integer

This contains all the necessary logic to read one byte from a previously defined address in PIC flash RAM. The address is set using CommandEEPROM commands. See the sample PIC Programmer application in Downloads section for the details.

ReadBuffer(adr as Byte) as Integer

This function is used to read data from the PC buffer that is located in the DLL memory space to the user application buffer. See the sample PIC Programmer application in Downloads section for the details.

WriteBuffer(adr as Byte, data as Byte) as Integer

This function is used to write data to the PC buffer that is located in the DLL memory space to the user application buffer. See the sample PIC Programmer application in Downloads section for the details.

v. JTAG PROGRAMMER SUPPORT

JTAG programmer transfers data to a microcontroller via 4 wires. The protocol is message oriented and supports communication in both directions (IN and OUT) at the same time. Each output message is composed of a multibit TMS header and a multibit input value and the output value of the same length. Most messages for PIC24 and PIC32 programming are unidirectional, which means that only an input or an output value is relevant. The other value has no meaning. TMS header and the value are given by two parameters: bit length and a 16-bit or a 64-bit variable. The maximum length is determined by data type length. The maximum TMS length is therefore 16-bits and the maximum value length is 64-bits. The length parameter defines a number of bits (starting from bit 0) from the variable to be sent via JTAG interface. If the number for bits is 5, then only the first 5 bits of the variable will be sent.

Program the appropriate FW HEX file (**PIC32 v2.9.1 or later)** to your PIC32 microcontroller, include *SVLIB_PIC32MX2X0 v1.0.NET4(x64).DLL* in your project and add the following function to your JTAG programming application to able to quickly send and receive JTAG messages:

```
Function jtag_SendReceive(PIC As SVLib_PIC32MX2X0.SVPICAPI, tms_header_len As Byte, tms As UInt16,
val_len As Byte, val As UInt64, ByRef out As Int64) As Boolean
    On Error GoTo abort
    out = PIC.jtag_SendReceive(tms_header_len, tms, val_len, val)
    If out < 0 Then GoTo abort
    Return True
    Exit Function
abort:
    Return False
  End Function
```

There are also functions that support program executive upload:

```
ReadPIC32PE(pos as Integer) as Uint32      ' Reads PE value at desired location, which contains a 32-bit word.
GetPIC32PE_Size() as Integer                ' Returns PE size in 32-bit words.
ReadPIC32PELoader(pos as Integer) as Integer    ' Reads PE Loader halfword (16-bits)
GetPIC32PELoader_Size() as UInt32           ' Returns PE Loader size in 16-bit halfwords.
```

To download PE to a PIC32 microcontroller add the following function to your VB application:

```
    Function PIC32_DownloadPE(PIC As SVLib_PIC18F24J50.SVPICAPI) As Boolean

        ' Serial execution mode must already be entered!

        On Error GoTo abort
        Dim n As Integer
        Dim data As UInt64
        Dim out As UInt64


        '     If Not jtag_SendCommand(PIC, MTAP_INST_LEN, MTAP_SW_ETAP) Then GoTo abort ' <--- ICSP
mode only
        '    If Not jtag_SetMode(PIC, 6, &H1F) Then GoTo abort ' <--- ICSP mode only
        If Not jtag_XferInstruction(PIC, &H3C04BF88) Then GoTo abort
        If Not jtag_XferInstruction(PIC, &H34842000) Then GoTo abort
        If Not jtag_XferInstruction(PIC, &H3C05001F) Then GoTo abort
        If Not jtag_XferInstruction(PIC, &H34A50040) Then GoTo abort
        data = &HAC85
        data <<= 16
        If Not jtag_XferInstruction(PIC, data) Then GoTo abort
        If Not jtag_XferInstruction(PIC, &H34050800) Then GoTo abort
        data = &HAC85
        data <<= 16
        data = data Or &H10
        If Not jtag_XferInstruction(PIC, data) Then GoTo abort

        ' LW A1,0x40(A0)
        data = &H8C85
        data <<= 16
        data = data Or &H40
        If Not jtag_XferInstruction(PIC, data) Then GoTo abort

        ' ORI $A1,$ZERO,0x8000 (32k)
        'If Not jtag_XferInstruction(PIC, &H34058000) Then GoTo abort

        data = &HAC85
        data <<= 16
        data = data Or &H20
        If Not jtag_XferInstruction(PIC, data) Then GoTo abort
        data = &HAC85
        data <<= 16
        data = data Or &H30
        If Not jtag_XferInstruction(PIC, data) Then GoTo abort
        If Not jtag_XferInstruction(PIC, &H3C04A000) Then GoTo abort
        If Not jtag_XferInstruction(PIC, &H34840800) Then GoTo abort
        ' Step 6
        For n = 0 To PIC.GetPIC32PELoader_Size() - 1 Step 2
            data = &H3C060000 Or PIC.ReadPIC32PELoader(n)
            If Not jtag_XferInstruction(PIC, data) Then GoTo abort
            data = &H34C60000 Or PIC.ReadPIC32PELoader(n + 1)
            If Not jtag_XferInstruction(PIC, data) Then GoTo abort
            data = &HAC86
            data <<= 16
            If Not jtag_XferInstruction(PIC, data) Then GoTo abort
            If Not jtag_XferInstruction(PIC, &H24840004) Then GoTo abort
        Next

        ' execute

        If Not jtag_XferInstruction(PIC, &H3C19A000) Then GoTo abort
        If Not jtag_XferInstruction(PIC, &H37390800) Then GoTo abort
        If Not jtag_XferInstruction(PIC, &H3200008) Then GoTo abort
        If Not jtag_XferInstruction(PIC, &H0) Then GoTo abort

        ' Step 7 - A: Start PE loader

        Delay(100)  ' Allow loader to start

        If Not jtag_SendCommand(PIC, ETAP_CMD_LEN, ETAP_FASTDATA) Then GoTo abort

        data = &HA000
        data <<= 16
```

```
        data = data Or &H900

        If Not jtag_XferFastData(PIC, 32, data, out) Then GoTo abort ' Starting Address
        If Not jtag_XferFastData(PIC, 32, PIC.GetPIC32PE_Size, out) Then GoTo abort 'Size

        ' Step 7 - B:

        For n = 0 To PIC.GetPIC32PE_Size() - 1
            If Not jtag_XferFastData(PIC, 32, PIC.ReadPIC32PE(n), out) Then GoTo abort ' transfer 1 CPU
command
        Next

        If Not jtag_XferFastData(PIC, 32, 0, out) Then GoTo abort ' 0

        data = &HDEAD  ' magic word
        data <<= 16

        If Not jtag_XferFastData(PIC, 32, data, out) Then GoTo abort ' Send Magic sign to stop loading

        Delay(100)
        PEDownloaded = True
        Return True
        Exit Function
abort:
        Return False
    End Function
```

3. HIGH LEVEL FUNCTIONS FOR PIC18 AND PIC32 MICROCONTROLLERS

INTRODUCTION

A universal DLL (dynamic program library) for Windows simplifies the use of supported PIC microcontrollers. 32-bit or 64-bit versions of **LIB_PCUSBProjects v6.0.NET4 (or later)** are available from Downloads section. See sections 2.a, 2.b, 3.a, 3.b and 3.c for more information on opening connection and selecting your microcontroller type.

NOTE: This chapter mainly describes PIC18 and PIC32 common functions. Functions that are implemented for particular microcontrollers only, contain a note in red text.

Transfering data arrays between DLL and VB.NET applications

VB.NET applications communicate with **PIC18xxFxxxx**, **PIC18FxxKxx**, **PIC18FxxJxx** and **PIC32MXxxxFxxxB** firmwares through a set of functions that return scalar result values and a set of functions that return arrays. Five VB.NET functions are used to transfer data:

Function **ReadBuffer**(adr as Byte) as Integer
Function **WriteBuffer**(adr as Byte, data as Byte) as Integer
Function **ReadBufferWORD**(adr as Byte) as Integer
Function **WriteBufferWORD**(adr as Byte, data as UInt16) as Integer
Function **ReadBufferSingle**(byte adr) as Single

The first two functions are used for 8-bit data transfers, while the last two functions are intended for 16-bit data transfers. Both sets of functions work on the same read and write buffers. However, **ReadBufferWORD** and **WriteBufferWORD** functions read or write two consecutive 1-byte buffer locations and combine them into a 16-bit word.

Adr parameter represents a relative location address within a buffer, while **data** parameter contains a new value. All functions return an integer result. A negative value indicates an error. A positive value (including 0) represents actual data. **WriteBuffer** and **WriteBufferWORD** functions only return 0, if they execute correctly. **ReadBufferSingle** function reads a 4-bit floating point value in IEEE format from buffer and represents it as a single in VB.NET or float in VC.NET.

SPI1 support for PIC18

NOTE: YOU CAN ONLY USE RFM69CW AND SPI1 FUNCTIONS IN LIB_PCUSBProjects v5.3.NET4(x64).dll WITH PIC18F26J50 firmware v2.6.5 or later. RFM69CW Responder functionality is only supported with PIC18F26J50 firmware v2.6.7 or later

PIC18F2xJ50 microcontrollers have two MSSP (Master Synchronous Serial Port) interfaces (also referred as MSSPx): MSSP1 and MSSP2. MSSP1 can operate in either SPI or I2C mode, but MSPP2 only supports SPI mode due to lack of PORTD. Only a PIC18F4xJ50 microcontroller supports SPI and I2C modes on both MSSPs.

SPI is a three wire interface: SDO, SDI and SCK. The data is simultaneously transmitted and received on a master and a slave device. A special SSPxBUF register is used as a gateway to read and write bytes to SSPxSR shift register. When a byte of data on the master device is written to SSPxBUF register, it is copied to SSPxSR shift register and automatically transmitted to the slave device. As each bit is shifted out to the slave device a bit from the slave device is shifted into the SSPxSR register. After 8 shift operations the contents of the master SSPxSR register and the slave SSPxSR are exchanged.

PIC18 supports double buffering for data reception. Reading the register during the shift operation will return an 8-bit content being transmitted, but upon completion of the transfer then received value would be returned. Note: Only a write operation to SSPxBUF register will trigger data exchange between the master and the slave device. Even if the master device only needs to read one byte of data from the slave device, a dummy write operation (write of 0 to SSPxBUF register) must precede a read from SSPxBUF.

a. Setting up MSSPs

Getting the hardware SPI interface working on a PIC18 microcontroller may be quite tricky. The interface will only work, if the configuration is right. PIC18F2xJ50 microcontrollers use bits SPI1OD (0) and SPI2OD (1) in ODCON3 (Open Drain Control 3) register to enable open drain operation in SDO and SCK lines. Pull-up resistors may be used to achieve a higher voltage in SPI bus that is used for PIC18F2xJ50 power supply. This feature is rarely used. Though it is important to set SPI1OD and SPI2OD bit to 0, or no signal will be produced in SCK and SDO lines. This is the most common mistake made by beginners.

The next task is to properly set the PPS (Peripheral Pin Select) module. Though MSSP1 module has fixed connections to the PIC18F2xJ50 pins when enabled, its operation may still be obstructed by other remappable modules, like ECCP (Enhanced Capture, Compare and Pulse width modulation) module. Setting the values of RPOR8 and RPOR7, RPOR18 to zero will assure that the PORTB and PORTC pins used as SCK1, SDO1 and SDI1 would be controlled by MSSP1 module.

MSSP2 module has remappable SCK2, SDO2 outputs and an SDI2 input. PPS must be set appropriately to map MSSP2 to PIC18F2xJ50 outputs.

The next consideration is appropriate setup of port B and port C. Each of them has three control registers: PORTx, LATx and TRISx. The first contains the current output and input values. Latency register (LATx) is used when a corresponding bit in TRISx register is set to 0. If this causes the bit to turn in an output, an appropriate bit value from LATx register is used as a default value in PORTx register. So, first set the appropriate default value in LATx and then set TRISx register. For MSSP1 to operate properly, RB5 must be setup as an input, and RB4 and RC7 must be set as outputs. A similar setup is required for MSSP2.

Finally, don’t let the microcontroller’s A/D converter module to "spoil your day"! Though it might be logical to some, others may forget that PIC18F2xJ50's A/D converter may use all pins with ANx marks (see Microchip datasheet for PIC18F46J50 microcontroller family: DS39931D, pages: 5 and 349). ANCON0 and ANCON1 registers determine ANx pins operation. All the pins that are used by MSSPx should be programmed as digital input/outputs. This is achieved by setting the appropriate bits to 1.

b. Additional control pins

A device with SPI interface may also have additional control pins besides SPI interface pins. There are two such pins on HopeRF RFM69CW wireless communication module: Reset and NSS. Reset signal is used to restart RFM69CW without the need to power it off and on. NSS pin not used only as a chip select signal, but it also determines the data access mode. There are three access modes: Single, Burst and FIFO (first in first out buffer). Please see the module information sheet for details (look for RFM69CW-V1.1.pdf document with a web search engine, ex. www.Google.com) .

Make sure that all additional pins operate as normal port outputs, so they are not used by other PIC18 modules.

c. Data transfer

RFM69CW module uses SPI interface to enable serial access to its internal registers and FIFO. NSS pin is used as an additional signal to set data transfer mode for each transfer. NSS must be set to zero before each SPI access to enable SPI interface on RFM69CW. Next, the address byte is transferred, followed immediately by the data byte. RFM69CW does not support date exchange, so ti dummy read is performed during each write operation and a dummy write is required during read operation.

d. Software SPI interfaces

Software (SW) SPI interface is supported by PIC18F26J50 firmware v2.6.4 or later. It disables MSSP1 module and sets RB5 (SDI), PB4 (SCK) and RC7 (SDO) as normal port pins. All the SW SPI functions mimic the operation of HW SPI functions with “FOR” loops in the firmware subroutines.

SOFTWARE FUNCTIONS ARE INTENDED FOR TESTING HARDWARE CONNECTIONS AND DATA TRANSFER SPEED ONLY:

```
Function SWSPIInit() as Integer
Function SWSPIReadReg(addr as Byte) as Integer
Function SWSPIWriteReg(addr as Byte, dta as Byte) as Integer
```

NOTE: The software SPI interface is mainly intended for situations, where hardware SPI interface on MSSP1 cannot be used, because the required microcontroller pins are used for other purposed.

EXAMPLE 1: How to use software functions?

```
Dim SPI1InitFlag=False
Dim A as Integer

If Not SPI1InitFlag Then
   PIC.SWSPI1Init()
   SPI1InitFlag=True
End If
A = PIC.SWSPI1ReadReg(1)  ‘ A gets values of register 1
If A < 0 then
       Lb_Report.Items.Add(“Error”)
Else
       Lb_Report.Items.Add(“Register value = “+CStr(A))
End If
```

EXAMPLE 2: Direct PC communication with an SPI device for testing purposes

You can also use only MemRead and MemWrite intstructions to directly communicate with PIC18F26J50 registers, but it is very slow… Therefore the following example is only meant for reference on how the communication protocol actually works:

‘ SW SPI Initialzation

```
PIC.ClearDataMemBit(SSP1CON1, 5) ' disable ssp1
PIC.SetDataMemBit(ODCON3, 0) ' SPI1 Open drain OFF
PIC.DataMemWrite(CCP1CON, 0) ' Disable ECCP1
PIC.DataMemWrite(RPOR13, 0)   ' PPS RPOR13 = 0 --- RP13 = normal port, not ECCP1 port
PIC.DataMemWrite(LATB, PIC.DataMemRead(LATB) And &HCF)  ' LATB = LATB AND 0xCF --> RB5 = 0,
RB4 = 0
PIC.DataMemWrite(LATC, (PIC.DataMemRead(LATC) And &H7F) Or &H44) ' LATC = (LATC AND 0x7F)
OR 0x44 --> RC7=0, RC6=1, RC2=1
PIC.DataMemWrite(TRISB, (PIC.DataMemRead(TRISB) And &HEF) Or &H20)  ' TRISB  --> RB4 = SCK
= 0 , RB5 = SDI = 1
PIC.DataMemWrite(TRISC, PIC.DataMemRead(TRISC) And &H3B)  ' TRISC --> RC7 = SDO = 0, RC6 =
RESET = 0, RC2 = NSS = 0
PIC.SetDataMemBit(PORTC, 2)   ' NSS = 1
PIC.SetDataMemBit(PORTC, 6)   ' Reset = 1
Delay(100) ' delay in ms
PIC.ClearDataMemBit(PORTB, 4) ' SCK = 0
PIC.ClearDataMemBit(PORTC, 7) ' SDO = 0
addr = CByte(TbAddress.Text)
PIC.ClearDataMemBit(PORTC, 6)   ' Reset = 0
Delay(100) ' delay in ms
HW SPI Initialzation
... Use the same code as for SW SPI and add the following:
PIC.DataMemWrite(SSP1STAT,&h40)
PIC.DataMemWrite(SPP1CON,&h22)

‘ Read from a RFM69CW register

Function SWSPI1ReadReg(addr as Byte)
 Dim n as Byte
 Dim m as Byte
 Dim tr as Byte=addr
 PIC.ClearDataMemBit(PORTC,2)
 m=&h80
 for n=0 to 7   ' WRITE BYTE TO SPI
  if(tr&m)>0 then
     PIC.SetDataMemBit(PORTC,7)
  else
     PIC.ClearDataMemBit(PORTC,7)
  endif
  PIC.SetDataMemBit(PORTB,4)
  PIC.ClearDataMemBit(PORTB,4)
  m>>=1
 }
 m=&h80
 tr=0
 for n=0 to 7  ' READ BYTE FROM SPI
  PIC.SetDataMemBit(PORTB,4)
  if (PORTB&0x20)>0 then
   tr=tr or m
  end if
  PIC.ClearDataMemBit(PORTB,4)
  m>>=1;
 }
 PIC.SetDataMemBit(PORTC,2) ' NSS=1
 return tr
```

```
}

‘ Write to a RFM69CW register

Function SWSPI1ReadReg(addr as Byte, dta as Byte)
 Dim n as Byte
 Dim m as Byte
 Dim tr as Byte=addr or &h80
 PIC.ClearDataMemBit(PORTC,2)
 m=&h80
 for n=0 to 7   ' WRITE BYTE TO SPI
  if(tr&m)>0 then
     PIC.SetDataMemBit(PORTC,7)
  else
     PIC.ClearDataMemBit(PORTC,7)
  endif
  PIC.SetDataMemBit(PORTB,4)
  PIC.ClearDataMemBit(PORTB,4)
  m>>=1
 }
 tr=dta
 m=&h80
 for n=0 to 7   ' WRITE BYTE TO SPI
  if(tr&m)>0 then
     PIC.SetDataMemBit(PORTC,7)
  else
     PIC.ClearDataMemBit(PORTC,7)
  endif
  PIC.SetDataMemBit(PORTB,4)
  PIC.ClearDataMemBit(PORTB,4)
  m>>=1
 }
}
```

