INSIDER V1.0 USER'S MANUAL

By E.E. Atanasios Melimopoulos (pic.insider@gmail.com)

INTRODUCTION:

In-Circuit-Debuggers, as you may already know, have become the PIC’s debugging standard tool for many programmers because it’s easy use and handy interface to the target pic-placed-board. They come with MPLAB plug-ins that provides a full rich set of commands and functions in order to debug your code in real time.

After hours of using some brands of ICDs, ICD2, etc. on different projects, I faced some hardware situations where the two pin interface ICD <-> PIC becomes annoying and sometimes difficult to work around. Apart from the fact that your target pic must run at selected clock frequencies that allows the ICD-Uart baudrate multiplier to fit. Also, some pics do not allow the same on-hook commands upon which ICDs are based. There is no electrical isolation between the pic-target board and the USB–Serial PC-GND interface.

Thinking about it, I decided to build the INSIDER, based on the following:

- 1Bit Interface placed on ANY I/O selected spare pin (or carefully shared).
ICD 2bit interface is too much, usually on 12Fxxx 8-pin pics where there are only 6 I/O pins. Also on bigger pics, like 16Fxxx and 18Fxxxx where the RB port is the most useful, the fact that RB6-RB7 are forced to be the 2bit ICD interface, may goes against your hardware I/O connections and sometimes you can’t share their use.

- 1Bit Interface independent of the target PIC clock speed from 20KHz to 50MHz.
ICD 2bit interface communicates to the target pic via serial ASCII link, that’s why the pic clock freq must be close to an integer multiple of the chosen baudrate.

- 1Bit Interface independent of the target PIC type (12Fxxx, 16Fxxx, 18Fxxxx)
ICD commands and features are based on some bootstrap routines that Microchip places on most of its MPU types. But depending of the devices class, family and generation, this routines differs and so the ICD implementation of this functions.

- Electrical Isolation between the target PIC hardware and the PC workstation.
PICs MCU are very useful on AC-line applications where the pic floats driving a MOSFET H-Bridge PWM converter or a Triac controlled power assembly, etc. That’s why it is so important to isolate the PC from the Target-PIC using an isolated connection ICD tool.

- Small, easy to build, easy PC interface and very easy to use.
With a double side 1”x 2” PCB, just one 18pin dip 16F628 pic, a dual-optocoupler, leds, a pushbutton, some resistors, serial connector, etc. I think this is a small and easy to build circuit. With no more than 10 well designed commands and the most basic interface available: Hyperterminal using the PC serial-Ports.

Against some opinions, PC serial comm ports are not obsolete, my Dual Core PC desktop has two and my Core 2 Duo Laptop (HP nx6320) has one. Of course I could place an USB-Uart bridge chip in order to use the PC USB port (maybe later) and you can find a handy PCI card serial port for your Desktop PC. (USB-serial-port cable doesn’t work).

By now I have no time to design and build an IDE Windows interface that would allow Symbolic cross-references to the user program to be debugged, the interface between the INSIDER and the PC is via Hyperterminal. So, all the variables and address references to the user program are absolute HEX, although not being so friendly because of this. At least, the INSIDER’s Commands Set is designed to ease the HEX management as much as it was possible. In the near future, this Command set will be the console commands upon which the IDE windows based INSIDER Program will work (Anyone wants to help? Welcome).

This is a non-profit based project so the same is applicable to any other source of contribution sent and shared with.
There is no responsibility to the author about any kind of damage, injuries and consequences of any type in projects where the INSIDER will be involved.
Any feedback, contributions or modifications needed to build and IDE windows program for the INSIDER, are welcome and you can contact me via email: pic.insider@gmail.com

INSIDER CONCEPT:
Although some aspects of ICDs implementations, PIC Hardware Emulators, Soft Simulators, Starter Kits, etc, has been taken into the INSIDERs design. The final implementation differs on its actual usage; I think it will be useful to define the INSIDER tool as a concept in order to understand its use, commands and installation.

- Real Time Debugging – That’s the big issue. No simulator available can give the same behavior as the real thing does, apart from their bugs, most of the time it is very difficult to simulate the in/out environment connected to your pic-target hardware.

