BlueTooth Demoboard for PIC 18Fx

FHDK/B2-001

User Manual

Version 1.2

Version: 1.2
Last revision : July 25, 2003
Printed in Switzerland

Warning
This device is not intended to be used in a medical, life-support or space product.

Any failure of this device that may cause serious consequences should be prevented by implementation of backup systems. The user agrees that protection against consequences resulting from device failure is the user's responsibility.

Changes or modifications to this device not explicitly approved by FiveCo will void the user's authority to operate this device.

INDEX

1. Package and operating conditions

Package contents

- FiveCo Bluetooth DemoBoard
- CDRom with BlueTooth stack, examples, documentations and Win32 programs.
- This manual

Note: Microchip MPLAB ICD 2 programmer/debugger is not included in this kit but is mandatory. It is available from Microchip Distributors (DV164007 = ICD2 with power supply). Latest version of MPLAB IDE can be downloaded from Microchip's web site, version 6.x (or later version) is required.

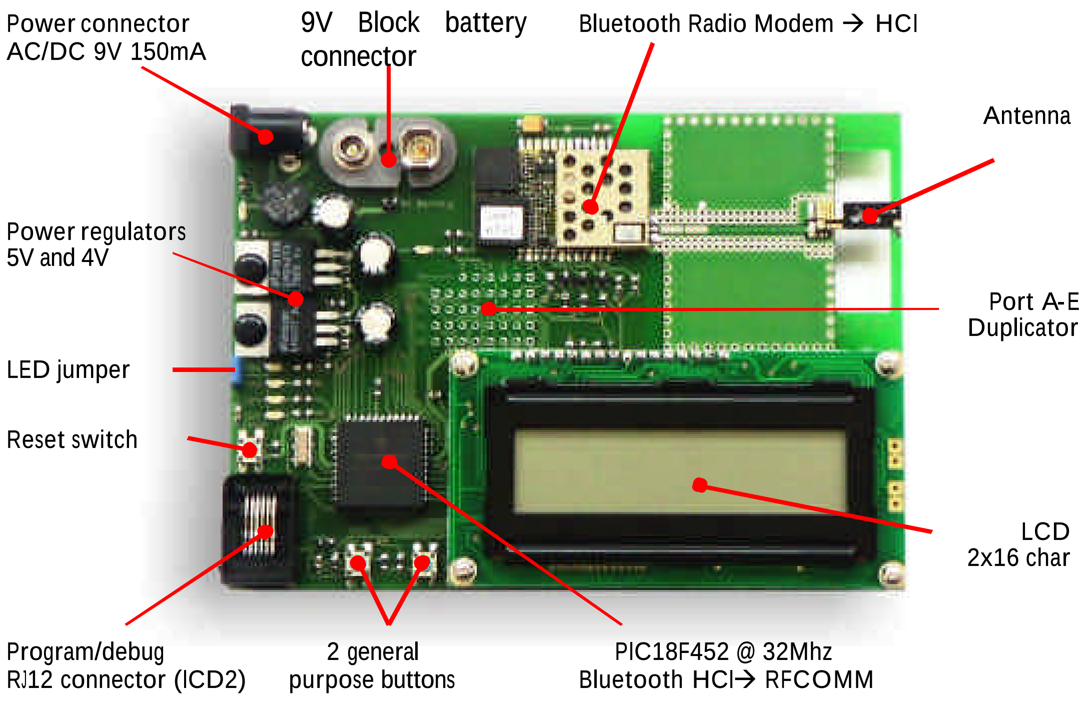
Operating conditions

- Supply voltage: 7-24 V DC or AC (ICD2 power supply connector), or 9v Block battery
- Supply current: 150mA max (70mA typical)
- Operating temperature: 0 – 50 °C

2. Overview

Board Overview

FiveCo's Bluetooth Demo Board is a complete Bluetooth application and does not need extra components to work (except power supply). It provides a Bluetooth Serial Port for data transmission. The user can program his own application on the PIC processor and interact with FiveCo's Bluetooth Stack by the way of functions and events. The user does not need a particular background about Bluetooth specifications to use this kit.


Power connector AC/DC 9V 150mA
9V Block battery connector
Bluetooth Radio Modem → HCI
Antenna
Power regulators 5V and 4V
Port A-E Duplicator
LED jumper
Reset switch
LCD 2x16 char
Program/debug RJ12 connector (ICD2)
2 general purpose buttons
PIC18F452 @ 32Mhz Bluetooth HCI→ RFCOMM

Stack Overview

FiveCo has written the upper layers of Bluetooth stack (HCI ⇔ RFCOMM) in assembly code in order to optimize the RAM, the ROM, and the CPU used. The stack can be integrated in PIC18Fx family microcontrollers without adding extra RAM or ROM extension. Thanks to that feature, the power consumption, design size and cost are optimized.

The stack is already compiled and only needs to be included in PIC applications. The commands are sent from main code through functions, and answers with callback methods.