‘ CONSTANTS (for PIC18F2xJ50 microcontrollers):

```
' ECCPx

Const CCP1CON As UInt16 = &HFBA ' CCP1 PIC18FxxJxx
Const CCPR1L As UInt16 = &HFBB
Const CCPR1H As UInt16 = &HFBC
Const CCP2CON As UInt16 = &HFB4 ' CCP2 PIC18FxxJxx
Const CCPR2L As UInt16 = &HFB5
Const CCPR2H As UInt16 = &HFB6

' Ports

Const TRISA As UInt16 = &HF92
Const TRISB As UInt16 = &HF93
Const TRISC As UInt16 = &HF94
Const LATA As UInt16 = &HF89
Const LATB As UInt16 = &HF8A
Const LATC As UInt16 = &HF8B
Const PORTA As UInt16 = &HF80
Const PORTB As UInt16 = &HF81
Const PORTC As UInt16 = &HF82

' Open drain control

Const ODCON3 As UInt16 = &HF40

' PPS

Const RPOR13 As UInt16 = &HED3
```

```
' SSP

Const SSP1CON1 As UInt16 = &HFC6
Const SSP1CON2 As UInt16 = &HFC5   ' I2C only
Const SSP1STAT As UInt16 = &HFC7
Const SSP1BUF As UInt16 = &HFC9

' Interrupt

Const PIR1 As UInt16 = &HF9E
```

e. Hardware SPI functions in PIC18F26J50 firmware v2.6.5 or later

HARDWARE FUNCTIONS

```
Function SPI1Init() as Integer
Function SPI1ReadReg(addr as Byte) as Integer
Function SPI1WriteReg(addr as Byte, dta as Byte) as Integer
Function SPI1Write(dta as UInt16) as Integer
Function SPI1BurstReadReg(addr as Byte, size as Byte) as Integer
Function SPI1BurstWriteReg(addr as Byte, size as Byte) as Integer
```

There are two sets of three functions that enable initialization of the microcontroller SPI interface, and single read and write access to HopeRF RFM69CW wireless communication module registers, where the address is 7-bit RFM69CW internal register address.

The initialization functions set HW SPI operation and the appropriate SPI operating mode, or or SW SPI operation. Get PIC18F26J50 - SPI example (x64) from Downloads section for more information.

EXAMPLE 1: OPEN CONNECTION AND READ DATA

```
Dim SPI1InitFlag=False
Dim A as Integer

If Not SPI1InitFlag Then
   PIC.SPI1Init()
   SPI1InitFlag=True
End If
A = PIC.SPI1ReadReg(1)  ‘ A gets values of register 1
If A < 0 then
       Lb_Report.Items.Add(“Error”)
Else
       Lb_Report.Items.Add(“Register value = “+CStr(A))
End If
```

EXAMPLE 2: OPEN CONNECTION AND WRITE DATA

```
Dim SPI1InitFlag=False
Dim A as Integer

If Not SPI1InitFlag Then
   PIC.SPI1Init()
   SPI1InitFlag=True
End If
If (PIC.SPI1WriteReg(1,4)<0) Then  ‘ A set 4 to register 1
       Lb_Report.Items.Add(“Error”)
Else
       Lb_Report.Items.Add(“Register value set to 4“)
End If
```

Software I2C functionality in PIC18F26J50 firmware

NOTE: YOU NEED PIC18F26J50 firmware 2.6.10 or later to use this functionality.

Sometimes, it is impossible to use hardware support for I2C bus due to unavailability of the required PIC18F26J50 pins. But I2C protocol can also be implemented on any four of free microcontroller pins. PIC18F26J50 firmware v2.6.10 or later supports parameterized I2C support on and two pins of any microcontroller port. The configuration is stored in PIC18F26J50 EEPROM and may be altered with PC USB Projects HEX Editor v3.0 or later. One also needs an appropriate FWI (firmware information) file: PIC18F26J50 v4.fwi or later. See USB Projects HEX Editor user guide for more information.

The I2C settings are stored in the following EEPROM table:

Field Address (HEX)	Field Name	Default Value	Description
A400	SWI2C_PORT	0x0F80 (PORTA)	I/O port data register address
A402	SWI2C_TRIS	0x0F92 (TRISA)	I/O port tristate register address
A404	SWI2C_LAT	0x0F89 (LATA)	I/O port latency register address
A406	SWI2C_MCLK	0x0000 (RA0)	ECCP2 is used to provide clock on MCLR (used for HP03MA barometer sensor)
A408	SWI2C_SCL	0x0001 (RA1)	Master data clock
A40A	SWI2C_SDA	0x0002 (RA2)	I/O Data
A40C	SWI2C_XCLR	0x0003 (RA3)	XCLR I2C device reset
A40E	SWI2C_MCLK_CCP2_AUTOSTART	0x0001	0 = auto-start disable , 1 = auto-start enable

I2C also depends on common register values presets that are stored in the following table:

Field Address (HEX)	Field Name	Default Value	Description
A480	default_LATA	0x00	default LATA register value
A481	default_LATB	0x00	default LATB register value
A482	default_LATC	0x00	default LATC register value
A483	default_TRISA	0xF4	default TRISA register value
A484	default_TRISB	0x00	default TRISB register value
A485	default_TRISC	0xF9	default TRISC register value
A486	default_TCLKCON	0x00	default TCLKCON register value
A487	default_ANCON0	0xFF	default ANCON0 register value
A488	default_ANCON1	0xFF	default ANCON1 register value
A489	default_T1CON	0x03	default T1CON timer 1 control register value
A48A	default_T2CON	0x05	default T2CON timer 2 control register value
A48B	default_T4CON	0x04	default T4CON timer 4 control register value
A48C	default_PR2	0x5B	default PR2 timer 2 period length register value
A48D	default_PR4	0xFF	default PR4 timer 4 period length register value
A48E	default_CCP1CON	0x00	Default CCP1CON ECCP 1 control register
A48F	default_CCP2CON	PWM_setup	0xC = general version, 0xF K8055 adapter version → predefined in FW
A490	default_PWM_setup	PWM_setup	0xC = general version, 0xF K8055 adapter version → predefined in FW
A491	default_CCPR1L	0x00	default CCPR1L ECCP1 period register

			value
A492	default_CCPR2L	0x00	default CCPR2L ECCP2 period register value
A493	default_PSTR1CON	0x04	Default PSTR1CON ECCP1 PWM steering register value
A494	default_PSTR2CON	0x04	Default PSTR2CON ECCP2 PWM steering register value
A495	default_ ECCP1DEL	0x80	Default ECCP1DEL ECCP1 initial output value
A496	default_ ECCP2DEL	0x80	Default ECCP2DEL ECCP2 initial output value
A497	default_PL_CCP2H	0x00	PWM pulse length value in CCP2H register
A498	default_PL_CCP2L	0xAA	PWM pulse length value in CCP2H register
A499	default_RPOR0	0x14	RP0 output mapping
A49A	default_RPOR12	0x10	RP12 output mapping
A49B	default_RPOR13	0x00	RP13 output mapping
A49C	default_RPINR8	0x00	RP08 input mapping

The following new functions are added to support I2C operation:

```
Function SWI2C_Init() as Integer
Function SWI2C_WriteReg(adr as Byte, reg as Byte, dta as Byte) as Integer
Function SWI2C_ReadReg(adr as Byte, reg as Byte) as Integer
Function SWI2C_WriteRegWord(adr as Byte, reg as Byte, dta as Uint16) as Integer
Function SWI2C_ReadRegWord(adr as Byte, reg as Byte) as __Int32
```

Init function initializes all the register values og the desired port for I2C operation. High logical state is achieved by setting the SDA output to tristate and letting pull-up resistors to set high values, while low logical state is represented by voltage close to 0 V by setting SDA output to logical value 0.

Though the external devices my acknowledge any data without the need for the program to care about the right timing.

There are two kinds of register write and read functions: **SWI2C_WriteReg** writes 1 byte of data to the external device, while **SWI2C_WriteRegWord** writes two bytes to two consecutive device registers. The other pair of functions: **SWI2C_ReadReg** and **SWI2C_ReadRegWord** reads one and two consecutive bytes from an external I2C device.

EXAMPLE 1: Reading air pressure from HP03MA sensor

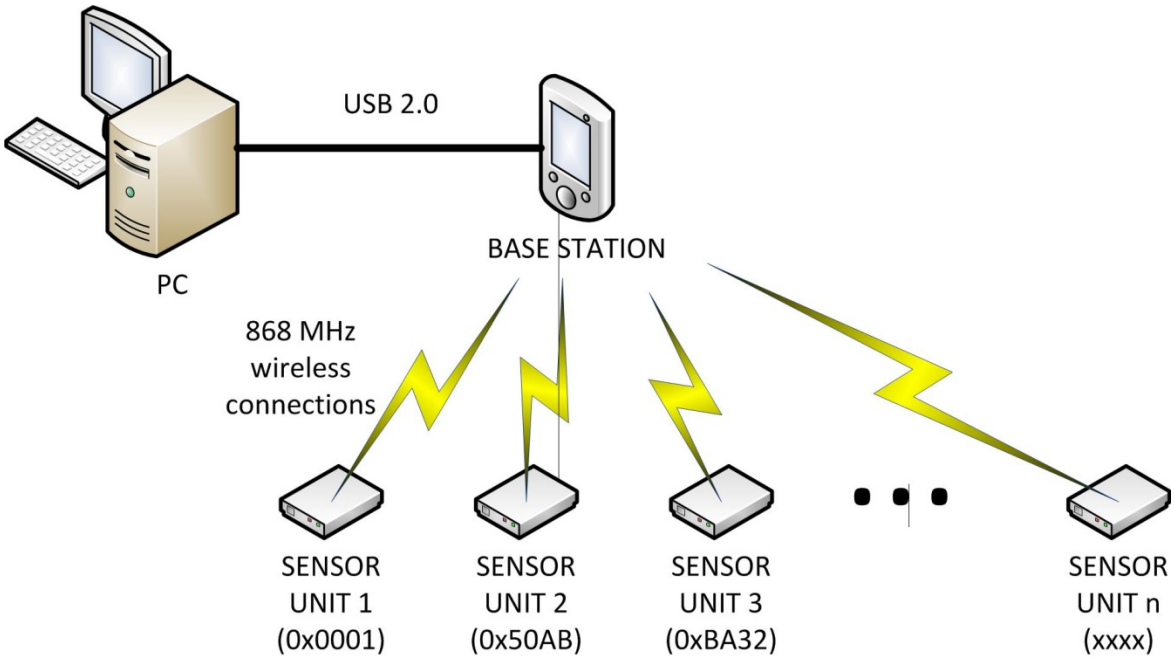
```
Function ReadAP_D1() As Int32    ' Air Pressure
    Dim w As Int32
    If PIC.SWI2C_WriteReg(&H77, &HFF, &HF0) = 1 Then
        Delay(40) ' Wait for 40 ms
        w = PIC.SWI2C_ReadRegWord(&H77, &HFD)
        If w < 0 Then
            Return -1
            Exit Function
        End If
    Else
        Return -1
    End If
    Return w
End Function
```

Weather Station Functions for PIC18

NOTE: YOU CAN ONLY USE THE FUNCTIONS BELOW WITH LIB_PCUSBProjects v6.2.NET4(x64).dll AND PIC18F26J50 firmware v2.7.6 or later

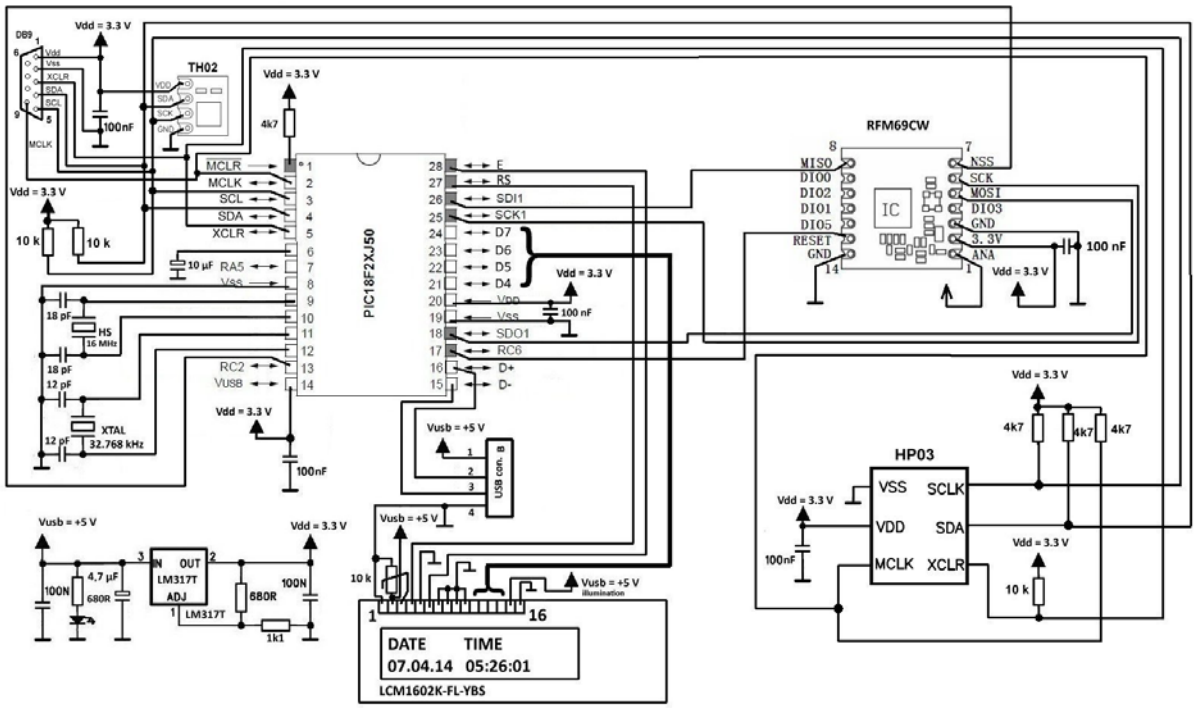
There are a number of (home) weather stations on the World's consumer electronics market, but none supports as much functionality as PC USB Projects Modular Weather Station. The weather station is based on a modular concept, which allows one to connect as many wireless external measurement units as needed to the base station that doubles as a hub and has a USB connection to (home) control center or a home PC.

Each remote measurement unit (RMU, also denoted as sensor unit) has a wireless transceiver and a number of sensors. The sensors values are transferred on demand to the base station. The later relies on round robin polling model to collect data from all the measurements units in real time. The data is also available on a PC via USB connection.