Emulators are excellent, but you need to have expensive ic-pods for each type of pic you may use and a complete working platform is very expensive.

In Circuit Debugger (ICD) is the best cost/performance tools available, low price, easy to use and work in real time with your pic-target board. But they force you to leave two I/O pins, RB6-RB7, for its interface. All pics ICD bootstrap software implementations are not the same so all ICD’s commands are not available to all pics types. And you can’t use any oscillator frequency except those that close fit the baudrate multiplier used in the Uart-like pic-ICD interface.

The ICD concept relies on the strategic deployment of as many breakpoints needed in you target pic-code, so whenever your code reach at any breakpoint, it will call and execute the bootstrap ICD routine placed at some address outside the user code address range.

The started bootstrap ICD routine will initiate an ASCII serial communication via RB6-7 with the ICD’s external hardware which handles the entire User command interface on behalf of specially designed plug-ins on the MPLAB IDE interface upon which you can view, edit any data, SFR register or code area, figuring out what happened with your halted code and rerun it from the last breakpoint address or from another user selected address until the next breakpoint is reached.

As you can see, the ICD concept is based on a carefully placement of many breakpoints wherever it makes sense to stop/run your code giving the opportunity to review and edit your program variables, flags, SFR contents, etc. Also, you can force any output or read any input in order to check any interface problem with your connected hardware.
Of course ICD has more functions but the key ones are the breakpoints related.

The INSIDER concept also relies on the strategic deployment of as many breakpoints needed in you target pic-code, but it only needs one user selected I/O pin for its interface and it can be carefully shared with any simple input/output user connected hardware like led, pushbutton, pull-up external input, etc. The INSIDER interface protocol is capable to filter any I/O activity, external or from the user code, from its actual data link. Any pic type can be used (12Fxxx, 16Fxxx, 18Fxxxx) any frequency from 20KHz to 50MHz, from VCC 3.3V to 5V.

The main difference between the INSIDER and ICD is that its pic-bootstrap routine executed from any breakpoint must be included at the end of the user target code. Although being small it takes up to: 160(12Fxxx), 170(16Fxxx), 400(18Fxxxx) prog-words and needs no more than 6(12Fxxx/16Fxxx) or 7(18Fxxxx) data bytes.

The other difference is that breakpoints must be placed and flashed in the user code source, so there is no way to clear or change them at run/debug time. But commands are included in order to mask/unmask/trace selected breakpoints as they are debugged reducing the times the user code must be edited/flashed to clear/insert any breakpoints. When the user code halts at any unmasked breakpoint and after viewing-editing the pic target data and resources, you can rerun (G cmd) the code from the halted breakpoint address or reset you application to start again, there is no way to change the rerun address on 12Fxxx and 16Fxxx pics. But on 18Fxxxx you can change the rerun address from the halted breakpoint (G nnnnnn cmd).

There is no opcode-trace command, INSIDERs Debug sessions relies on the strategic deployment of up to 256 Breakpoints (00-FF break number) and its view/edit commands active when the user code halts at any unmasked breakpoint.
But deployed breakpoints are traceable, so you can trace your code execution.

INSIDER OPERATION:
Once you have your code written and ready to be tested on a real hardware, you will have to insert the INSIDER's bootstrap routine and deploy as many breakpoints as needed to debug your code wherever it makes sense to your code logic.

There are 3 items to insert in your code, 2 macros definitions and a bootstrap routine. Assuming an assembly code, you have to insert at your code definition start area where you define all your variables, constants and macros, the following 2 INSIDERs macros:

```
DBRK  EQU   1     ;This EQU will ENABLE/DISABLE(1/0) all the INSIDER items in your code.
DBEE  EQU   1     ;This EQU will ENABLE/DISABLE(1/0) EEPROM-READ-CMD, If you don't want
                  ;or your pic doesn't have EEPROM area, this will reduce bootstrap size

INSIDER MACRO     ;INSIDER MACRO DEF, Right after the power-up reset init code, after
 IF DBRK==1       ;setting up all you pics resources and variables, Placing the code
      CALL  DG000 ;line: INSIDER, will stop your code execution, so you can START it
 ENDIF            ;from the Hyperterminal interface pressing G (INSIDERs G cmd).
      ENDM

BREAK MACRO bkn   ;BREAK MACRO DEF, At any code line wherever you decide to place a
 IF DBRK==1       ;breakpoint, you have to insert the code line: BREAK nn, where nn is
      MOVWF DB0   ;the HEX number (00H-FFH) that identifies your breakpoint at the
      MOVLW bkn   ;Hyperterminal screen during your debugging sessions.
      CALL  DBUG
 ENDIF
      ENDM
```