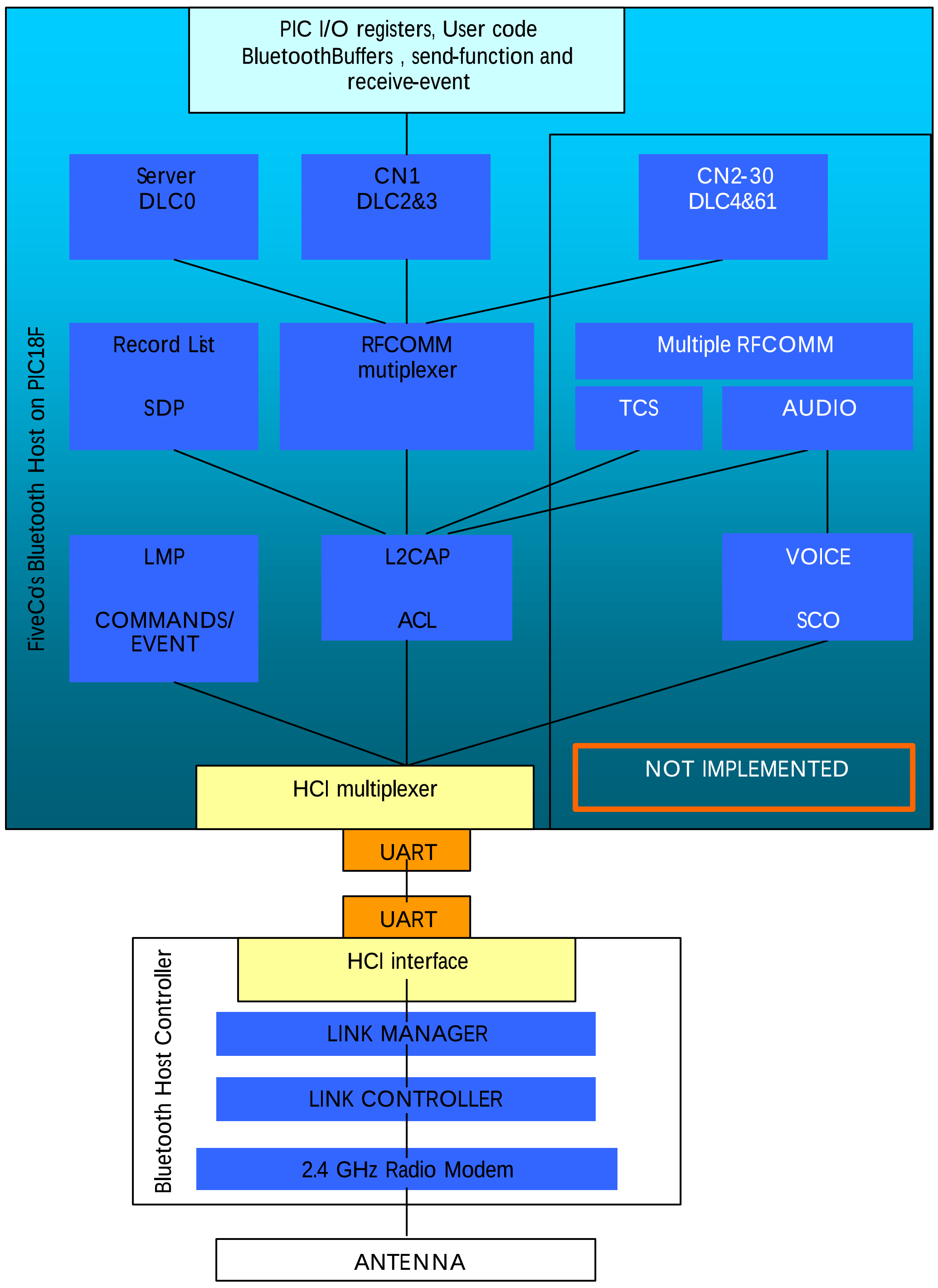
The next pages show that the stack is data transfer oriented. Telephony features (voice data, message exchanges) are not implemented. The easiest way to exchange data with PC or PDA with Bluetooth is to use the emulated serial cable connection defined as “Bluetooth Serial Port”. This feature is totally implemented in the stack. The stack works as a server, and never tries by itself to open a communication with another device. It stays, attending a connection request.

Pairing

If user want to pair (password to access) the module, a unique key is already registered: “fiveco”, other key cannot be added or changed with the precompiled stack.

Module name

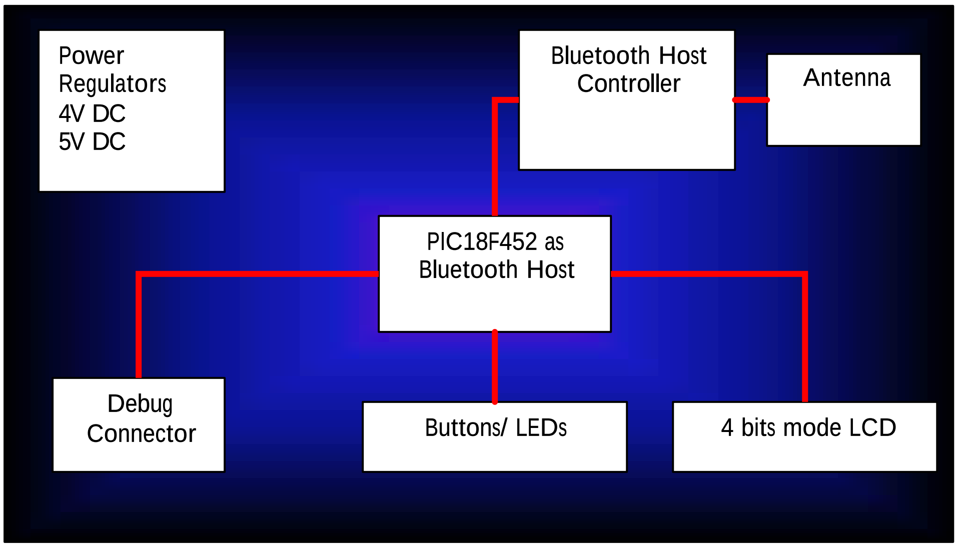
The friendly name of the module can be changed through the software.