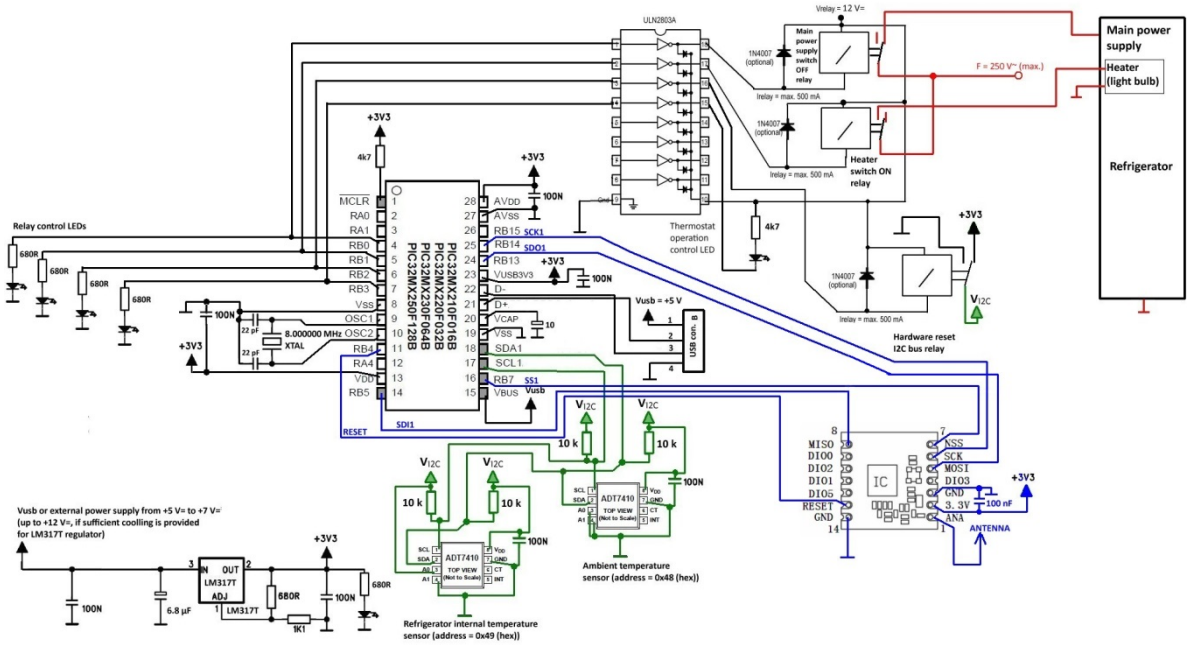


MODULAR CONCEPT SCHEMATIC

There are three kinds of sensors supported on measurements units: temperature sensors (Advance Devices ADT7410), air pressure sensors and temperature sensors (HopeRF HP03MA) and humidity and temperature sensors (HopeRF TH02).



BASE STATION SCHEMATIC



REMOTE MEASUREMENT UNIT (RMU) SCHEMATIC

Modular Weather Station is a developing project. The new software examples and applications are made available from Downloads section as soon as they are completed. They cover various Weather Station hardware functionalities, but not necessarily all of them. Occasionally all the functionalities will be covered as the software is fully developed. All the application software source code is also published on PC USB Projects Downloads section.

The current version of Weather Station control application (v2.9) supports all the sensors attached to the base station (Weather Station v2.9 (x64).zip). The v2.9 example reads buffered sensor values or

final temperature, humidity and air pressure values gathered by PIC18F26J50. Now, it is really easy to read the current values into a PC:

```
PIC.WeatherStationGetData()
temperature_from_air_pressure_sensor_as_single = PIC.ReadBufferSingle(0)
air_pressure_as_single = PIC.ReadBufferSingle(4)
temperature_from_humidity_sensor_as_single = PIC.ReadBufferSingle(8)
humidity_as_single = PIC.ReadBufferSingle(12)
temp_balcony_as_single = PIC.ReadBufferSingle(16)
temp_refrigrator_as_single = PIC.ReadBufferSingle(20)
temp_room1_as_single = PIC.ReadBufferSingle(24)
temp_room2_as_single = PIC.ReadBufferSingle(28)
```

The last four temperatures are gathered from RMUs. Up to 2 RMUs are supported by PIC18F26J50 firmware v2.7.6.

Alternatively, you can use PIC.**WeatherStationGetRawData**() function and make calculations yourself. Both options are included in v2.9 example.

Weather Station control application example v2.6 Weather Station v2.6 x64.zip is still available to demonstrate a way for immediate access to sensors. In fact it enables a PC to gather current temperature, air pressure and humidity values in real time.

Gathering data from remote units is currently supported by a different sample application (Digital thermometer and thermostat (ADT7410) v9 x64 with RFM69CW SPI connection example.zip), but will be soon integrated.

Time can be set and displayed on the weather station with PIC18F2xJ50 Digital Clock with LCD application.

NOTE: You have to program PIC18F26J50 firmware v2.7.6 to your microcontroller before you use Weather Station control application. Get PIC18F26J50 firmware v2.7.6 - universal.zip from PC USB Projects Downloads section.

If PIC18F26J50 firmware v2.7.6 is configured for weather station operation, it starts displaying room temperature, air pressure and humidity automatically immediately after the weather station is plugged in a USB port, or powered by a USB charger.

a. Weather Station Functions

Weather station automatic operation is controlled by the following three functions:

```
Function WeatherStationEnable(BYTE enable) as Integer
Function WeatherStationGetRawData() as Integer
Function WeatherStationGetData() as Integer
```

WeatherStationEnable function sets a Weather station operation mode. The operation modes are the following:

- 0 – Stop automatic Weather Station data processing
- 1 – Initialize Weather Station

2 – Initialize and start Weather Station

If Weather Station is running in automatic data processing mode, it will stop if enable variable is set to 0. If the variable is set to 1, it will prepare for operation, including sensors initializations, but it will not start to perform automatic data processing. Full initialization and start of automatic data processing will only be performed, when enable variable is set to 2.

b. Using raw data

WeatherStationGetRawData function returns the following data array:

Buffer Offset (DEC)	Data type	Value	Description
0	**byte**	Weather Station Enabled	0 = no, 1 = yes
1	**byte**	Weather Station Initialized	0 = no, 1 = yes
2	**word (2 bytes)**	**C(1)**	air pressure sensor constant used for temperature and relative air pressure calculation
4	**word (2 bytes)**	**C(2)**	air pressure sensor constant used for temperature and relative air pressure calculation
6	**word (2 bytes)**	**C(3)**	air pressure sensor constant used for temperature and relative air pressure calculation
8	**word (2 bytes)**	**C(4)**	air pressure sensor constant used for temperature and relative air pressure calculation
10	**word (2 bytes)**	**C(5)**	air pressure sensor constant used for temperature and relative air pressure calculation
12	**word (2 bytes)**	**C(6)**	air pressure sensor constant used for temperature and relative air pressure calculation
14	**word (2 bytes)**	**C(7)**	air pressure sensor constant used for temperature and relative air pressure calculation
16	**byte**	**AX**	air pressure sensor constant used for temperature and relative air pressure calculation
17	**byte**	**BX**	air pressure sensor constant used for temperature and relative air pressure calculation
18	**byte**	**CX**	air pressure sensor constant used for temperature and relative air pressure calculation
19	**byte**	**DX**	air pressure sensor constant used for temperature and relative air pressure calculation
20	**word (2 bytes)**	Humidity sensor temperature reading	This is 14-bit raw data. It has to be divided by 32 and then 50 must be subtracted to get the temperature in st. C
22	**word (2 bytes)**	Humidity sensor humidity reading	This is 12-bit raw data. It must be

			converted to a relative humidity value by two polynomials.
24	**word (2 bytes)**	**D1** value from air pressure sensor	The value is used in current temperature and air pressure calculation.
26	**word (2 bytes)**	**D2** value from air pressure sensor	The value is used in current temperature and air pressure calculation.
28	**word (2 bytes)**	Relative air pressure compensation value	Compensation value must be used to normalize the air pressure to the sea level. If compensation is not used, the air pressure is displayed as an absolute value.

The data is accessed by array transfer functions described in the beginning of this chapter in section: **Transferring data arrays between DLL and VB.NET applications**.

EXAMPLE1: HOW TO READ HUMIDITY AND AIR PRESSURE SENSOR RAW VALUES

```
'Constants

Dim HC_A0 As Double = -4.7844
Dim HC_A1 As Double = 0.4008
Dim HC_A2 As Double = -0.00393
Dim HC_Q0 As Double = 0.1973
Dim HC_Q1 As Double = 0.00237

' Variables

Dim AP_C(7) As Double
Dim AP_AX As Double
Dim AP_BX As Double
Dim AP_CX As Double
Dim AP_DX As Double
Dim AP_AirPressure As Double = -10000
Dim AP_Temperature As Double = -1000
Dim temp As Double
Dim temp_hex As UInt16
Dim hum As Double
Dim hum_hex As UInt16
Dim hum_linear As Double
Dim hum_comp As Double
Dim D1 As UInt16
Dim D2 As UInt16
Dim T As Double
Dim P As Double
Dim dUT As Double
Dim OFF As Double
Dim SENS As Double
Dim X As Double
…

PIC.WeatherStationGetRawData()

' Humidity Sensor

temp_hex = PIC.ReadBufferWORD(20)
temp = temp_hex
temp = (temp / 32) - 50

If AP_Temperature > -1000 Then  ' HP03 has more accurate temperature measurement
    temp = AP_Temperature
End If
```

```
hum_hex = PIC.ReadBufferWORD(22)
hum = hum_hex
hum = (hum / 16) - 24
hum_linear = hum - (hum ^ 2 * HC_A2 + hum * HC_A1 + HC_A0)
hum_comp = hum_linear + (temp - 30) * (hum_linear * HC_Q1 + HC_Q0)

' Air pressure sensor

For n = 1 To 7
    AP_C(n) = PIC.ReadBufferWORD(2 + (n - 1) * 2)
Next

AP_AX = PIC.ReadBuffer(16)
AP_BX = PIC.ReadBuffer(17)
AP_CX = PIC.ReadBuffer(18)
AP_DX = PIC.ReadBuffer(19)

D1 = PIC.ReadBufferWORD(24)
D2 = PIC.ReadBufferWORD(26)

AP_compensation = PIC.ReadBufferWORD(28)

If D2 >= AP_C(5) Then
  dUT = D2 - AP_C(5) - ((D2 - AP_C(5)) / 2 ^ 7) * ((D2 - AP_C(5)) / 2 ^ 7) * AP_AX / 2 ^
P_CX
Else
  dUT = D2 - AP_C(5) - ((D2 - AP_C(5)) / 2 ^ 7) * ((D2 - AP_C(5)) / 2 ^ 7) * AP_BX / 2 ^
AP_CX
End If

OFF = (AP_C(2) + (AP_C(4) - 1024) * dUT / 2 ^ 14) * 4
SENS = AP_C(1) + AP_C(3) * dUT / 2 ^ 10
X = SENS * (D1 - 7168) / 2 ^ 14 - OFF
P = (X * 10 / 2 ^ 5 + AP_C(7) + AP_compensation) / 10
T = (250 + dUT * AP_C(6) / 2 ^ 16 - dUT / 2 ^ AP_DX) / 10

AP_Temperature = T
AP_AirPressure = P
```

c. Reading temperature, air pressure and humidity from PIC18F26J50

WeatherStationGetData function offers a much simpler way of reading temperature, relative humidity and relative air pressure. Four 32-bit floating point values are returned and no further calculations are needed:

Buffer Offset (DEC)	Data type	Value	Description
0	**dword (4-bytes)**	Air pressure sensor (HP03) temperature	This is an absolute temperature in st. C measured by air pressure sensor
4	**dword (4-bytes)**	Air pressure sensor (HP03) relative air pressure	This is a relative air pressure measured by air pressure sensor and normalized to the sea level (compensation constant)
8	**dword (4-bytes)**	Humidity sensor (TH02) temperature	This is an absolute temperature in st. C measured by humidity sensor
12	**dword (4-bytes)**	Humidity sensor (TH02) relative air humidity	This is a relative humidity value in % measured by the humidity sensor

EXAMPLE1: HOW TO READ HUMIDITY AND AIR PRESSURE SENSOR FINAL VALUES

```
Dim temp_s As Single
Dim hum_s As Single
Dim temp_hs_s As Single
Dim ap_s As Single
…

PIC.WeatherStationGetData()

temp_s = PIC.ReadBufferSingle(0)
ap_s = PIC.ReadBufferSingle(4)
temp_hs_s = PIC.ReadBufferSingle(8)
hum_s = PIC.ReadBufferSingle(12)
```

d. Setting firmware constants

PIC18F26J50 firmware v2.7.6 or later has the following inbuilt defaults that may be altered by a programmer using **PC USB Projects HEX Editor v3.0 tool**:

Field Address (HEX)	Field Name	Default Value	Description
A540	Weather Station – Initialize/Start	0x00 (general use) 0x02 (weather station use)	If the original HEX-file is modified for Weather Station use this value is set to 0x02.
A541	Humidity sensor present	0x01	Determines, if humidity sensor (TH02) is attached to the microcontroller.
A542	Air pressure sensor present	0x01	Determines, if air pressure (HP03) sensor is attached to the microcontroller.
A543	Air pressure altitude compensation in **hPa** (2-byte value)	0x01B3	The value of 435 (decimal) is the air pressure compensation value for Ljubljana, Slovenia, Europe. **You should a compensation value for the place, where you intend to measure air pressure. Alternatively, you can also set this parameter to 0 and display absolute air pressure in hecto Pascal (hPa).**
A545	I^2C External Display Address	0x30	0 = no external display attached to I2C bus <other values> = 7-bit external address range start. 8 consecutive addresses are used for external display (default address range: 0x30 .. 0x37)
A546	Remote measurement unit (RMU) address	0x00AB	0 = first RMU is not attached <other values> = 16 – bit RMU 1 address
A548	Remote measurement unit 2 (RMU 2) address	0x0000	0 = second RMU is not attached <other values> = 16 – bit RMU 2 address

See **PC USB HEX Editor v3.0 User Guide** for more information about the **PC USB HEX Editor v3.0** tool. The tool is also included in the firmware ZIP file.

e. PIC18F26J50 remote GLCD (RGLCD) support

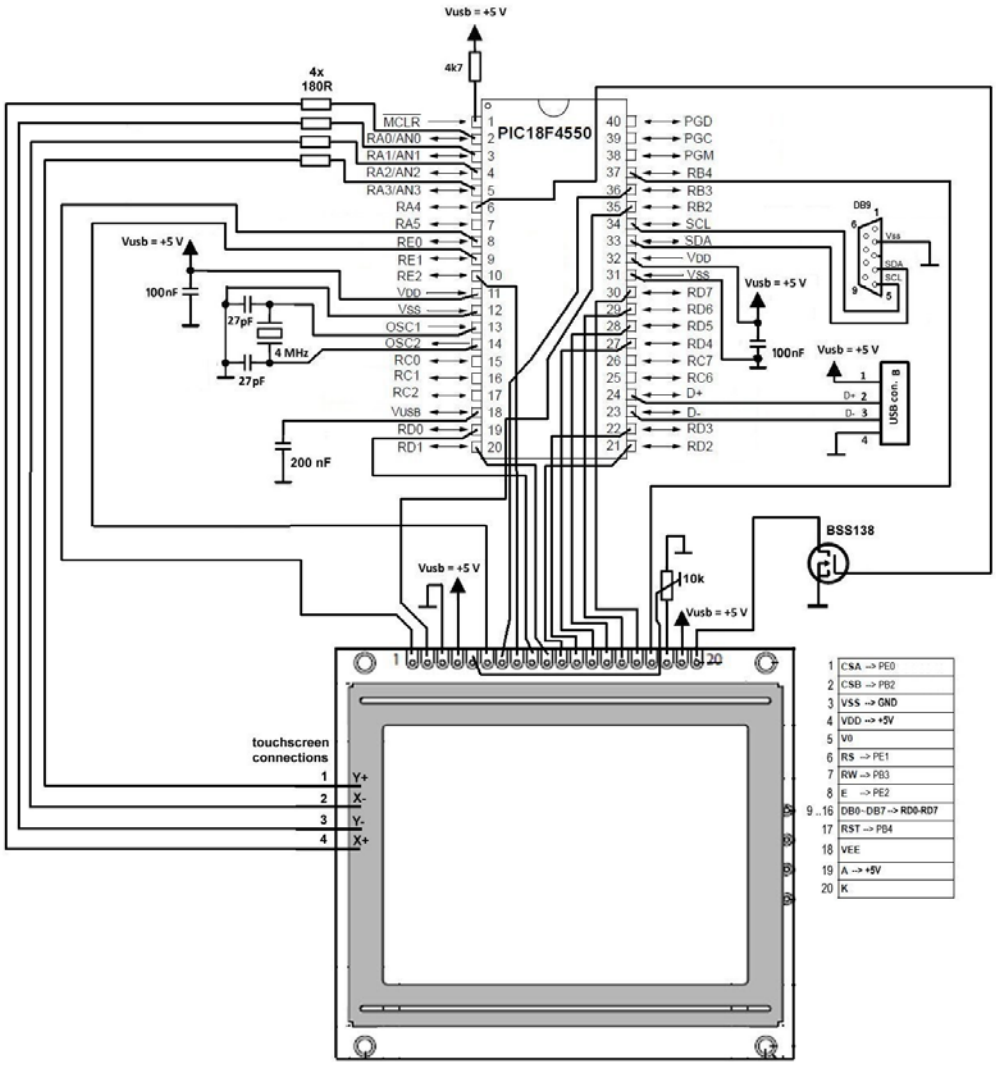
Sending commands via USB 2.0 bus is not the only way to control GLCD. The commands and data may also be transferred via a remote microcontroller I^2C bus. LIB_PCUSBProjects v6.2.NET4(x64).dll has the following additional functions to support remote GLCD operation from a VB.NET application:

Function **RGLCD_Setup**(Adr As Byte, enable As Byte, Illumination As Byte) As Integer
Function **RGLCD_SetPosition**(Adr As Byte, X As Byte, Y As Byte) As Integer // Set current position
Function **RGLCD_WriteCharXY**(Adr As Byte, Ch As Byte, X As Byte, Y As Byte) As Integer
Function **RGLCD_WriteCharXY**(Adr As Byte, Ch As Byte, X As Byte, Y As Byte, Style As Byte, Font As Byte) As Integer
Function **RGLCD_WriteChar**(Adr As Byte, Ch As Byte) As Integer
Function **RGLCD_WriteChar**(Adr As Byte, Ch As Byte, Style As Byte) As Integer
Function **RGLCD_WriteChar**(Adr As Byte, Ch As Byte, Style As Byte, Font As Byte) As Integer
Function **RGLCD_WriteByteHEX**(Adr As Byte, Dta As Byte) As Integer
Function **RGLCD_WriteByteHEX**(Adr As Byte, Dta As Byte, Style As Byte) As Integer
Function **RGLCD_WriteByteHEX**(Adr As Byte, Dta As Byte, Style As Byte, Font As Byte) As Integer
Function **RGLCD_Print**(Adr As Byte, Row As Byte)As Integer
Function **RGLCD_Print**(Adr As Byte, Row As Byte, Style As Byte) As Integer
Function **RGLCD_Print**(Adr As Byte, Row As Byte, Style As Byte, Font As Byte) As Integer
Function **RGLCD_Fill**(Adr As Byte, Dta As Byte) As Integer
Function **RGLCD_Clear**(Adr As Byte)As Integer
Function **RGLCD_BufferChar**(Adr As Byte, Ch As Byte) As Integer
Function **RGLCD_SetBufferCounter**(Adr As Byte, Dta As Byte) As Integer
Function **RGLCD_ResetBufferCounter**(Adr As Byte) As Integer
Function **RGLCD_Draw**(Adr As Byte, X As Byte, Y As Byte, Colour As Byte) As Integer
Function **RGLCD_Line**(Adr As Byte,X1 As Byte, Y1 As Byte, X2 As Byte, Y2 As Byte, Colour As Byte) As Integer
Function **RGLCD_Rectangle**(Adr As Byte,X1 As Byte, Y1 As Byte, X2 As Byte, Y2 As Byte, Colour As Byte) As Integer
Function **RGLCD_Box**(Adr As Byte,X1 As Byte, Y1 As Byte, X2 As Byte, Y2 As Byte, Colour As Byte) As Integer
Function **RGLCD_GetTouchcScreenInfo**(Adr As Byte) As Integer
Function **RGLCD_DisplayRegisters**(Adr As Byte) As Integer

The functions are based on **SWI2C_WriteReg** and **SWI2C_ReadReg** instructions. Their operation is the same are operation of the equivalent USB 2.0 GLCD functions (without "R" as the first letter for their names). However, it is also possible to use GLCD with an arbitrary I^2C master device. I2C slave commands are described in the next chapter: Graphics LCD (LGM12864B) with touchscreen functions for PIC18.

Graphics LCD (LGM12864B) with touchscreen functions for PIC18

NOTE: Graphics LCD functions are currently only implemented in PIC18F4550 firmware v2.6.1 or later.



Schematic: GLCD to PIC18F4550 with touchscreen connection and I2C connection to the PIC18F26J50 weather station base on PIC18F26J50.

LGM12864B graphics LCD (GLCD) has no character generator, so all the ASCII characters has to be displayed in graphics. However, very different text sizes are possible. The character patterns have to be stored in a microcontroller that operates the GLCD.

GLCD needs a high speed data channel to the microcontroller. LGM12864B GLCD has an 8-bit bus with 6 control signals. A PIC18F4550 microcontroller wih GLCD is connected to a PC via USB 2.0, or to another microcontroller via I2C bus.

http://sites.google.com/site/pcusbprojects
simonvav@gmail.com

The GLCD features an optional touchscreen that can easily be easily mounted by sticky stripes that attach it to the frame of the GLCD. The touchscreen is connected to PIC18F4550 microcontroller through 4 pins that function as analog inputs and digital outputs. Reading position and pressure from the touchscreen is described in “How to connect a 4-wire resistance touchscreen to a PIC microcontroller?” article.

GLCD also provides negative voltage generator to drive LCD. Only a small trimmer is needed to set the display contrast. The display illumination can be switched on or off via a small MOS-FET transistor that is connected to RA4 pin of PIC18F4550.

a. Text and graphics functions

LIB_PCUSBProjects v6.2.NET4(x64).dll program library has a number of functions for communication with GLCD:

Function **GLCD_Setup**(enable **As Byte**, Illumination **As Byte**) **As Integer**
Function **GLCD_SetPosition**(x **As Byte**, y **As Byte**) **As Integer**
Function **GLCD_WriteData**(void) **As Integer**
Function **GLCD_ReadData**(void) **As Byte**
Function **GLCD_WriteData**(x **As Byte**, y **As Byte**) **As Integer**
Function **GLCD_ReadData**(x **As Byte**, y **As Byte**) **As Integer**
Function **GLCD_WriteChar**(ch **As Byte**, x **As Byte**, y **As Byte**) **As Integer**
Function **GLCD_WriteChar**(ch **As** B**yte**) **As Integer**
Function **GLCD_Fill**(dta **As Byte**) **As Integer**
Function **GLCD_Clear**(void) **As Integer**
Function **GLCD_Draw**(x **As Byte**, y **As Byte**, colour **As Byte**) **As Integer**
Function **GLCD_Line**(x1 **As Byte**, y1 **As Byte**, x2 **As Byte**, y2 **As Byte**, colour **As Byte**) **As Integer**
Function **GLCD_Rectangle**(x1 **As Byte**, y1 **As Byte**, x2 **As Byte**, y2 **As Byte**, colour **As Byte**) **As Integer**
Function **GLCD_Box**(x1 **As Byte**, y1 **As Byte**, x2 **As Byte**, y2 **As Byte**, colour **As Byte**) **As Integer**
Function **GLCD_WriteChar**(ch **As Byte**, x **As Byte**, y **As Byte**, style **As Byte**) **As Integer**
Function **GLCD_WriteChar**(ch **As Byte**, style **As Byte**) **As Integer**
Function **GLCD_Print**(cnt **As Byte**, x **As Byte**, y **As Byte**) **As Integer**
Function **GLCD_Print**(cnt **As Byte**, x **As Byte**, y **As Byte**, style **As Byte**) **As Integer**
Function **GLCD_GetTouchcScreenInfo**(void) **As Integer**
Function **GLCD_WriteControlRegister**(BYTE reg, BYTE dta) **As Integer**
Function **GLCD_ReadControlRegisters**(void) **As Integer**

Most functions have **x** and **y** parameters. **x** defines row number (0..8) and y defines horizontal position. **y** must be set between 0 and 15, so a maximum of sixteen 8x8 non-overlaying characters can be displayed in each row. **style** parameter defines text style. Four text styles are possible:

0 - NORMAL
1 - INVERTED NORMAL
2 - WIDE
3 - INVERTED WIDE

Let’s describe functions in more detail:

GLCD_Setup function prepares GLCD for operation. It must be called prior to other functions. GLCD operation can be enabled or disabled by **enable** parameter (0 = disable , 1 = enable). The next

parameter switches on or off the display backlight (**illumination** parameter: 0 = backlight off, 1 = backlight on).

GLCD_SetPosition sets new current position on display (x, y). The position is used with functions that write to display and do not set starting position.

GLCD_WriteData writes one byte of data to display. The second from of the function enables setting the starting position.

GLCD_ReadData reads one byte of data from display. Current position is updated: y = y + 1. The second from of the function enables setting the starting position.

GLCD_Fill fills all the display locations with the same byte value. If the value of 0 is specified, the display is cleared.

GLCD_Clear clears the display.

GLCD_Draw draws a single dot to the desired display position. Since the display is B/W only values 0 and 1 may be specified. In this case only: x = 0..63 (vertical position), y = 0..127 (horizontal position). This enables a programmer to directly program any LCD cell.

GLCD_Line draws a line to connect display positons (x1,y1) and (x2,y2). Since the display is B/W only values 0 and 1 may be specified. In this case only: x1 and x2 = 0..63 (vertical position), y1 and y2 = 0..127 (horizontal position). This enables a programmer to directly program any LCD cell.

GLCD_Rectangle draws a rectangle with upper left corner position (x1,y1) and lower right corner position (x2,y2). Since the display is B/W only values 0 and 1 may be specified. In this case only: x1 and x2 = 0..63 (vertical position), y1 and y2 = 0..127 (horizontal position). This enables a programmer to directly program any LCD cell.

GLCD_Box draws a filled box with upper left corner position (x1,y1) and lower right corner position (x2,y2). Since the display is B/W only values 0 and 1 may be specified. In this case only: x1 and x2 = 0..63 (vertical position), y1 and y2 = 0..127 (horizontal position). This enables a programmer to directly program any LCD cell.

GLCD_WriteChar function writes a single ASCII character to the desired position (x, y) on display. ASCII table with CP852 encoding is used. The function also enables selection of the character style. If style parameter is omitted, normal style is used.

GLCD_Print function writes ASCII characters from a byte character array to the single display row. If the style parameter is omitted, normal style is used for all character. The **cnt** (count) parameter specified the length of the character array. If the character count is greater than 15 the excessive characters are omitted.

GLCD_Print function (string parameter instead of character count and character array) for displaying strings must be further defined in VB.NET programming as:

```
Function GLCD_Print(s As String, x As Byte, y As Byte) As Boolean
    GLCD_Print(s, x, y, 0)
    Return True
End Function
```

```
Function GLCD_Print(s As String, x As Byte, y As Byte, style As Byte) as Boolean
    Dim n As Integer
    For n = 0 To s.Length - 1
        PIC.WriteBuffer(n, Asc(Mid(s, n + 1, 1)))
    Next
    PIC.GLCD_Print(s.Length, x, y, style)
    Return True
End Function
```

As one can see style parameter may be omitted, if normal style is used. This also assures backwards compatibility. You may copy the functions above to your program.

The functions only convert a string to a byte array. The latter is then transferred to the microcontroller via USB 2.0.

GLCD_GetTouchcScreenInfo function gathers and processes the information from the touchscreen. The information consists of three parameters: **x**, **y** and pressure. Pressure is returned as the function result, while **x** and **y** can be read from the communication buffer (see introduction for more information on transferring data arrays between the DLL and a VB.NET application) addresses 0 and 2 with the following program:

```
x_press = PIC.GLCD_GetTouchcScreenInfo()
Lb_XPres.Text = CStr(x_press)

If x_press > 100 Then

    '' Position X
    x_pos = PIC.ReadBufferWORD(0)
    xp = x_pos
    xp -= 270

    LbAD0.Text = CStr(xp)

    '' Position Y


    y_pos = PIC.ReadBufferWORD(2)
    yp = y_pos
    yp -= 325
    LbAD1.Text = CStr(yp)
Else
    LbAD0.Text = "--"
    LbAD1.Text = "--"
End If
```

The touchscreen positions can be converted to positions on GLCD (**xp** and **yp**) by setting the right offsets. The offsets depend on touchscreen and its physical position relative to the underlying GLCD.

b. EXAMPLE 1: GLCD DEMO v3 APPLICATION WITH SOURCE CODE

GLCD DEMO v3 programming example with source code (GLCD DEMO v3 (x64).zip) has been added.
Four text styles are now supported:
0 - NORMAL
1 - INVERTED NORMAL
2 - WIDE
3 - INVERTED WIDE

There are also 5 inbuilt character fonts:
0 - Normal1 (X1250)
1 - Normal2 (CP850)
2 - Normal3 (CP850)
3 - Italic (X850)
4 - Space (CP437)

X1250 character font has letters “č”, “š”, “ž”, “Č”,”Š” and ”Ž” from Slovenian alphabet arranged according to code page 1250.

CP850 character fonts

The example is based on the new LIB_PCUSBProjects v6.2.NET4(x64).dll. LIB_PCUSBProjects v6.0.NET4 header file is available from Downloads section. The new example supports advanced display and touchscreen control functions that provide fast access from PC to the display via USB 2.0.

c. EXAMPLE 2: GLCD HARDWARE TESTING APPLICATION

The first GLCD example application (PIC18F4550 - GLCD example.zip) is still available from Downloads section. The application shows all the basic principles of a microcontrollers to GLCD communication and also provides touch screen position (X,Y) and touch pressure reading. This application also enables one to test operation of hardware connections to the display since each pin may be tested individually.

d. I^2C bus remote GLCD slave commands

PIC18F4550 firmware 2.7.1 supports I2C slave functionality. Any I2C master device that supports I2C clock stretching can use GLCD functionality via I2C interface. PIC18F4550 firmware 2.7.1 has a default I2C address range from 0x30 to 0x37. The range may be moved with PC USB HEX Editor v3.0.

SWI2C_WriteReg(address As Byte, register number As Byte, value As Byte) function is used to send text data to the GLCD. See the table on the right hand side.

SWI2C_ReadReg(address as Byte, register number as Byte) function can optionally be used to read back a register content.

Examples:

PIC.SWI2C_WriteReg(0x30,0,1) .. sets current display x position to 1

PIC.SWI2C_WriteReg(0x30,1,2) .. sets current display x position to 2

PIC.SWI2C_WriteReg(0x31,0,0x41) .. writes normal letter A to current display (x,y) position and increases y by 1, if y<15

PIC.SWI2C_WriteReg(0x31,2,0x41) .. writes double sized letter A to current display (x,y) position and increases y by 2, if y<14

PIC.SWI2C_WriteReg(0x33,0,0) .. clears display

PIC.SWI2C_WriteReg(0x34,0,1) .. clears display and displays registers contents

Touchscreen support

Touchscreen has three 16-bit registers: pressure, x and y. The registers are accessible on address 0x30 (default setting) as registers 5,6 and 7. Register 5 holds current register number:

0 - pressure register
1 - x position register
2 - y position register

Each register holds a 10-bit value obtained by A/D conversion. Pressure value indicates the amount of pressure that is applied to the touch screen. If the amount of pressure is lower than threshold pressure, all the registers reset to 0. x position register holds a 10-bit value that corresponds to horizontal position of the point where pressure is applied to the touchscreen. y position register holds a 10-bit value that corresponds to vertical position of the point where pressure is applied to the touchscreen.

Both x and y positions are absolute and have to be calibrated in accordance with the GLCD useful display area. The default threshold is 100 and it can be altered with PC USB Projects HEX Editor v3.0.

PIC18F4550 firmware GLCD I2C slave registers:

Address:	Register:	Value:	Description:
0x30	0	0 ... 7	GLCD current x position (vertical, line number)
0x30	1	0 ... 15	GLCD current y position (horizontal)
0x30	2	0 ... 16	character buffer counter
0x30	3	0 ... 63	Pixel graphics: current x1 position (vertical)
0x30	4	0 ... 127	Pixel graphics: current y1 position (horizontal)
0x30	5	0 ... 2	Touchscreen register selection (0 = pressure, 1 = x, 2 = y)
0x30	6	0 ... 255	Touchscreen register high byte
0x30	7	0 ... 255	Touchscreen register low byte
0x30	8	0 ... 4	Current character fonts
0x30	9	0 ... 63	Pixel graphics: current x2 position (vertical)
0x30	10	0 ... 127	Pixel graphics: current y2 position (horizontal)
0x31	Style	ASCII code	Displays an ASCII character on (x,y) display position*
0x32	Style	byte	Displays a byte in HEX on (x,y) display position*
0x33	0	0 ... 255	Fills the display with an arbitrary byte value.0 clears the display.
0x33	1	0 ... 1	Turns GLCD on (1) or off (0).
0x33	2	0 ... 1	Turns GLCD backlight on (1) or off (0)
0x34	don't care	don't care	Display registers contents
0x35	don't care	ASCII code	Enter a character to the character buffer
0x36	0	0 ... 1 (colour)	Pixel graphics: Draw point at (x1,y1) position
0x36	1	0 ... 1 (colour)	Pixel graphics: Draw line from (x1,y1) to (x2,y2)
0x36	2	0 ... 1	Pixel graphics: Draw rectangle from (x1,y1) to (x2,y2) position
0x36	3	0 .. 1	Pixel graphics: Draw box from (x1,y1) to (x2,y2) position
0x37	Row (0..7)	Style	Prints buffered characters to a selected display row

Styles are:	Character fonts are:
0 - normal	0 - Normal1 (X1250)
1 - inverted normal	1 - Normal2 (CP850)
2 - double	2 - Normal3 (CP850)
3 - inverted double	3 - Italic (X850)
	4 - Space (CP437)

*The characters may be printed to display, if there is enough space available in the selected display line. The current y display position is automatically increased by 1 for styles 0 and 1 and by 2 for styles 2 and 3 and when a HEX value is displayed.