FiveCo's Bluetooth stack from HCI to RFCOMM on Microchip's PIC18F family

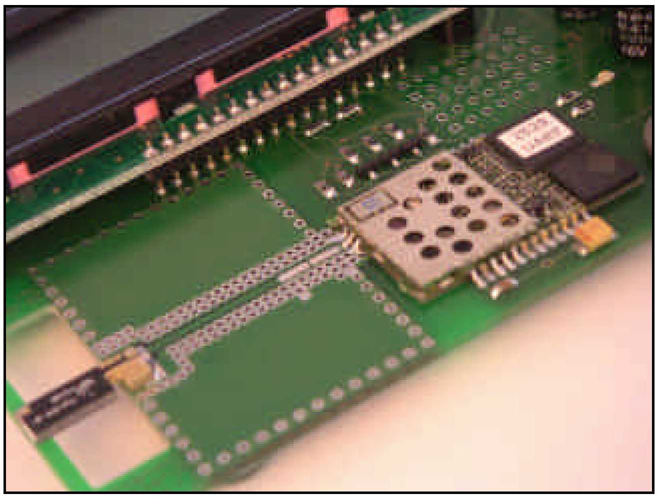
3. Hardware specifications

Schematic blocks



Hardware blocks of demo board

Interface between Bluetooth host and Bluetooth host controller



Bluetooth Host Controller with antenna

The upper layers of Bluetooth stack are implemented on PIC, it communicates with lower layer through standard Host Controller Interface (HCI) via UART. The Bluetooth Host Controller communicates with a baud rate of 115'200 bps, 8bits, 1 stop, with CTS/RTS null modem configuration. It is made with Silicon Wave ICs.

The function "*grBlue2InitRegisters*" performs this configuration with Fosc = 32 Mhz. A new design with another Bluetooth Host Controller and/or with another Fosc must correct baud rate in the SPBRG(1) register, after the execution of "*grBlue2InitRegisters*" function.

DTR is OFF (Vdd) during 100ms and then ON, and never changes after "*grBlue2InitRegisters*" function. DSR is connected but not used.

Bluetooth Host Controller can only manage data packet smaller than 43 bytes (+13 bytes header = 60).

UART transmission is full-duplex

When PIC manages an HCI (control or data) packet it continues to receive a new incoming packet (by interruptions), while an outgoing packet could be sent (by interruptions).

Power Supply

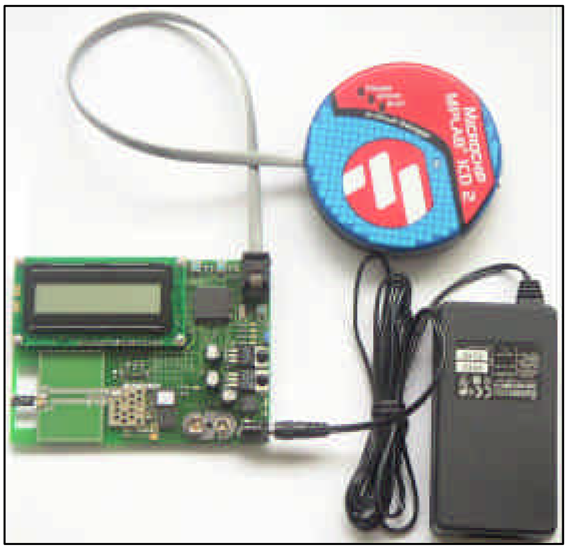
Power can be supplied to the board in two ways:

- 9 V Block battery, with standard connector
- 2 pins power connector, with AC or DC supply with minimum 9 V (max 25 V) (power supply included in ICD2 kit works fine).

Both can be successfully connected at the same time, only the higher voltage supply will be used (serial diode).

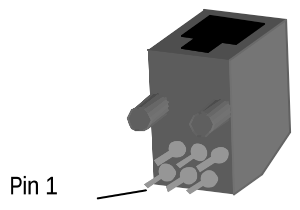
To debug, and run the DemoBoard, 5V on RJ12 cable (Programmer←→DemoBoard) must not be given from programmer module. Just connect the power supply connector to the board (or insert a 9V block battery) and connect the RJ12 6pins cable to your programmer/debugger.
In the case of an ICD2 kit connected through USB, it is sufficient to disconnect the power "jack-type" plug and connect it to the Bluetooth board (see picture).

- Correctly powered PIC works with 5V @32Mhz (8x4 PLL),
- LCD @ 5V,
- Bluetooth module @ 4V.



Programming connector

With Flash PIC, it is possible to program and/or hardware-debug a processor while it is running (e.g. ICD2). To have compatibility with multiple boards of Microchip, the same connector is used on FiveCo's Bluetooth DemoBoard.

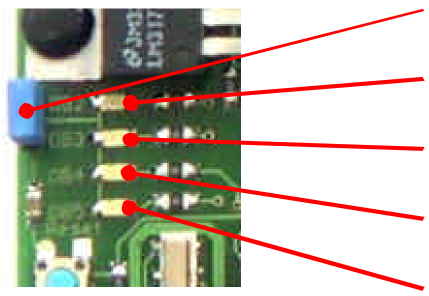


RJ12 Pin description:
VPP (MCLR)
5V (VCC)
0V (Gnd)
PGD (RB7)
PGC (RB6)
not connected

Programming / debugging connection

Module I/Os

- Wireless transmission (In/Out)
- LCD 2x 16 alphanumeric characters (Out)
- 2 buttons not inverted (RB0 and RA4) (In)
- 4 LEDs (Out)



Power jumper LED:	Connect 5V to the LED, disconnect it if port RB2-RB5 is used for custom I/O
Custom LED:	Behaviour changes depending on Bluetooth Serial port: OPEN with ASCII command "changeled"+ CR (carriage return, 0x0D) CLOSED blinking when button(s) are pressed, no packet could be sent
Serial Port Open LED:	State of the channel number 1 (CN1) for Virtual Bluetooth Serial Port. ON if open, OFF if closed
Heart Beat LED:	When there is no problem it blinks every second. Blinks every 100ms when fatal error. Stops blinking if a loop error happens.
CPU LED:	ON indicates the CPU used by interruptions and Bluetooth (all the code), OFF when waiting in main loop.

Stack hardware resources requirements

- Timer1
- UART(1) Port
- 4 UART-control bits on I/O ports
- <32 Access Bank registers
- <750 General Purpose Registers
- less than 7500 instruction words

The exact 3 last values depend on the version of the stack, but can be found in the "*Blue2Stack.def.inc*" file.