NOTE: Program PIC18F4550 firmware v2.7 to your PIC18F4550 to use the sample applications. The new firmware provides fast communication to GLCD and ASCII character patterns. The new firmware also supports reconfiguration with PC USB Projects HEX Editor v3.0 and a special PIC18F4550.FWI file. I2C bus slave support will also be provided in forth coming PIC18F4550 firmware versions.

SPI support for PIC32

NOTE: YOU CAN ONLY USE RFM69CW AND SPI1 FUNCTIONS WITH LIB_PCUSBProjects v6.2.NET4(x64).dll or later AND PIC32 firmware v2.9.1 or later

PIC32MX microcontrollers have two SPI (Serial Peripheral Interface) modules. Unlike PIC18's MSSPs they are fully automated and buffered. Two 16-byte buffers are provided in the enhanced buffer mode. Therefore no interrupt handlers are needed in FW for simple applications. The buffers depth depends on the selected operating mode in SPI1CON register: 8-bit, 16-bit or 32-bit. RFM69CW transceiver prefers 16-bit mode which enables a human programmers to send and receive data by sending a single 16-bit word. Upper 8-bits consist of 1-bit read/write flag followed by 7-bit register address. Lower 8-bits are data to be written to RFM69CW, or dummy data, if a RFM69CW register is read.

SPI is a three wire interface: SDO, SDI and SCK. The data is simultaneously transmitted and received on a master and a slave device. A special SSPxBUF hardware buffer is used as a gateway to read and write bytes to SSPxSR shift register. When a byte of data on the master device is written to SSPxBUF register, it is copied to SSPxSR shift register and automatically transmitted to the slave device. As each bit is shifted out to the slave device a bit from the slave device is shifted into the SSPxSR register. After 8 shift operations the contents of the master SSPxSR register and the slave SSPxSR are exchanged.
So, a PIC18 microcontroller is just a toy compared to a PIC32. It's surprising that a PIC32 in fact needs much less programming and it is much easier to program than a PIC32 microcontroller.

a. SPI functions

PIC32F250J128B firmware v2.9.1 and LIB_PCUSBProjects v5.3.NET4(x64).dll provide the following functions:

HARDWARE FUNCTIONS

```
Function SPI1Init() as Integer
Function SPI1ReadReg(addr as Byte) as Integer
Function SPI1WriteReg(addr as Byte, dta as Byte) as Integer
Function SPI1Write(dta as UInt16) as Integer
Function SPI1BurstReadReg(addr as Byte, size as Byte) as Integer
Function SPI1BurstWriteReg(addr as Byte, size as Byte) as Integer
```

SOFTWARE FUNCTIONS ARE INTENDED FOR TESTING HARDWARE CONNECTIONS AND DATA TRANSFER SPEED ONLY:

```
Function SWSPIInit() as Integer
Function SWSPIReadReg(addr as Byte) as Integer
Function SWSPIWriteReg(addr as Byte, dta as Byte) as Integer
```

Three functions that enable initialization of the microcontroller SPI interface, and single read and write access to HopeRF RFM69CW wireless communication module registers, where the address is 7-bit RFM69CW internal register address.

Digital thermometer and thermostat (ADT7410) v9 (x64) example from Downloads section provides more information in how to use HW SPI interface.

EXAMPLE 1: OPEN CONNECTION AND READ DATA

```
Dim SPI1InitFlag=False
Dim A as Integer

If Not SPI1InitFlag Then
   PIC.SPI1Init()
   SPI1InitFlag=True
End If
A = PIC.SPI1ReadReg(1)  ‘ A gets values of register 1
If A < 0 then
       Lb_Report.Items.Add(“Error”)
Else
       Lb_Report.Items.Add(“Register value = “+CStr(A))
End If
```

EXAMPLE 2: OPEN CONNECTION AND WRITE DATA

```
Dim SPI1InitFlag=False
Dim A as Integer

If Not SPI1InitFlag Then
   PIC.SPI1Init()
   SPI1InitFlag=True
End If
If (PIC.SPI1WriteReg(1,4)<0) Then  ‘ A set 4 to register 1
       Lb_Report.Items.Add(“Error”)
Else
       Lb_Report.Items.Add(“Register value set to 4“)
End If
```

EXAMPLE 3: You can use only MemRead and MemWrite intstructions

…to directly communicate with PIC32 registers, but it is very slow… Therefore the following example is only meant for reference on how the communication protocol actually works:

Let’s see a code example that initializes PIC32MS250F128B SPI1 to use advanced buffer mode, resets RFM69CW, reads a few RFM69CW registers and finally returns the data from PIC32MS250F128B SPI1 hardware buffers:

```
Dim addr As UInt32
Dim clear_ip7_spi1 AsUInt32 = &HF000000
Dim spi1con_value As UInt32 = &H1
'addr = Convert.ToByte(TBX_Address.Text, 16)
spi1con_value <<= 16
spi1con_value += &H8527 ' We use 16-bit words to communicate with RFM69CW. Register read:
Upper 8 bits = register address, lower 8 bits = data be written
clear_ip7_spi1 <<= 4
' PORTB configuration
pic.MemWrite(ANSELB, pic.MemRead(ANSELB) And& H814F) ' all SDI bits are digital
pic.MemWrite(ODCB, pic.MemRead(ODCB) And& H814F) ' all SDI bits as not open drain
```

```
pic.MemWrite(LATB, (pic.MemRead(LATB) And& H8310) Or &H10) ' Set bits {4 (RESET), 7 (SS1)
,13 (SDO1), 14 (SCK1)} to 0 and bit 5 (SD1) to 1 --> RESET RFM69CW
pic.MemWrite(TRISB, (pic.MemRead(TRISB) And& H8310) Or &H20) ' Set bits {4 (RESET), 7 (SS1)
,13 (SDO1), 14 (SCK1)} to 0 (OUTPUTS) and bit 5 (SDI1) to 1 (INPUT)
' Lb_Report.Items.Add("PORTB = " + Hex(pic.MemRead(PORTB)))
Delay(10) ' Wait until RFM69CW reset is appled (RB5=1 --> set in LATB)
'PPS configuration
pic.MemWrite(RPBR7R, 3)
pic.MemWrite(RPBR13R, 3)
pic.MemWrite(SS1R, 4)
pic.MemWrite(SDI1R, 1)
pic.MemWrite(PORTB_CLR, &H10) ' RESET = 0
Delay(10) ' Wait until RFM69CW restarts
'SPI initialization
pic.MemWrite(IEC1_CLR, &H70) ' disable SPI1 interrupts
pic.MemWrite(SPI1CON, 0) ' stop SPI operation
pic.MemWrite(PORTB_SET, &H80)
pic.MemRead(SPI1BUF) ' clear recive buffer
pic.MemWrite(IFS1_CLR, &H70) ' clear SPI1 interrupt flags
pic.MemWrite(IPC7_CLR, clear_ip7_spi1)
pic.MemWrite(SPI1BRG, 1) ' CLK freq. = (FPB = CPU freq.) / 4
pic.MemWrite(SPI1STAT, &H40) ' clear overflow
pic.MemWrite(SPI1CON2, &H300) ' set framed mode operation
pic.MemWrite(SPI1CON, spi1con_value)
' Send Read Command for register 1
pic.MemWrite(PORTB_CLR, &H80) ' NSS = 0
addr = 1 ' Read register 1
addr<< = 8 ' add second byte to enble reading
pic.MemWrite(SPI1BUF, addr) ' NSS = 1
pic.MemWrite(PORTB_SET, &H80)
pic.MemWrite(PORTB_CLR, &H80) ' NSS = 0
addr = 3 ' Read register 3
addr<< = 8 ' add second byte to enble reading
pic.MemWrite(SPI1BUF, addr) ' NSS = 1
pic.MemWrite(PORTB_SET, &H80)
pic.MemWrite(PORTB_CLR, &H80) ' NSS = 0
addr = 4 ' Read register 3
addr<< = 8 ' add second byte to enble reading
pic.MemWrite(SPI1BUF, addr) ' NSS = 1
pic.MemWrite(PORTB_SET, &H80)
pic.MemWrite(PORTB_CLR, &H80) ' NSS = 0
addr = 6 ' Read register 3
addr<< = 8 ' add second byte to enble reading
pic.MemWrite(SPI1BUF, addr) ' NSS = 1
pic.MemWrite(PORTB_SET, &H80)
pic.MemWrite(PORTB_CLR, &H80) ' NSS = 0
addr = 3 ' Read register 3
addr<< = 8 ' add second byte to enble reading
pic.MemWrite(SPI1BUF, addr) ' NSS = 1
pic.MemWrite(PORTB_SET, &H80)
' Read all the received data from the buffer..
Lb_Report.Items.Add("SPI1BUF = "+ Hex(pic.MemRead(SPI1BUF)))
Lb_Report.Items.Add("SPI1BUF = " + Hex(pic.MemRead(SPI1BUF)))
Lb_Report.Items.Add("SPI1BUF = "+ Hex(pic.MemRead(SPI1BUF)))
Lb_Report.Items.Add("SPI1BUF = "+ Hex(pic.MemRead(SPI1BUF)))
Lb_Report.Items.Add("SPI1BUF = "+ Hex(pic.MemRead(SPI1BUF)))
```

Full sample code is available as an extension of Digital thermometer and thermostat example. **Get Digital thermometer and thermostat (ADT7410) v9 x64 with RFM69CW SPI connection example.zip** from Downloads section.

Hoperf RFC85 and RFC83C(L) simple 1-bit wireless communications

http://sites.google.com/site/pcusbprojects
simonvav@gmail.com

There are a number of functions for building custom wireless PWM based communication protocols:

```
WirelessNextState(xval as Integer) as Integer
WirelessVerifyConditions(xval as Integer) as Integer
WirelessProcess(x as Integer) as Integer
WirelessGetSignalPhase(x as Integer) as Integer
WirelessSetSignalPhase(x as Integer, ph as Integer) as Integer
WirelessGetCondAbove(x as Integer) as Integer
WirelessSetCondAbove(x as Integer, ph as Integer) as Integer
WirelessGetCondBelow(x as Integer) as Integer
WirelessSetCondBelow(x as Integer, ph as Integer) as Integer
WirelessReadBufferAddr(x as Integer, addr as Integer) As UInt64
WirelessReadBuffer(x as Integer) as UInt64
WirelessResetBuffer() as Integer
WirelessReadCurrentChar() as UInt64
```

The functions enable detection of communication signs and data word exchange. Data words that are transferred through a wireless communication channel may have lengths from 4 bits to up to 64-bits. A communication buffer is used by receiver to collect all transmitted data words.

There are also four preloaded tables that define wireless transmission protocol operation:

```
Dim STATE_tran(9) as Integer
Dim cond_above(21) as Integer
Dim cond_below(21) as Integer
Dim trans_set(20) as Integer
```

There are three states of communication protocol operation: IDLE, DATA_TRANSMISSION and NIBBLE_SEPARATOR_DETECTED. STATE_tran table contains rule indexes for each state transition. Since there are 3 states, there are 9 possible state transitions (3 transitions from each state). The transition table can therefore be linear under condition that the first three state transition entries in the table relate to IDLE state, the next three to DATA_TRANSMISSION state and the last three to the NIBBLE_SEPARATOR_DETECTED state.

trans_set table is intended for the transmitter. It contains an optimal pulse width for each communication sign. The next two tables **cond_above** and **cond_below** are used by a receiver. They define minimum and maximum pulse width for a particular communication signal detection. **WirelessVerifyConditions** function takes current pulse width (x) as a parameter and returns the communication sign. **WirelessNextState** function also takes current pulse width and returns next state of operation based on the detected communication sign. **WirelessProcess** function that calls **WirelessNextState** and **WirelessVerifyConditions** functions enables receiver operations. It takes pulse with as an input parameter, stores data payload communication signs and stores them to the communication buffer. The communication buffer operates as a stack (LIFO = last in first out). There are a number of functions that operate on the communication buffer: **WirelessReadBufferAddr** enables reading of an arbitrary communication sign from a chosen buffer address. **WirelessReadBuffer** returns the last received communication sign from the top of the buffer and removes it from the buffer. **WirelessResetBuffer** resets the buffer pointer to its initial state (0).

A number of **"Set" and "Get" functions** enable a programmer to read current settings and/or set his or hers optimal values.

WirelessReadCurrentChar has primarily diagnostic role. It enables a programmer to determine the current communication sign that is being processed by the receiver.

NOTE: Wireless protocol relies on the pulse width values received from ECCP and IC (input control) modules and the pulse width values transmitted by ECCP and OC (output control) modules. At least one digital transmitter (like Hoperf RFC85) and one digital receiver external module (like Hoperf RFC83CL or RFC83C) is needed to establish a wireless communication channel.

See PIC32MX250F128B DATA TRANSFER PROTOCOL(x64).zip for details on wireless communication implementation.

HopeRF RFM69CW wireless transceiver module

a. How it works?

Wireless communications are crucial for remote control applications. Transceiver modules are capable of both receiving and transmitting data.

A simple 1-bit transceiver module requires a complex communication program library, but communication modules with microcontrollers are much better option. RFM69CW is the latest HopeRF communication module that communicates with a microcontroller through an SPI interface. It has a number of control registers that enable setting different communication hardware configurations.

A crucial part of RFM69CW is also a data packet handler unit. It has a special set of control and stratus registers that define various packet formats for data transfer as well as optional CRC error detection and data encryption with AES algorithm.

RFM69CW packet handler enable fully buffered unidirectional communications. A data packet is first loaded to the module internal buffer and it is then transmitted to another RFM69CW module that operates in reception mode.

b. Packet communications

PIC18F26J50 firmware v2.6.6 and PIC32 firmware v2.9.1 support packet communications with CRC error detection. An inbuilt communication protocol enables copying a byte array with a maximum length of 255 bytes to RFM69CW output buffer. The byte array is then transmitted over to another RFM69CW module and emerges in its buffer. A microcontroller on the receiving side can download and process the packet.

c. RFM69CW functions

PIC18F26J50 firmware v2.6.7 and PIC32 firmware v2.9.1 have the following inbuilt functions that are available to high level applications through LIB_PCUSBProjects v5.3.NET4(x64).dll library:

```
Function RFM69CW_Init_SPI1() as Integer
Function RFM69CW_SendPacket_SPI1(size as Byte) as Integer
Function RFM69CW_SendPacketAndSetReceiveMode_SPI1(size as Byte);
Function RFM69CW_SetReceiveMode_SPI1(size as Byte) as Integer
Function RFM69CW_ReceivePacket_SPI1(size as Byte) as Integer
Function RFM69CW_SetDestAddress(addr as UInt16) as Integer
```

All functions return a negative value in case of an error. There are common error codes for all functions and special error codes for a particular function. Common error codes are the following:

Equal to 0 or above **0** = Function completed successfully. The data represents the function output
-2 = USB communication failed
- 4 = DLL internal error
-99 = unsupported function or receive mode set (RFM69CW_ReceivePacket_SPI1 function, only)

RFM69CW_Init_SPI1 function initializes SPI1 microcontroller interface, then enters default values to external RFM69CW module and finally sets idle operation mode to RFM69CW. The carrier frequency is set to 868 MHz.

RFM69CW_SendPacket_SPI1 function uploads data from PC output buffer packet (data array) from PIC18F26J50 or PIC32 to RFM69CW via SP1 interface and transmits it to another RFM69CW module that operates either in data receive mode. The length of the packet is determined by size parameter. PC output buffer is accessible through **WriteBuffer** and **WriteBufferWORD** commands. See Transfering data arrays between DLL and VB.NET applications in Chapter 3: PIC32MX2X0 Programming guide for more information.

RFM69CW_SendPacketAndSetReceiveMode_SPI1 function works almost the same as RFM69CW_SendPacket_SPI1 function, except that is sets RFM69CW to receive mode after a successful completion. This enables faster transition from transmitter mode to receiver mode and consequentially faster data exchange between base station and responder.

RFM69CW_SetReceiveMode_SPI1 function sets RFM69CW to receive mode (RFM69CW **register 1** is set to **0xC**).

RFM69CW_ReceivePacket_SPI1 function first verifies RFM69CW operation mode. If it is not (**0x10**) receive mode, it sets receive mode and exits with error code **-99**. If the receive mode is set, it checks status of the RFM69CW internal buffer. It returns error code **-100**, if the buffer is empty, or error code **-101** in case of CRC error during reception. If RFM69CW internal buffer is not empty, data is download to PC buffer that is accessible through **ReadBuffer** and **ReadBufferWORD** commands. See Transfering data arrays between DLL and VB.NET applications in Chapter 3: PIC32MX2X0 Programming guide for more information.

Size parameter determines data packet length and must have the same value as the size parameter used with RFM69CW_SendPacket_SPI1 function on the transmitting microcontroller.

RFM69CW_SetDestAddress sets destination address of a remote PIC32 microcontroller with firmware v2.9.1 or later. The destination address is used by the functions above to route the data packet to a particular PIC with RFM69CW transceiver that operates on 868 MHz carrier frequency. The preprogrammed destination address is 0x00AB. See the table in section **e**.

RFM69CW_DEMO(x64) and **RFM69CW Transceiver Chat (x64)** programming examples is available from PC Projects Downloads section (https://sites.google.com/site/pcusbprojects/6-downloads). Here are two short examples:

EXAMPLE 1: INITIALIZE RFM69CW MODULE, IF NECESSARY AND SEND STRING

```
Dim s As String
Dim n As Integer
s = Tb_Text.Text
While s.Length < 33
    s = s + " "
End While
If Not RFM69CW_initialized Then
    PIC.RFM69CW_Init_SPI1()
    Lb_Report.Items.Add("RFM69CW Intialization finished...")
End If
For n = 0 To 31     ' Fill buffer
    PIC.WriteBuffer(n, Asc(Mid(s, n + 1, 1)))
Next
n = PIC.RFM69CW_SendPacket_SPI1(32)
Lb_Report.Items.Add("String: " + s + " sent!")
Lb_Report.Items.Add("error code = " + CStr(n))
```

EXAMPLE 2: INITIALIZE RFM69CW MODULE, IF NECESSARY AND RECEIVE STRING

```
Dim n As Integer
Dim s As String
If Not RFM69CW_initialized Then
    PIC.RFM69CW_Init_SPI1()
    Lb_Report.Items.Add("RFM69CW Intialization finished...")
End If
n = PIC.RFM69CW_ReceivePacket_SPI1(32)
If n = 0 Then
    s = ""
    For n = 0 To 31
        s = s + Chr(PIC.ReadBuffer(n))
    Next
    Lb_Report.Items.Add("> " + s)
Else
    Lb_Report.Items.Add("Receive packet error code = " + CStr(n))
End If
End Sub
```

d. Inbuilt wireless software responder module

PIC32 firmware v2.9.1 (or later) and PIC18F26J50 firmware v2.6.7 (or later) have inbuilt wireless SW responder modules. Two functions are currently supported. First, transmits PIC32 firmware version and the second returns temperature readings from one more ADT7410 16-bit temperature sensors. You can more about ADT7410 functions on page 31.

The responder is initially disabled and must be enabled through USB interface with **RFM69CW_ResponderCfg** function:

```
Function RFM69CW_ResponderCfg(enable As Byte, init_enable As Byte, op_enable As Byte) as
Integer
```

There are three parameters: **enable** temporary enables or disables responder functionality (0 = disable, 1 = enable). **init_enable** enables RFM69CW initialization, if it is not initialized, **op_enable** enables the responder operation upon initialization.

Example 1: The following command initializes and starts responder:

```
RFM69CW_ResponderCfg(1,1,1)
```

Example 2: The following command stops and disables responder:

RFM69CW_ResponderCfg(0,0,0)

During normal operation wireless SW responder module sets RFM69CW to operate in receive mode. Packet length is set to 32 bytes and CRC is enabled. The responder automatically checks RFM69CW packet handler control register (address 0x28). If a packet is received, the first byte is presumed to be a function number.

Two functions are currently supported:

0 = Get firmware number

1 = Get buffered temperatures from attached ADT7410 sensors.

Use **RFM69CW_SendPacket_SPI1(32)** function on the remote microcontroller to call either function. The first function returns four bytes that contain PIC32 firmware version numbers and oscillator frequency number (example: 2.9.1.5). The oscillator frequency numbers have the following meaning: 1 = 4 MHz, 2 = 8 MHz, 3 = 12 MHz, 4 = 16 MHz, 5 = 20 MHz.

The next function (1) returns 2 bytes per each ADT7410 temperature sensor. The temperature may be read with **RFM69CW_ReceivePacket_SPI1(32)** function in the remote microcontroller.

Here is a programming example on how to call a responder function from a remote microcontroller attached to a PC and receive results:

```
Dim RFM69CW_initialized As Boolean = False

Private Sub GetRemoteFWVersion_Click(sender As System.Object, e As System.EventArgs)
Handles Button1.Click
   Dim s As String
   Dim n As Integer
   If Not RFM69CW_initialized Then
      PIC.RFM69CW_Init_SPI1()
      Lb_Report.Items.Add("RFM69CW Intialization finished...")
   End If
   PIC.WriteBuffer(0, 0)  ' Write remote microcontroller function number to buffer
location 0
   For n = 1 To 31    ' Fill buffer
       PIC.WriteBuffer(n, 0)
   Next
   n = PIC.RFM69CW_SendPacket_SPI1(32)
   Lb_Report.Items.Add("Command sent! Error code = " + CStr(n))
   If n = 0 Then
       Do
           Delay(1) ' Wait for 1 ms
           n=PIC.RFM69CW_ReceivePacket_SPI1(32)
       Loop While n<>0
       If n = 0 Then
          Lb_Report.Items.Add("Receive packet error code = " + CStr(n))
         Lb_Report.Items.Add("FW Version = " + Hex(PIC.ReadBuffer(0)) + "." +
         Hex(PIC.ReadBuffer(1)) + "." + Hex(PIC.ReadBuffer(2)) + "." +
         Hex(PIC.ReadBuffer(3)))
       End If
   End If
End Sub
```

e. Setting Inbuilt wireless software responder module EEPROM defaults

It is obvious that an automatic software responder can only serve its purpose, if it starts automatically at microcontroller boot time. However, this functionality is not enabled by default for two reasons: All PC USB Projects firmware is meant to be universal, so a programmer must decide what functionality he or she actually needs. ADT7410 sensors are not inbuilt into PIC microcontrollers, so they are only needed, it the microcontroller is to be used for temperature measurement. This is currently the only function that is supported by the SW responder module. There will be more in the future.

The EEPROM defaults are in four 16-bit words on the following EEPROM locations:

Address	Default Value	Description
0xBD01E700	0x0000	0 = RFM69CW transceiver initialization disabled at startup, 1 = RFM69CW transceiver initialization enabled at startup
0xBD01E702	0x0000	0 = RFM69CW transceiver initialization disabled at startup, 1 = RFM69CW transceiver initialization enabled at startup
0xBD01E704	0x2000	responder delay to transmit data in us (LWORD) (PIC32 firmware only)
0xBD01E706	0x0000	responder delay to transmit data in us (HWORD) ((PIC32 firmware only)
0xBD01E708	0x00AB	16-bit responder address
0xBD01E70A	0xFEFE	responder response distribution address

To enable automatic start of the inbuilt SW wireless responder, set values on locations 0xBD01E700 and 0xBD01E702 to 1. Do not forget to also enable temperature control at startup. See page 32 for details.

The above memory locations may be set from .NET application with standard EEPROM programing functions: **`PIC32_ErasePageEEPROM`** and **`PIC32_WriteEEPROM`**. Do not forget to store the EEPROM page content prior to erasing it, if you need to preserve any data. The details on using the functions are in page 32.

4. MICROCONTROLLER PROGRAMMERS

INTRODUCTION: A micro controller programmer is an essential tool to get a PC USB Project from the drawing board to reality. This section contains various blueprints of PIC microcontroller programmers. Choose one that best suits your needs. Those with no access to a microcontroller programmer (ex. if your friend has one) to transfer the firmware to PIC18F2xJ50 may opt for Simple USB microcontroller programmer that is based on MCP2200 manufacturer preprogrammed microcontroller. But only PIC18F2xJ50 based programmers can do the programming in a short time. However, if you want to instantly build a super speed microcontroller programmer, you may also use Velleman K8055N board together with a simple adapter (see subsection 4.b on details). The programming of your first PIC18J2xJ50 microcontroller will have to be done at slow speed, but as soon as the chip is successfully programmed with PIC18F2xJ50 firmware v2.0 or above you would be to replace the original Velleman chip by it, or build a custom programmer with super speed programming capability.

If you are eager to program one of the newest 32-bit microcontrollers, there is also a homemade programmer, but there is

@.WINDOWS APPLICATIONS for Microcontroller Programmers

PIC programmer v1.1 MS Windows application currently supports Microchip PIC18FxxJxx microcontroller family and HEX file data format and works only with simple USB microcontroller programmer. It works on all Microsoft Windows versions with .NET 2.0 framework or later installed. You can obtain .NET framework 4.0 for your windows version on the Microsoft developer's website (http://developer.microsoft.com).

PIC Programmer v2.x MS Windows applications currently support Microchip PIC18FxxJxx microcontroller family and HEX file data format and work only with fast and superspeed USB microcontroller programmers. They work on all Microsoft Windows versions with .NET 2.0 framework or later installed. You can obtain .NET framework 4.0 for your windows version on the Microsoft developer's website (http://developer.microsoft.com).

PIC Programmer v2.3 x64 is a 64-bit MS Windows application that currently supports Microchip PIC18FxxJxx microcontroller family and HEX file data format and works only with superspeed USB microcontroller programmers. A 64-bit Microsoft Windows version with .NET 4.0 framework or later is required. You can obtain .NET framework 4.0 for your windows version on the Microsoft developer's website (http://developer.microsoft.com).

PIC Programmer v2.4 x64 is a 64-bit MS Windows application that currently supports Microchip PIC18FxxJxx microcontroller family and HEX file data format and works only with superspeed USB microcontroller programmers. A 64-bit Microsoft Windows version with .NET 4.0 framework or later is required. You can obtain .NET framework 4.0 for your windows version on the Microsoft developer's website (http://developer.microsoft.com). Automatic detection of super speed programmer and faster programming with Velleman K8055 and K8055N boards and chip reset function were added. There is no initial delay before super speed operations start. There is also a 32-bit version: PIC Programmer v2.4 for 32-bit Windows versions.

PIC Programmer v2.5 x64 is a 64-bit MS Windows application with source code based on SVLIB_PIC18F24J50 v2.1.NET4.DLL and PIC18F2xJ50 firmware 2.0 or later!

PIC Programmer v2.6 (x64) is a 32-bit and 64-bit MS Windows application with source code based on SVLIB_PIC18F24J50 v2.6.st.NET4.DLL and PIC18F2xJ50 firmware 2.2 or later! The application features enhanced USB connection detection.

PIC Programmer v2.9.9 (x64) is a 32-bit and 64-bit MS Windows application with source code based on SVLIB_PIC18F24J50 v2.7c.st.NET4.DLL and PIC18F2xJ50 firmware 2.6.1 or later! The application features

enhanced USB connection detection and user function uploading via USB 2.0 (no programmer needed) and execution support. It also supports h**igh programming voltage adapter** that extends the programming options to the whole PIC18 microcontroller family.

WARNING! THE APLICATIONS ARE PROVIDED AS THEY ARE! YOU USE THEM AT YOUR OWN RISK!

INSTALLATION INSTRUCTIONS FOR PIC programmer v1.1

1. Install Microchip DLL or driver for MCP2200 from Microchip website: http://www.microchip.com/ --> "MCP2200" into "Search Microchip" field (upper search field) or click on the following link:

http://ww1.microchip.com/downloads/en/DeviceDoc/MCP2200%20DLL.zip for DLL

or

http://ww1.microchip.com/downloads/en/DeviceDoc/MCP2200%20Windows%20Driver.zip for Windows driver

2. Install MCP2200 driver if you have an older Windows version and make sure to copy SimpleIO-M.dll to the same directory as SV PIC Programmer application

(Pic Programmer v1.1.exe) from DOWNLOADS.

The install directory is not important! Create directory wherever you like and optionally make link to the desktop.

INSTALLATION INSTRUCTIONS FOR PIC programmer v2.x

Go to Downloads section and download file Pic Programmer v2.x.zip. The ZIP package contains 3 files: EXE, DLL and HEX. Program HEX file to PIC18F24J50 or PIC18F26J50 (ex. with simple USB microcontroller programmer) and then put EXE and DLL files in the same folder. Remember that the program will only work, if at least .NET framework iz installed. You don't need to download any files from Microchip! However for advanced users it might be usefull to download PIC18F24J50 or PIC18F26J50 user guide.

A. Simple USB microcontroller programmer

The simple USB microcontroller programmer is designed for beginners. It is built around Microchip MCP2200 microcontroller that is preprogrammed by the manufacturer. The only setback is the MCP2200 chip that is only produced in two SMD (surface mounted device) versions. Though it is possible to solder it to standard prototyping PCB (printed circuit board) and connect it to other necessary electronics, this certainly requires a steady hand and good eyesight. The pin spacing of the bigger version of MCP2200 is twice as small as normal spacing for standard chips. It is possible to use an alpha knife to split the contact circles in half and then solder each pin to a half-contact. Wires must be carefully on each half-contact soldered to make connections to other electronics on the prototyping board.

There are also other ways to build the programmer circuit. You can buy a rather expensive interface PCB board for SMD. Should you decide to design your own PCB from scratch, you need a high quality PCB mask to avoid etching-off tin SMD contacts.

B. Fast USB microcontroller programmer schematics

http://sites.google.com/site/pcusbprojects
simonvav@gmail.com

Fast USB programmer is based on a preprogrammed PIC18F24J50. Time critical functions were moved to firmware therefore the programmer is more than 20 times faster. Programming PIC18F24K50 with 16kB of program code takes only about 20 seconds.

It is important to use 4 MHz crystal oscilator. To obtain HEX, DLL and EXE files for PIC18F24J50 or PIC18F26J50 go to DOWNLOADS section and download package Pic Programmer v2.0.zip. This file also works on K8055 board as a replacement SW which vastly extends the board functionality.

C. Simple and fast microcontroller programmers based on Velleman K8055-1 and K8055N-2 boards

If you already own a Vellaman K8055 or K8055N board, why not turn it into microcontroller programmer? You can use either the original Velleman preprogrammed PIC16C745 (K8055 board) or PIC18F24J50 or PIC18F26J50 (K8055N board), or program PIC18F24J50 or PIC18F26J50 (K8055N board or K8055 board with PIC18F24J50 or PIC18F26J50 only) yourself to get fast programming capability.

Please go to section 4.b. to see how.

D. High programming voltage microcontroller programmer adapter

If you have already built a low voltage programmer, you just have to add a high programming voltage microcontroller programmer adapter and a 12 V power supply to enable high voltage programming. You also need a new vesrion of programming application. Go to Downloads section to get PIC programmer v2.0d application (version 3.0 is comming soon).

Resistors R1 through R4 values have to be calculated to meet the required voltage ranges. You may also decide to use trimmers (variable resistors). Here are the instructions for resistors values calculations:

$V_{high} = 1.25\ V\ (\ 1 + R_2 / R_1\ ) + I_{Adj}\ R_2$

$V_{ddprg} = 1.25\ V\ (\ 1 + R_4 / R_3\ ) + I_{Adj}\ R_4$

I_{Adj} is regulated by LM317 to be below 0,1 mA. To get the V_{high} and V_{ddprg}voltages into desired ranges: **V_{high} = 8 V .. 10 V** and **V_{ddprg} = 3.3 V .. 5 V** you have first to decide on the values of **R_1 and R_3.**You may go for **680 ohms** that you have already used for 3.3 V regulators. Now, we can calculate R_2 and R_4. It is important to meet as many PIC18 programming datasheets requirements as possible. Therefore we can go for: V_{high} = 8.4 V and V_{ddprg} = 4 V.

$R_2 = (V_{high} – 1.25\ V) / (1.25\ V / R_1 + I_{Adj})$

R_2 = (**8.4 V** – 1.25 V) / (1.25 / 680 A + 0.1 mA) = 7.15 V / 1.938 mA = **3,69 k ohms**

$R_4 = (V_{high} – 1.25\ V) / (1.25\ V / R_3 + I_{Adj})$

R_4 = (**4 V** – 1.25 V) / (1.25 / 680 A + 0.1 mA) =3.75 V / 1.938 mA = **1,42 k ohms**

If you decide to use a trimmer in place of R_2, it should have **5 k ohms** (or 10 k ohms) of maximum resistance. If you go for 10 k ohms trimmer, you will have to set it to less than ½ of full range.

If you decide to use a trimmer in place of R_4, it should have **2 k ohms** of maximum resistance (or 1 k ohms, if you add a 1 k ohms resistor in series).

Here is the schematics:

E. Super speed reconfigurable USB microcontroller programmer

Super speed and reconfigurability are both achieved by advanced microcontroller firmware and enhanced DLL libraries and a new version of programming application (v2.1). Super speed is **about 7 times faster** than the normal fast mode.