The "*Blue2InterruptHigh*" function must be included in HIGH priority interrupt code, to manage the different interrupts of Bluetooth stack. The maximum length of "*Blue2InterruptHigh*" function is 120 instructions (UART transmission, UART reception, Flow Control, Event Launcher, Time Manager) and 14 instructions length in the minimum case if all Bluetooth interruption flags are cleared.

MAX 120 instructions @ 8Minstructions/sec = 15 usec

MIN 14 instructions @ 8 Minstructions/sec = 1.75 usec

PIC18F452 port resources used

Bit \ Port	A	B	C	D	E
0	-	But	-	LCD	-
1	LCD	UART	UART	LCD	-
2	LCD	LED	UART	LCD	-
3	LCD	LED	-	LCD	x
4	But	LED	-	-	x
5	-	LED	UART	-	x
6	x	Prgm	UART	-	x
7	x	Prgm	UART	-	x

x	*does not exist*
-	*free (for custom use)*
But	*buttons*
Prgm	*programming and debugging pins*
UART	*HCI-UART data and flow control lines (CTS/RTS, DTR/DSR)*

4. Software specifications

Files list

The entire stack is contained in only 2 files. A header file "*Blue2Stack.def.inc*" (like *.h in C) and a compiled file "*Blue2Stack.inc*" (like *.dll files).

1. "*Blue2Stack.def.inc*" contains all the information about the Bluetooth stack like:

Version
All Bluetooth functions addresses (ROM) sorted by name
Global Registers (RAM) in access bank sorted by name
Some state bits definition (registers address and bit number)
First free register in each bank (bigger addresses are free)

This file does not contain any code line, only definitions.

2. "Blue2Stack.inc" contains the compiled code.

Ram used

The Bluetooth Stack requires memory in the different banks to work properly, in Access Bank and in General Purpose Bank (Bank1 to Bank3). This size depends upon the version of the stack.

In the "*Blue2Stack.def.inc*" file, one can find the address of the first free space for user registers in each bank.

Ram0Free	set	00000094
Ram1Free	set	000001F3
Ram2Free	set	00000300
Ram3Free	set	0000033C
Ram4Free	set	00000400
Ram5Free	set	00000500
RamAllFree	set	0000001D

So Bluetooth stack uses:

Bank ID	Used from	Used till	Total	Free
Access Bank	0x000	0x01C	29	99
Bank0	0x080	0x093	20	108
Bank1	0x100	0x1F2	243	13
Bank2	0x200	0x2FF	256	0
Bank3	0x300	0x33B	60	196
Bank4	0x400	-	0	256
Bank5	0x500	-	0	256
Total			608	929

Rom (Flash) used

To be compatible with multiple controllers of the 18Fx family, the stack is precompiled at the beginning of the program memory. After the hardware vectors organization (org 0x00, org 0x08, org 0x18), Bluetooth stack requires a fixed memory vector to run properly. It begins at address 0x28.

Callback is a function in user (main) code that is run when an event happens in the Bluetooth stack.

The first lines are used by callback labels from Bluetooth stack to main code. A table of 16 “goto callbackLabelX” is reserved, even if not all are used.
“goto” instruction takes 4 bytes of program memory, so 0x28 + 16 x goto = 0x68 for the stack itself. If user does not want to manage some callback functions, he should just replace a “goto callbackX” by a “return” followed by a “nop” instruction. (return+nop take 4 byte program memory !).

Just after this table, the code of the Bluetooth stack begins at the address 0x68. Following Bluetooth stack, user must write the name of the Bluetooth application visible from other Bluetooth devices. The name is written in UTF-8 standard code (ASCII compatible) with a maximum of 248 bytes. The end of the line is set by a 0x00 byte.

Example:

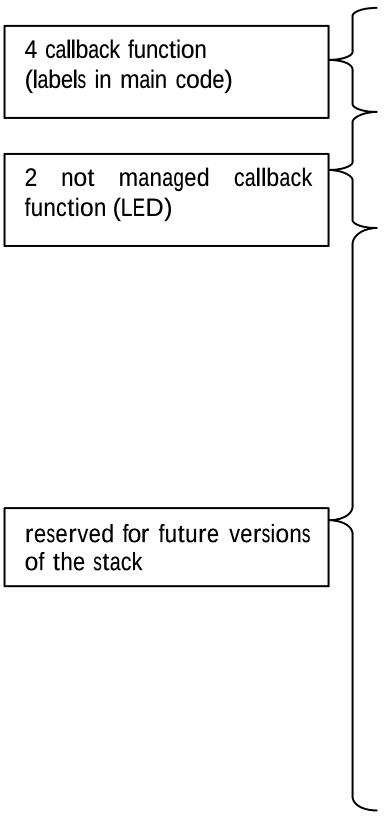
```
...
 org        0x0068
 #include "Blue2Stack.inc"
 db "My Friendly Name",0x00
...
```

After the name, user can write his own program within the remaining program memory.

0x000000 Reset vector
0x000008 HIGH interrupts vectors
0x000018 LOW interrupts vectors
0x000028 Bluetooth Callback vectors Table for 16 goto
0x000068 Bluetooth stack vectors
Next Bluetooth Module name +0x00
Next User code

ROM organisation

Real example of program memory organization:


4 callback function (labels in main code)
2 not managed callback function (LED)
reserved for future versions of the stack

```
ORG       0x0000
          goto      Start
ORG       0x0008
          goto      HighInt
ORG       0x0018
          goto      LowInt
ORG       0x0028
1         goto      FatalError
2         goto      Blue2Connected
3         goto      Blue2Disconnected
4         goto      Blue2PacketReceived
5         bsf       bOpenLed
          return
6         bcf       bOpenLed
          return
7         return    ; futur use
          nop
8         return
          nop
9         return
          nop
10        return
          nop
11        return
          nop
12        return
          nop
13        return
          nop
14        return
          nop
15        return
          nop
16        return
          nop
ORG       0x0068
#include "Blue2Stack.inc"
db        "FriendlyName",0x00
HighInt
...
LowInt
...
Start
...
END
```

Bluetooth stack Input/Output with main code

There are two types of interactions:

- from main code to Bluetooth stack → realized with functions
- from Bluetooth stack to main → callbacks

Callback is a function in user's (main) code that is run when an event happens in Bluetooth stack.