To achieve super speed programing you need PIC18F26J50 microcontroller with v2.0 firmware, a new DLL library (version 2.1) and a new Microsoft Windows application (version 2.1). You can download the following files from Download section:

- **PIC18F26J50 firmware v2.6.1.hex or PIC18F24J50 firmware v2.5.hex**
- **SVLIB_PIC18F24J50 v2.7c.st.NET4(x64).dll**
- **Pic Programmer v2.9.9.exe**

You have to copy the SVLIB_PIC18F24J50 v2.1.dll to the same folder as Pic Programmer v2.1.

Programming of 64 kB EEPROM now takes only 11 seconds, while verification completes in 8 seconds, provided that there are no errors discovered.

The idea of easy reconfigurablity is actually not new. Any fast microcontroller programmer on this website can program a new microcontroller of the same type. Therefore, reconfiguration is just a matter of replacing the existing microcontroller chip with a newly programmed one.

It would of course be easier, if everything could be done in software, but it is also true that one does not change programmer software every day.

WINDOWS APPLICATION for Microcontroller Programmer

PIC programmer windows application currently supports Microchip PIC18FxxJxx microcontroller family and HEX file data format. It works on all Microsoft Windows versions with .NET 4.0 framework installed. You can obtain .NET framework 4.0 for your windows version on the Microsoft developer's website (http://developer.microsoft.com).

WARNING! APLICATION IS PROVIDED AS IT IS! YOU USE THE APPLICATION AT YOUR OWN RISK!

F. Super speed programmer software development kit (SDK)

There are many different Microchip microcontrollers. Now, the first 64-bit microcontroller programming application ***PIC Programmer v2.9.9 x64 with source code*** is available from Downloads section. Super speed microcontroller programmer software development kit (SDK) only works with PIC18F2xJ50 firmware v2.0 or later. The firmware is now available for PIC18F24J50 and PIC18F26J50 microcontrollers.

SDK is based on previously undocumented ***SVLIB_PIC18F24J50 v2.7c.st.NET4(x64).dll*** functions. Please see subsection "5.A How to start programming" for the description of the basic DLL functions and programming techniques. Here are descriptions of EEPROM and flash RAM programming functions:

```
int SVPICAPI::EnterICSPEEPROM(DWORD ICSP_entry_code);
```

This function enables ICSP mode entry for Microchip PIC18FxxJxx family microcontrollers. These microcontrollers do not use the high voltage programming instead they enter program/verify mode while operating on normal power supply voltage. They use a special 32-bit code to prevent accidental entry into ICSP mode. PIC18FxxJ50 family ICSP entry code is: 4D434850 (hex).

```
int SVPICAPI::CommandEEPROM(byte cmd, WORD data, byte delay);
```

This function contains all the necessary logic to send a command to a Microchip PIC microcontroller. Please, see Microchip programming documentation (www.microchip.com) for further information on programming the Microchip microcontrollers.

```
int SVPICAPI::CommandResultEEPROM(byte cmd, byte delay);
```

This function contains a standard procedure enables reading data from a microcontroller flash RAM location. It is used for the verification purpose after the programming or to read a microcontroller configuration word that identifies the microcontroller.

```
int SVPICAPI::WriteEEPROM(byte cmd, byte offset, byte cnt, byte delay);
```

Data is written to the microcontroller flash RAM in blocks. The length of the blocks depends on the length of a microcontroller write buffer (ex. PIC18F2xJ50 microcontrollers have 64 bytes of buffer, while PIC18F14K50 microcontroller has only 16 bytes). There is also a post write delay parameter that is used to initiate the write buffer to flash RAM programming sequence. The PIC18Fxxxxx write buffer is accessed through the USB control and status report operations. Each operation had a limited data space and two or more USB operations are needed to transfer 64 bytes to the microcontroller. Therefore *offset* and *cnt* parameters are used to send a specified number of byte from the PC memory buffer to the PIC write buffer. The o*ffset* parameter enables correct positioning in the PC memory buffer.

```
int SVPICAPI::ReadEEPROM(byte cmd, byte cnt);
```

This contains all the necessary logic to read one byte from a previously defined address in PIC flash RAM. The address is set using CommandEEPROM commands. See the sample PIC Programmer application in Downloads section for the details.

```
int SVPICAPI::ReadBuffer(byte adr);
```

This function is used to read data from the PC buffer that is located in the DLL memory space to the user application buffer. See the sample PIC Programmer application in Downloads section for the details.

```
int SVPICAPI::WriteBuffer(byte adr, byte data);
```

This function is used to write data to the PC buffer that is located in the DLL memory space to the user application buffer. See the sample PIC Programmer application in Downloads section for the details.

G. Super speed microcontroller programmers can now be used with PIC18F24J50, too!

PIC18F24J50 v2.0 firmware is now available! You can now use super speed microcontroller programming with PIC18F24J50, too! The library supports all the standard functionalities that were previously reserved for PIC18F26J50. It was quite tricky to build, but now it works just fine! There are 6 subversions of the new firmware:

PIC18F24J50 firmware v2.0 (4 MHz) for general use.hex
PIC18F24J50 firmware v2.0 (12 MHz) for general use.hex
PIC18F24J50 firmware v2.0 (16 MHz) for general use.hex
PIC18F24J50 firmware v2.0 (4 MHz) for K8055 adapter.hex
PIC18F24J50 firmware v2.0 (12 MHz) for K8055 adapter.hex
PIC18F24J50 firmware v2.0 (16 MHz) for K8055 adapter.hex

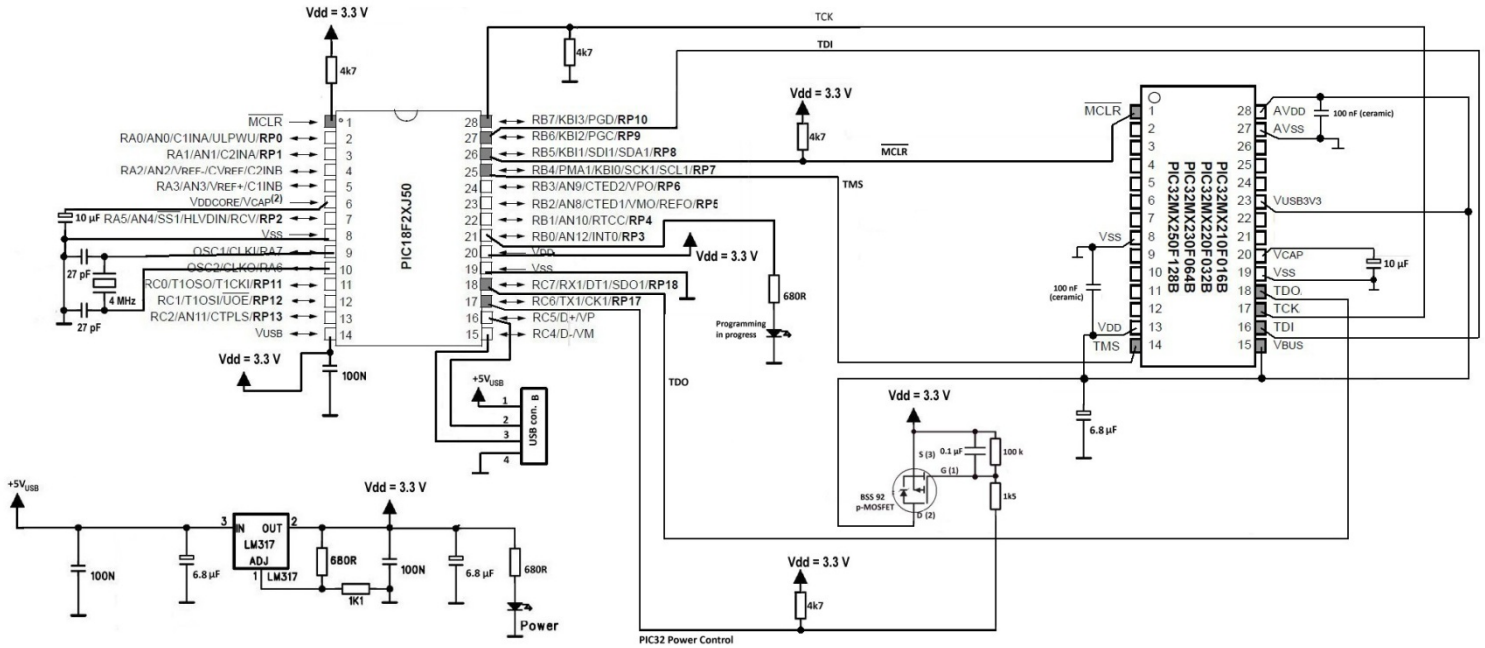
Use the appropriate subversion according to the crystal resonator and hardware you use. The adapter versions drive PWM outputs in inverse mode. If used on K8055N board the D/A outputs would function in inverse mode (0 = maximum voltage, 1023 = minimum voltage).

H. JTAG PROGRAMMER FOR PIC32 FAMILY

Programmers for all microcontrollers are the same, except the JTAG pin (TCK, TDI, TDO and TMS) connections might be different.
Programming guide describes the details on how to write your own JTAG programming. To use JTAG Protocol you need:

- **PIC18F24J50 firmware v2.5 - all subversions.zip** or **PIC18F26J50 firmware v2.6.1 - all subversions.zip** (choose the appropriate HEX file)
- **SVLIB_PIC18F24J50 v3.7.NET4.DLL** or **SVLIB_PIC18F24J50 v3.7.NET4x64.DLL**

You can get both from Downloads section. Here is the JTAG programmer schematic:



See subsection 4.j for more details on writing your own program.

http://sites.google.com/site/pcusbprojects
simonvav@gmail.com

5. STARTER KIT GUIDE

INTRODUCTION TO STARTER KITS

Many microcontroller manufacturers make their own starter kits. They are usually difficult to build, because they are mostly intended as technology demonstrators. SV PIC18F2xJ50 starter kit is of a different kind. It is intended to be easy to build as well as allow a practical use.

There is also a good programming support which includes:

- **Programming guide (subsection 5.a)**

- **PIC18F26J50 firmware v2.6.1 or PIC18F24J50 firmware v2.5** (6. Downloads section)

- various **32-bit and 64-bit DLLs** for single threaded (SVLIB_PIC18F24J50 v2.7.st.NET4.dll and SVLIB_PIC18F24J50 v2.7.st.NET4x64.dll) and multithreaded (SVLIB_PIC18F24J50 v3.7.NET4.dll and SVLIB_PIC18F24J50 v3.7.NET4x64.dll or **LIB_PCUSBProjects v4.1.NET4x64.dll**) applications

- and **sample Visual Studio 2010 VB.NET applications** with source code (see Downloads section and Custom projects section)

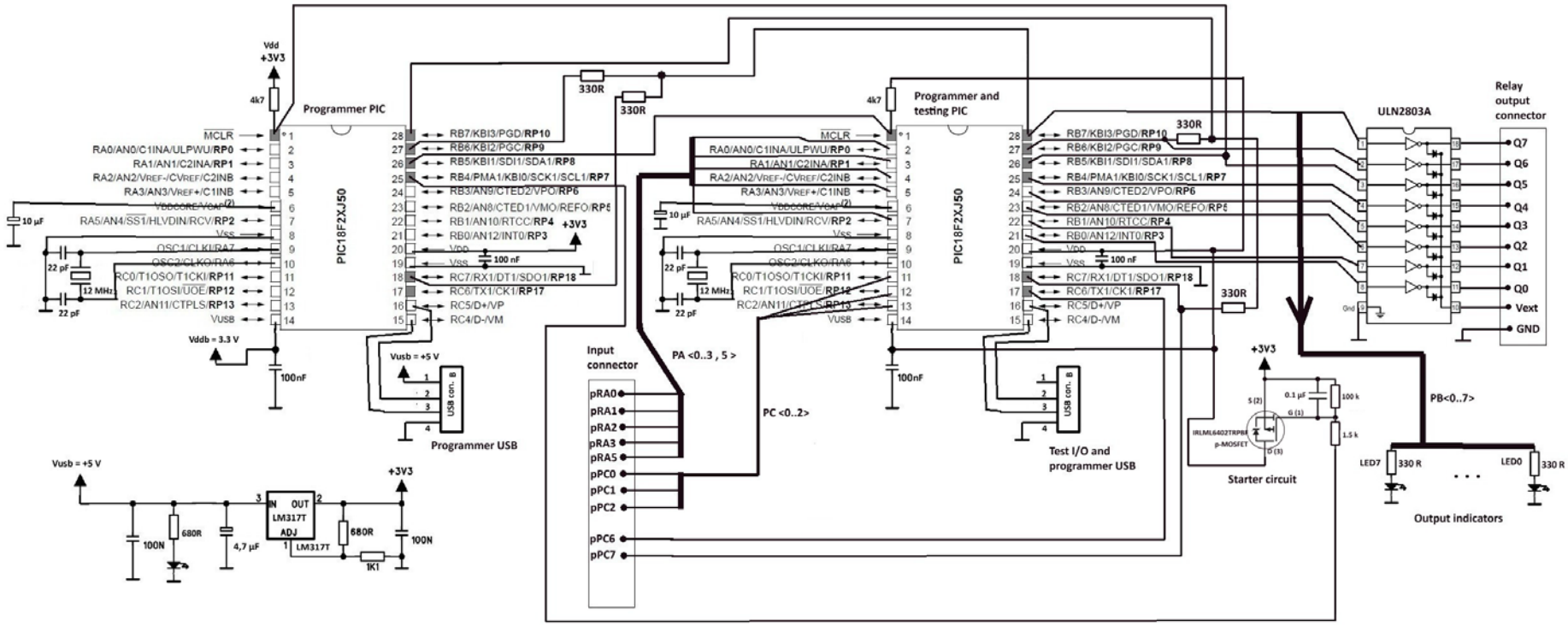
A. SV PIC18F2xJ50 STARTER KIT SCHEMATIC

PIC18F2xJ50 starter kit includes programming and testing functionality. It can also be used to build circuits that allow firmware upgrade without an external microcontroller programmer. Both PIC18F2xJ50 microcontroller firm wares can be altered in-place. This is because each microcontroller has the capability to set MCLR signal low and start ICSP sequence on the other microcontroller. One of the microcontrollers is connected to the ULN2803A relay driver and can drive up to 8 relays without the need for any other external circuitry (see section 2 for more details). The inputs of port A and port C are intended for digital and/or analog input and they are directly connected to the input connector. Please, read subsection 2.b, if you intend to connect power electronics where input protection might be needed. To protect a digital input against negative impulses and inducted voltage it might not be a bad idea to use an additional ULN2803A IC to protect digital inputs as well as operation amplifiers to protect analog inputs and to adapt the voltage ranges, especially if you would like to use A/D conversion in AC signals.

Both PIC18F2xJ50 microcontrollers have equal programming capabilities, but one lacks ULN2803A relay driver. If you need more outputs you can also connect a ULN2803A IC both microcontrollers as well as additional ULN2803As and operational amplifiers to enable digital and analog inputs protection.

The starter kit incorporates a starter circuit that prevent excessive electric current inrush. When the starter kit board is plugged to USB powering two microcontrollers would temporarily overload USB power supply, therefore a p-MOSFET transistor is used as a switch which is enabled by the PIC programmer microcontroller.

Schematic of the starter kit:

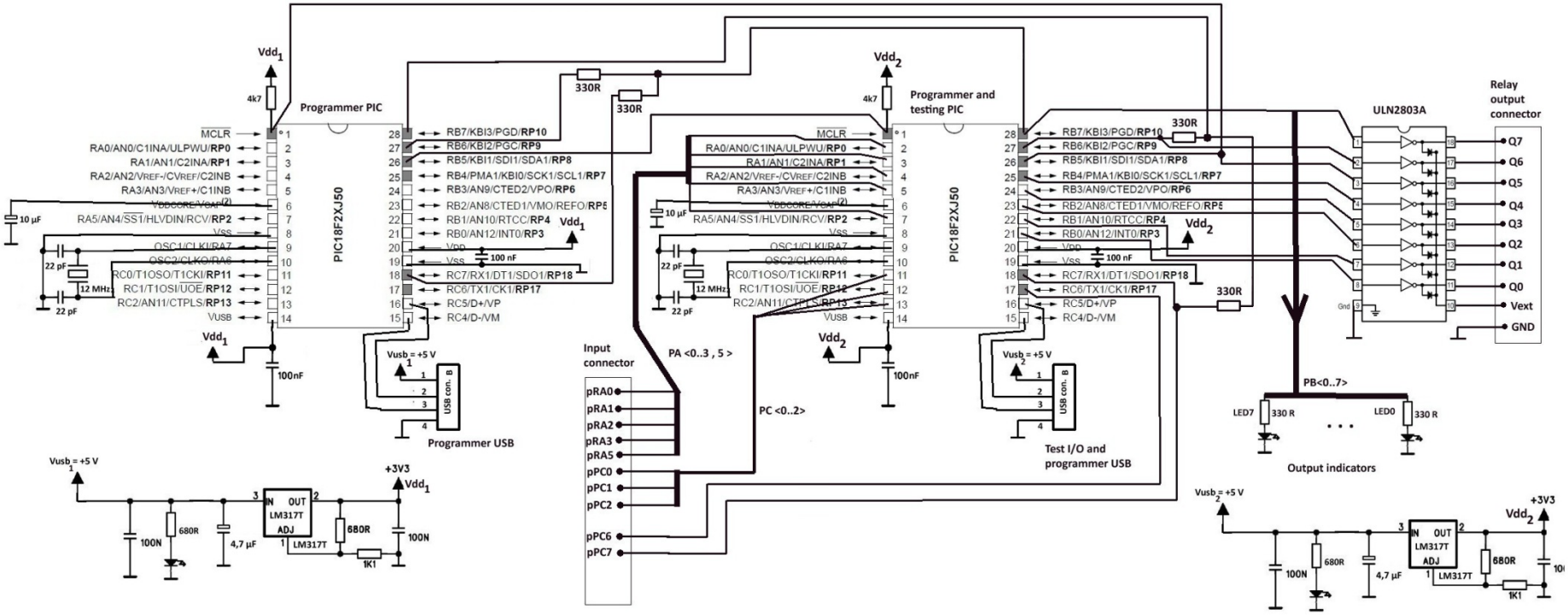


http://sites.google.com/site/pcusbprojects
simonvav@gmail.com

B. SV PIC18F2xJ50 STARTER KIT SCHEMATIC – alternative 1

The alternative starter kit schematic is based on two separate power supplies. Each microcontroller connects to its own voltage regulator which draws power from a separate USB port. This circuit is more advanced, because it allows completely independent microcontroller operation. Microcontrollers may even be connected to different PCs.