Functions list:

- grBlue2InitRegisters
- Blue2InitModule
- Blue2InterruptHigh
- GetBufferBlue2CN1IN
- InitBufferBlue2CN1OUT
- PutBufferBlue2CN1OUT
- grBlue2CN1OUTSend

"gr" means "goto return table when finished". The return address can change depending on function characteristics. A table of "goto labelx" (4 byte size) can follow this function; refer to each function to know the corresponding return.

General example: *grSquareW* (square of Wreg)
If an overflow occurs (result >255), return is made at "standard return address". If response is correct, return is made at "standard return address" + one Goto instruction length).

```
        movlw           .8
        call            grSquareW                       ; ret 0 overflow, ret 1 correct
        goto            CalculationError                ; case 0 = overflow
        goto            continue                        ; case 1 = correct
```

If user doesnot care about the return table, he can replace each goto by 2 nop instructions (2x2= 4 byte):

```
        movlw           .8
        call            grSquareW                       ; ret 0 overflow, ret 1 correct
        nop                                             ; case 0 = overflow
        nop
        nop                                             ; case 1 = correct
        nop
        …
```

grBlue2InitRegisters

Init all registers and I/O peripheral used by the stack. If UART control bit DSR stays OFF(1) after 100ms return + 0. If DSR is ON(0) return + 1.

Blue2InitModule

Must be called after activation interrupts; it configures the Bluetooth Host Controller with the set of HCI commands (e.g. module name).

Blue2InterruptHigh

Must be called once during interruptions HIGH to manage Bluetooth Interruptions.

GetBufferBlue2CN1IN

Gets one byte into Wreg from Bluetooth IN-buffer of channel number 1 (CN1). Channel 1 is the one used in RFCOMM layer to transport Bluetooth Serial Port Data. Must be called only during "*Blue2CN1EventPacketReceived*" callback (callback table id 4). Before calling this function, test if buffer contains at least one byte. "Blue2CN1INBufferFilled" register gives you this information.

InitBufferBlue2CN1OUT

Reset Bluetooth OUT-buffer of channel number1.

PutBufferBlue2CN1OUT

Add a byte (Wreg) to Bluetooth OUT-buffer of channel 1. Before filling a packet, call "InitBufferBlue2CN1OUT" function.

grBlue2CN1OUTSend

Test and send Bluetooth buffer of Channel 1 as output. The return is a goto table:

```
ret+0   Error Buffer too big (always), compared with HCI data
packet size.
ret+1   Channel 1 not open.
ret+2   Not enough resources (RAM), try later.
ret+3   OK.
```

Callback list

Callback label list of functions in main code:

- FatalError (1)
- Blue2Connected (2)
- Blue2Disconnected (3)
- Blue2CN1EventPacketReceived (4)
- Blue2CN1Open (5)
- Blue2CN1Close (6)

The callback list begins at program memory address 0x000028 and is able to contain 16 instructions “goto CallbackFunctionX”. Every goto instruction needs 4 bytes, so if user does not want to manage an event he can just write a “return” + “nop” (2+2=4 bytes) instead of each unused “goto” instruction.

FatalError

Is launched to indicate a critical error in the stack. (eg. In SDP server, if service record list is not coherent)

Blue2Connected

When a new device has been connected (new handle, HCI layer).

Blue2Disconnected

When a device has been disconnected, it is launched if a transmission is lost.

Blue2CN1Open

Channel 1 (Bluetooth serial port channel, RFCOMM layer) in now open. This event can only appear after a “Blue2Connected” event.

Blue2CN1Close

Channel 1 has been closed, (RFCOMM layer). Not launched if transmission is lost.

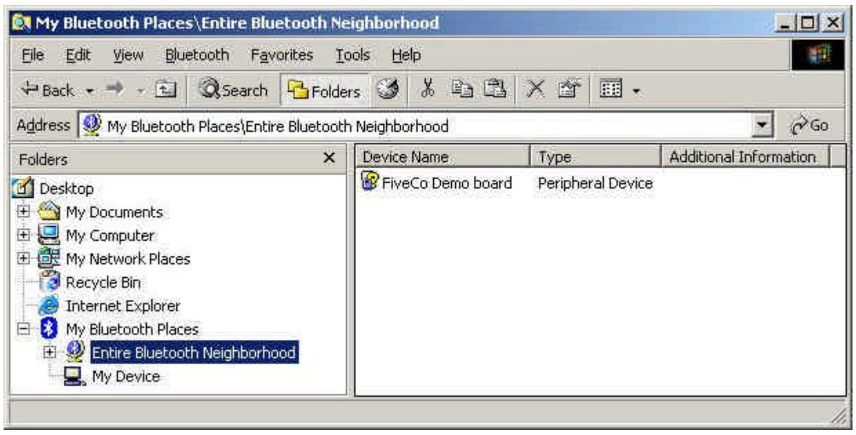
Blue2CN1EventPacketReceived

Channel 1 as input has received a packet in a buffer. Data can be extracted with “Blue2CN1INBufferFilled” register and “GetBufferBlue2CN1IN” function.

5. Easy use

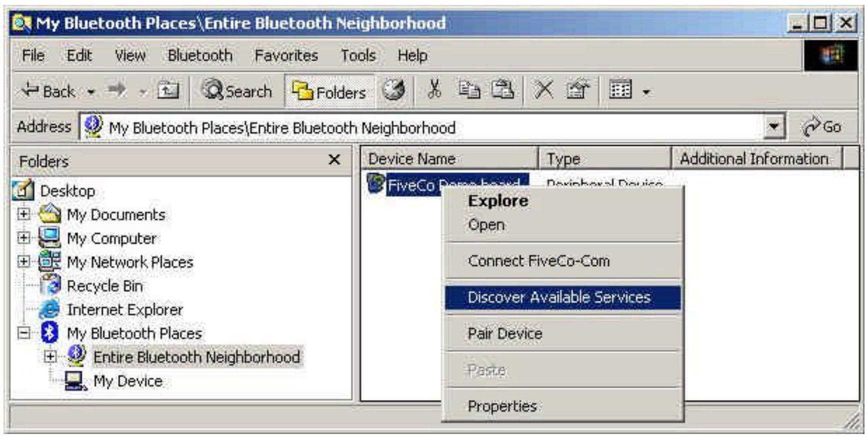
Testing the board

- User needs an ASCII terminal if the Microsoft Windows® operating system is not use
- User should put a Bluetooth dongle on the computer (or other Bluetooth device).
- User should install the driver/software (stack) of the dongle for the operating system.
- User should copy CDROM:\software\Terminal.exe on the desktop.
- User should put the 9V block battery on FiveCo Bluetooth DemoBoard, see HeartBeat LED blinking. The LCD displays messages.
- User should start dongle software, and scan for Bluetooth Neighbo rhood, the DemoBoard should be detected.

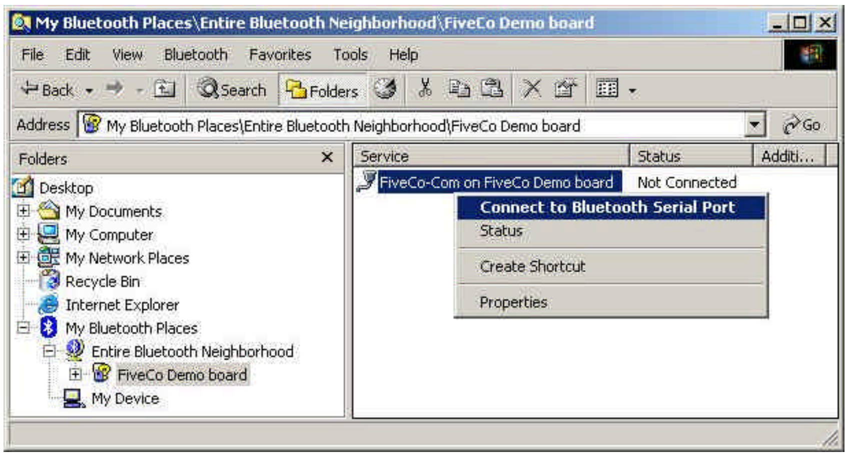

My Bluetooth Places\Entire Bluetooth Neighborhood
File Edit View Bluetooth Favorites Tools Help
Back Search Folders
Address My Bluetooth Places\Entire Bluetooth Neighborhood Go
Folders
Desktop
My Documents
My Computer
My Network Places
Recycle Bin
Internet Explorer
My Bluetooth Places
Entire Bluetooth Neighborhood
My Device

Device Name	Type	Additional Information
FiveCo Demo board	Peripheral Device	

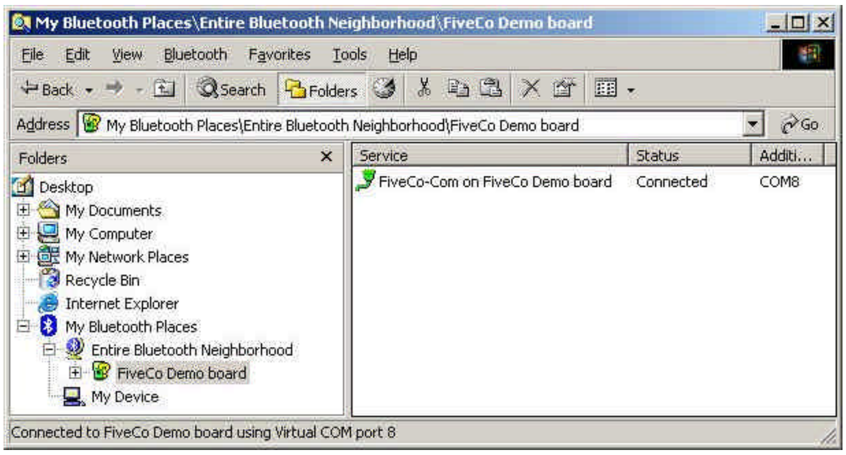
- Detect available services of DemoBoard.



- Connect to the detected Bluetooth Serial Port. If pairing is asked, user should type “fiveco”. Note on which COM port it is connected. On the DemoBoard LCD, user shou ld see “Connected”.


My Bluetooth Places\Entire Bluetooth Neighborhood\FiveCo Demo board
File Edit View Bluetooth Favorites Tools Help
Back Search Folders
Address My Bluetooth Places\Entire Bluetooth Neighborhood\FiveCo Demo board
Go
Folders
Desktop
My Documents
My Computer
My Network Places
Recycle Bin
Internet Explorer
My Bluetooth Places
Entire Bluetooth Neighborhood
FiveCo Demo board
My Device
Service
Status
Additi...
FiveCo-Com on FiveCo Demo board
Not Connected
Connect to Bluetooth Serial Port
Status
Create Shortcut
Properties