Here is the schematic of the starter kit – alternative 1:

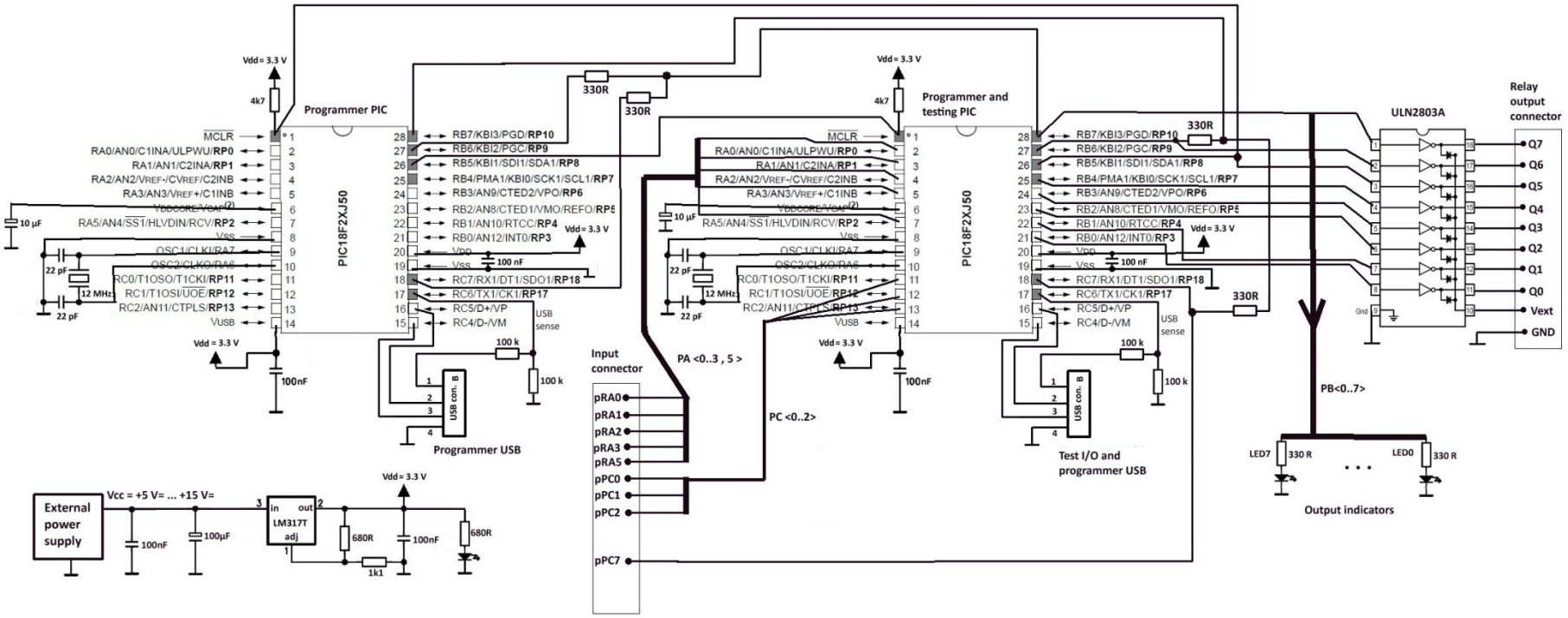


http://sites.google.com/site/pcusbprojects
simonvav@gmail.com

C. SV PIC18F2xJ50 STARTER KIT SCHEMATIC – alternative 2

The second alternative starter kit schematic is based on an external power supply. This enables the PIC microcontroller board to operate even when the computer is completely turned off, or even when USB cable is removed. This starter kit is very useful for applications where PC is only used to set configuration parameters or gather collected data and then the microcontroller works on its own.

Here is the schematic of the starter kit – alternative 2:

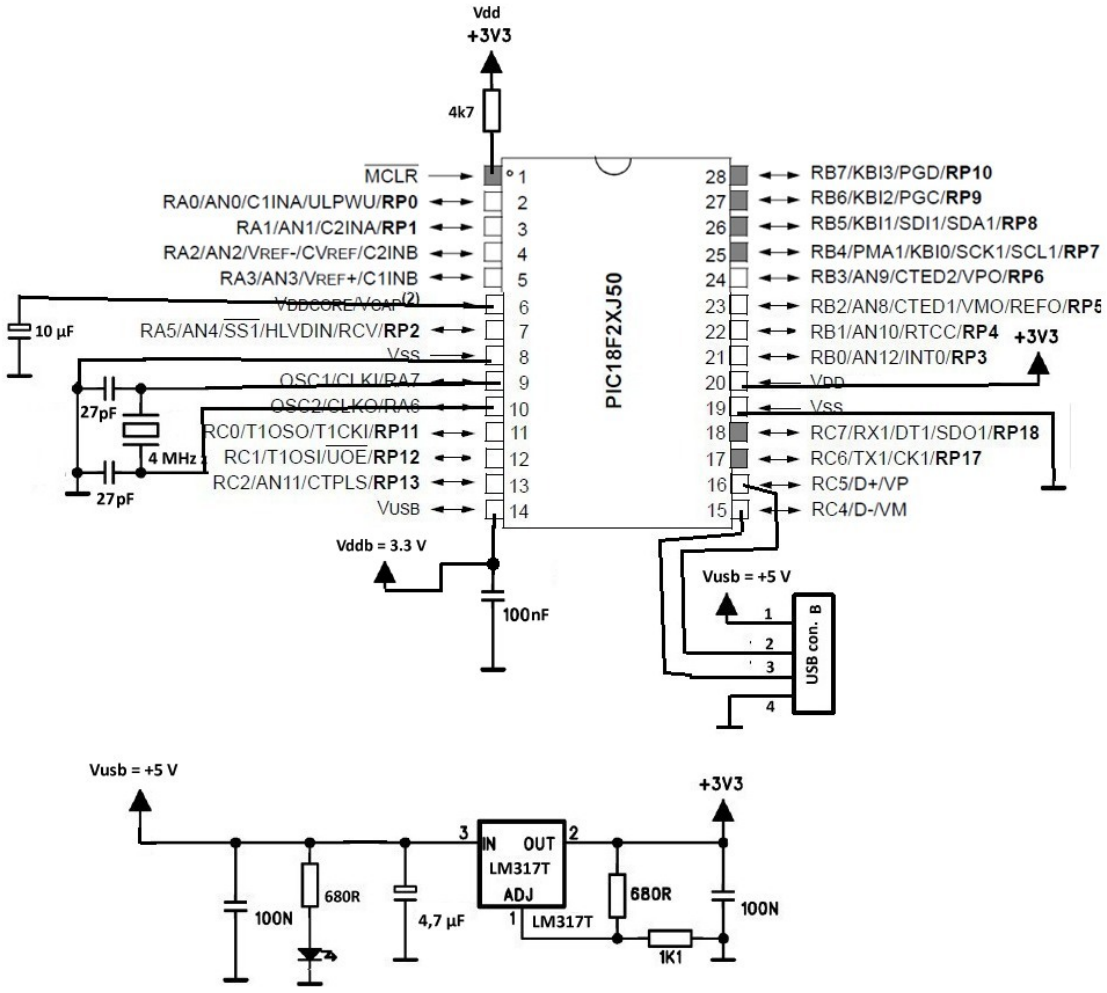


http://sites.google.com/site/pcusbprojects
simonvav@gmail.com

D. Basic circuit for PIC18F24J50 or PIC18F26J50 to work over USB

The basic circuit below will work with any PIC18F2xJ50 microcontroller. You just have to upload the appropriate firmware. Go to Downloads section to find firmware hex files for the microcontroller and the resonator you would like to use. **PIC18F26J50 firmware v2.6.1 - all subversions.zip** contains the following general use subversions: 4 MHz, 8, MHz, 12 MHz, 16 MHz and 20 MHz, and the following subversions for K8055 adapter: 4 MHz, 8, MHz, 12 MHz, 16 MHz and 20 MHz. There is also a variety of subversions of **PIC18F24J50 firmware v2.5** hex file for general use (4 MHz, 8, MHz, 12 MHz and 16 MHz) and for K8055 adapter (4 MHz, 12 MHz and 16 MHz).

If you would like to use the basic circuit for PIC18F2xJ50 microcontroller programming, just go to subsection 4.b and build the adapter initially intended only for K8055 and K8055N boards. The adapter also works perfectly, if you stick it between the basic circuit DIP socket and the PIC18F2xJ50 microcontroller that you are going to use as a programmer for another PIC2xJ50 microcontroller. You may also use the originally programmed Velleman PIC18F24J50 microcontroller with the basic circuit, if you build it with 4 MHz crystal resonator and appropriate capacitors. See subsection 5.c. to get details on choosing the right capacitor values.



Yes, the basic circuit may also serve as a super speed microcontroller programmer (you must manually select option **02 - Super speed K8055 or K8055N programmer**, or set VID = 10CF (hex). You just need to use a small piece of PCB to solder a DIP socket and a few resistors and two transistors and two capacitors. Don't mind about the circuit in the plastic box. It is K8055, but you don't need it! Use PIC Programmer v2.6 and above for PIC18F2xJ50 microcontroller with v2.2 firmware or later and previous version of the application for PIC18F2xJ50 microcontrollers with lower versions of the firmware.

http://sites.google.com/site/pcusbprojects
simonvav@gmail.com

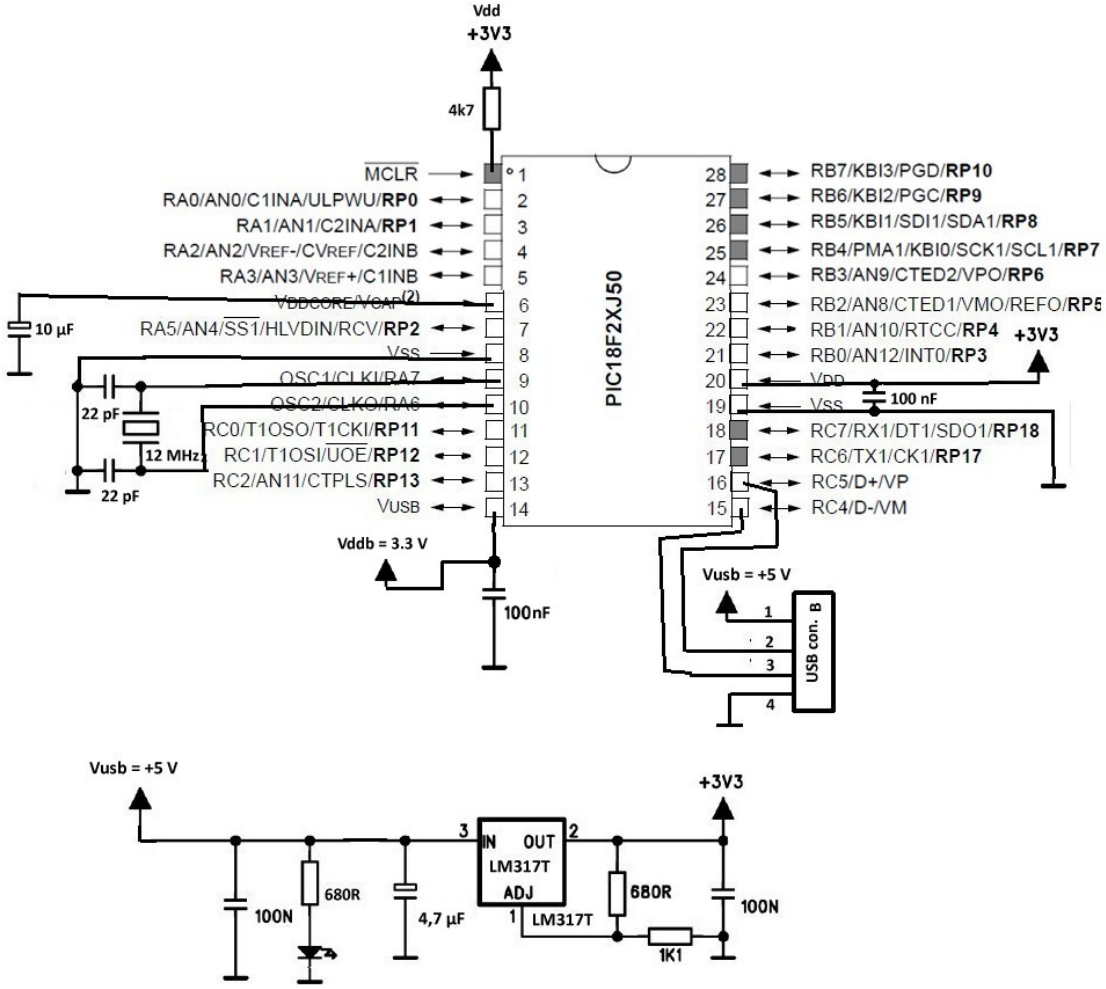
E. Need more speed? Here is a basic circuit for PIC18F24J50 and PIC18F26J50 for high speed operation

A PIC18F2xJ50 microcontroller can run at variety of clock speeds. Though it runs on 48 MHz internally, the selection of a different external clock source frequency has a great impact on some of its functional units, such as timer/counter.

However, communication via USB requires precise timing and use of crystal resonator. The crystal resonator must provide frequency that is a multiple of 4 MHz. There are also some other requirements, so clock frequencies that allow proper USB operation are: 4, 8, 12, 16, 20, 24, 40 and 48 MHz. The external part of PIC18F2xJ50 oscillator circuit consists of two ceramic capacitors and a resonator. The Microchip recommended values for the capacitors are the following:

Ceramic capacitors value	Resonator frequency
27 pF	4 MHz
22 pF	8 MHz
22 pF	12 MHz
18 pF	16 MHz

Microchip recommends add 100 nF capacitor between Vdd and Vss power inputs of the PIC18F24J50 or PIC18F26J50 microcontroller. Though the chip works perfectly with 4 MHz crystal even without it, the microcontroller would often fail when it runs on higher frequencies. Here is the circuit for a 12 MHz application:


Vdd
+3V3
4k7
MCLR
RA0/AN0/C1INA/ULPWU/RP0
RA1/AN1/C2INA/RP1
RA2/AN2/VREF-/CVREF/C2INB
RA3/AN3/VREF+/C1INB
VDDCORE/VCAP(2)
RA5/AN4/SS1/HLVDIN/RCV/RP2
VSS
OSC1/CLKI/RA7
OSC2/CLKO/RA6
RC0/T1OSO/T1CKI/RP11
RC1/T1OSI/UOE/RP12
RC2/AN11/CTPLS/RP13
VUSB
PIC18F2XJ50
RB7/KBI3/PGD/RP10
RB6/KBI2/PGC/RP9
RB5/KBI1/SDI1/SDA1/RP8
RB4/PMA1/KBI0/SCK1/SCL1/RP7
RB3/AN9/CTED2/VPO/RP6
RB2/AN8/CTED1/VMO/REFO/RP5
RB1/AN10/RTCC/RP4
RB0/AN12/INT0/RP3
VDD
VSS
RC7/RX1/DT1/SDO1/RP18
RC6/TX1/CK1/RP17
RC5/D+/VP
RC4/D-/VM
10 µF
22 pF
12 MHz
22 pF
100 nF
Vddb = 3.3 V
100nF
Vusb = +5 V
USB con. B
Vusb = +5 V
+3V3
IN
OUT
ADJ
LM317T
LM317T
100N
680R
4,7 µF
680R
1K1
100N