First click on connect


My Bluetooth Places\Entire Bluetooth Neighborhood\FiveCo Demo board

File Edit View Bluetooth Favorites Tools Help

Address: My Bluetooth Places\Entire Bluetooth Neighborhood\FiveCo Demo board

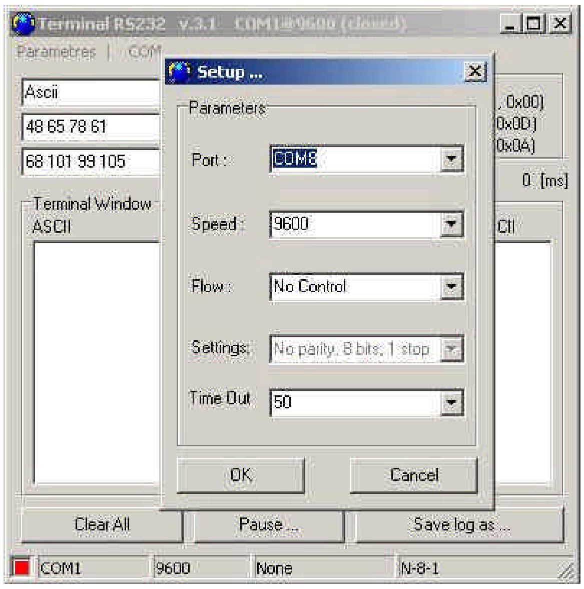
Service	Status	Additi...
FiveCo-Com on FiveCo Demo board	Connected	COM8

Connected to FiveCo Demo board using Virtual COM port 8

Connected on COM8

- on LCD, user can see:

- Start Terminal.exe (or another ASCII terminal) opening the corresponding port. (baud rate is not used).



Terminal RS232 v.3.1 COM1@9600 (closed)

Setup ...

Parameters

Port : COM8

Speed : 9600

Flow : No Control

Settings: No parity, 8 bits, 1 stop

Time Out 50

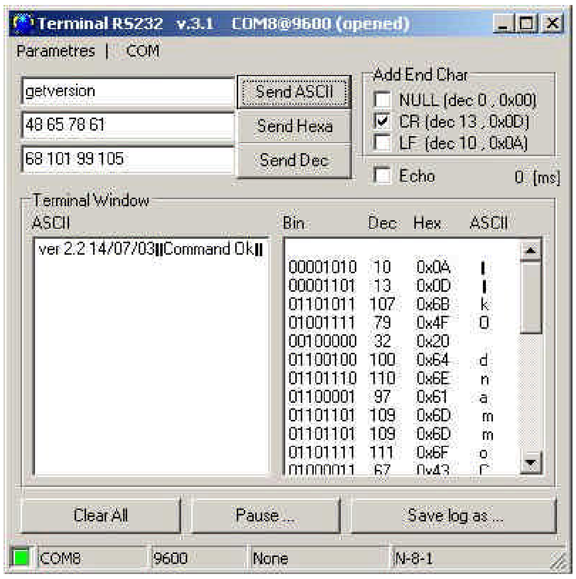
OK | Cancel

Ascii
48 65 78 61
68 101 99 105

COM1 | 9600 | None | N-8-1

- Choose same COM port number

- Write "getversion" + carriage return (0x0D)



Terminal RS232 v.3.1 COM8@9600 (opened)

getversion — Send ASCII
48 65 78 61 — Send Hexa
68 101 99 105 — Send Dec

Add End Char
- [ ] NULL (dec 0 , 0x00)
- [✓] CR (dec 13 , 0x0D)
- [ ] LF (dec 10 , 0x0A)
- [ ] Echo 0 [ms]

Terminal Window

ASCII:

```
ver 2.2 14/07/03||Command Ok||
```

```
Bin       Dec  Hex   ASCII
00001010  10   0x0A  |
00001101  13   0x0D  |
01101011  107  0x6B  k
01001111  79   0x4F  O
00100000  32   0x20
01100100  100  0x64  d
01101110  110  0x6E  n
01100001  97   0x61  a
01101101  109  0x6D  m
01101101  109  0x6D  m
01101111  111  0x6F  o
01000011  67   0x43  C
```

COM8 | 9600 | None | N-8-1

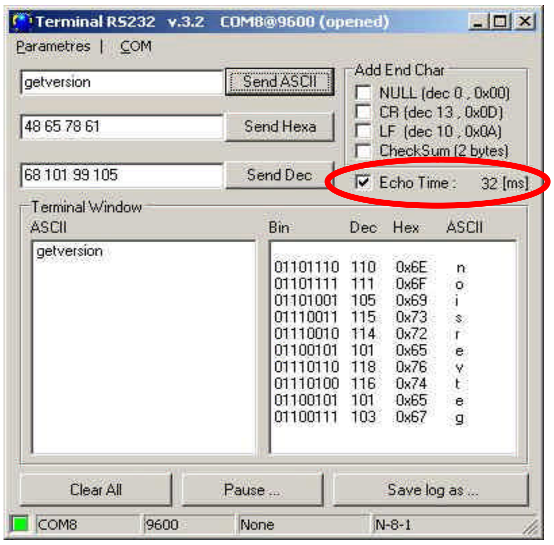
write "getversion", select CR, Send ASCII

- User should receive an ASCII response "version x + date" and on LCD "Getversion"

- Try other ASCII commands: "changeled", "writelcd helloworld", "getversion".

- Press on the 2 buttons near the LCD, nothing change on LCD, but they send information back to the terminal every 500ms.

- Select on Terminal "Echo Time" to see the delay of a packet to be sent and the response received


Terminal RS232 v.3.2 COM8@9600 (opened)

Parametres | COM

getversion — Send ASCII

48 65 78 61 — Send Hexa

68 101 99 105 — Send Dec

Add End Char
- [ ] NULL (dec 0 , 0x00)
- [ ] CR (dec 13 , 0x0D)
- [ ] LF (dec 10 , 0x0A)
- [ ] CheckSum (2 bytes)
- [✓] Echo Time : 32 [ms]

Terminal Window

ASCII
```
getversion
```

```
Bin       Dec  Hex   ASCII
01101110  110  0x6E   n
01101111  111  0x6F   o
01101001  105  0x69   i
01110011  115  0x73   s
01110010  114  0x72   r
01100101  101  0x65   e
01110110  118  0x76   v
01110100  116  0x74   t
01100101  101  0x65   e
01100111  103  0x67   g
```

COM8 | 9600 | None | N-8-1

- When exiting the program, and closing COM port, connection is deactivated. LED for Bluetooth Serial Port Channel switches OFF first (RFCOMM), then on LCD you should read "Disconnected" (HCI).

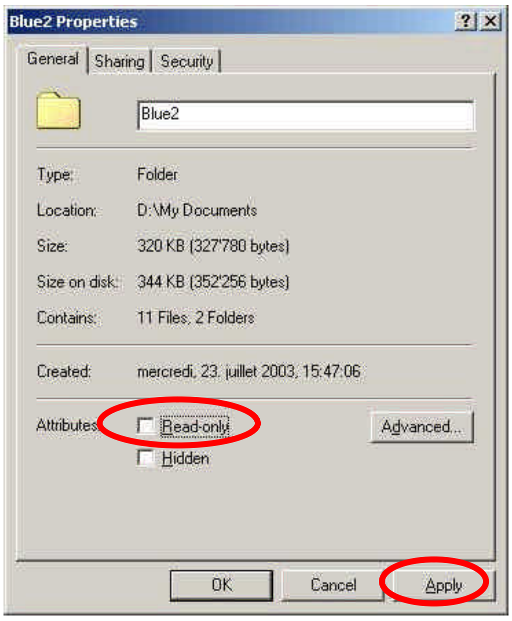
Programming the board

Copy entire directory CDROM:\software\Blue2 somewhere on your hard disk (e.g. C:\MyDocuments\Blue2) .

BEWARE:

If path + file name > 60 characters, MPLAB IDE 6.10 cannot debug correctly.

Deactivate “read-only” protection on all copied files, by selecting the “properties” of the directory.



Blue2 Properties

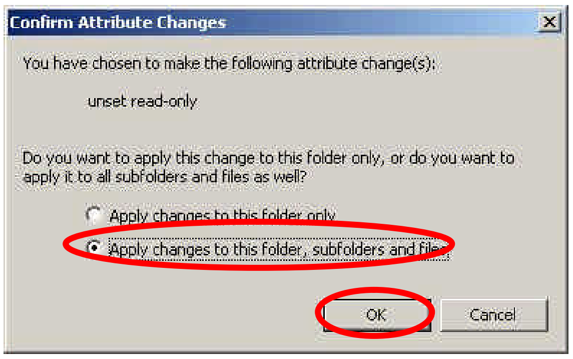
General | Sharing | Security

Blue2

Type: Folder
Location: D:\My Documents
Size: 320 KB (327'780 bytes)
Size on disk: 344 KB (352'256 bytes)
Contains: 11 Files, 2 Folders

Created: mercredi, 23. juillet 2003, 15:47:06

Attributes: [ ] Read-only [ ] Hidden



Confirm Attribute Changes

You have chosen to make the following attribute change(s):

unset read-only

Do you want to apply this change to this folder only, or do you want to apply it to all subfolders and files as well?

( ) Apply changes to this folder only
(●) Apply changes to this folder, subfolders and files

- Start MPLAB IDE 6.x
- Configure ICD2, cable, power, connect to FiveCo Bluetooth DemoBoard, processor version should be detected by MPLAB IDE.
- Open from “Blue2Demo.asm” from your CDrom copy.
- Modify it as needed.
- Compile it (processor type is by default 18F452)
- Program Flash (with or without debug)
- Run it

6. Examples

Example 1: Template

Files needed:

• Blue2BoardTemplate.asm	
• Blue2Stack.def.inc	for all Bluetooth stack I/O definition
• Blue2Stack.inc	stack compiled

In the directory CDROM:\software\Blue2\Template, user will find the minimum required to make the first application with FiveCo's Bluetooth stack.

Open the main file "*Blue2BoardTemplate.asm*". The file includes the two files of the Bluetooth stack. Complete user own code every time the following appears

```
*** add your code/macro/function/register… here ***
```

In this example no button is used, neither LCD, but only two LEDs.

LED on RB2 is ON when the "Bluetooth Serial Port Connection" is open. With an ASCII terminal send character "0" to switch off LED (RB3), "1" to switch on or "2" to change LED state. Just after receiving a packet, PIC answers with "OK ".

Example 2: Complete board application

This example is already programmed when receiving the HDK.

Files needed:

- Blue2demo.asm
- Blue2demo.fct.inc main custom functions of Blue2demo.asm

LCD.fct.inc	functions to use LCD
Table.fct.inc	functions for “goto” or “branch” table
Buffer.mac.inc	buffers macros : creation, put, get
ASCIIBuf.fct.inc	useful functions to manage ASCII words in buffer
Blue2Stack.def.inc	for all Bluetooth stack I/O definition
Blue2Stack.inc	compiled stack

This example presents interaction with buttons, LCD, all the LEDs. It answers to three ASCII commands: “getversion” , “writelcd helloworld” , “changeled”.

Each of those commands must be followed by a carriage return to be executed. It is possible to send in one packet mutliple commands: “getversion changeled”+ CR. Or a command can be sent in multiple packets: “getv” send, “ersion” send, CR send.

7. Compatibility PIC18Fx Family

FiveCo's Bluetooth stack can be transferred to other microcontrollers of Microchip's 18Fx family with 16bits word instruction size. If RAM and ROM requirements are covered, and peripheral UART(1), CTS, RTS and timer 1 are on-chip, then other PIC18F than PIC18F452 could be used:

Needed:	RAM	: up to Bank 3
	ROM	: minimum 7500 word (15 kb program memory)
	UART	: to communicate with the Bluetooth Host Controller

PIC with extended capability of the 8Fxxxx family can also be used.

Picture: Some of Microchip PIC 18F family

Contact address :

FiveCo - Innovative Engineering
PSE-C
CH-1015 Lausanne
Switzerland
Tel: +41 21 693 86 71
Fax: +41 21 693 86 70

www.fiveco.ch
info@fiveco.ch