At last, you have to place the PIC-specific Bootstrap at the end of the code memory, being careful to do not overlap the clock calibration const usually placed at the last code-memory address on some pics. There are 3 bootstrap routines to choose from:

18FDBUG:	For 18Fxxxx pics
16FDBUG:	For 16Fxxx pics
12F6DBUG:	For 12F6xx pics 14Bit opcodes (like: 12F615, 12F629, 12F683, etc)

Each Bootstrap source code includes its specific MACROS and variables definitions. So you can cut and paste them to your target code. They are carefully designed to preserve the user's target code environment (W, STATUS, FSR, INTCON, etc) on every breakpoint.
Once your code execution is halted by a breakpoint, The INSIDER's commands will aid you to view/edit the real users DATA/SFR registers contents, and any changes you made will be reflected on the code rerun from the last breakpoint until the next one.

After saving the users INTCON value, any enabled interrupt will be masked during the INSIDER's execution of a breakpoint by clearing the GIE-bit in the INTCON SFR. Then restored to the saved or user edited value when code is rerun after the breakpoint.

It is very important to define the PIC I/O-Pin-Port to be used as the INSIDER's 1Bit interface; this is done by setting it in the header of the respective Bootstrap routine.

Debug sessions begins by first starting Hyperterminal and configuring the PC comm port to: 38400BPS, 8Bit, 1 stop, No Parity, No Hand-Shake, ANSI, No-ECHO, Word wrap active.

Second, connect the 1Bit INSIDER's Interface to the target selected I/O pin and GND. Attach the INSIDER's port to the PC-serial port, then press the INSIDER's RESET Push Button to make it show its start up prompt (in case it doesn't show up automatically).

Third, power up the target board with all the INSIDER's macros, bootstrap routine and breakpoints included in the user code already flashed in the target pic. So your code will start and run until it stops at the INSIDER code line waiting for the G command.

Fourth, whenever you are ready with your hardware, start your application issuing a G command from Hyperterminal. Then your code will run until a Breakpoint is reached.

Fifth, when your code halts by a Breakpoint, The INSIDER will show you thru the Hyperterminal screen: the Breakpoint number, WREG and STATUS contents and the PIC-freq.

Sixth, with your code halted, you can view-edit-display any DATA/SFR register.
Read/Write any port in order to check your hardware I/O.
Mask/UnMask/Trace any deployed breakpoint; view the EEPROM area, etc.

Seventh, when you are ready reviewing you PIC target variables and resources, you can rerun your code from the last breakpoint by pressing G, rerun from other selected address by pressing G nnnnnn (18Fxxxx only) or reset your application together with the INSIDER and start again the debugging session. Also, depending of the errors found, you may want to edit your code and relocate the Breakpoints and start from the beginning.

This cycle (3-7) will be repeated until all your code is debugged and works properly.

The 1Bit interface between the INSIDER and your hardware is a high impedance input; it will never source or drain any power to the target, so you don't need to disconnect it from your target during its power cycle.

There are two jumpers aside the INSIDER's DB9 port and a 9V-battery holder underneath.

When there is no battery and the jumpers are placed, the INSIDER is powered from the PC comm port and the PC-GND is connected to your target GND (no isolation).

When there is a battery and the jumpers are not placed, the Insider is powered from the battery and the PC-GND is NOT connected to your target GND (isolation).

The INSIDER has three operating modes:

ON-RESET: (Leds OFF)
This is the Power-up Reset state of the INSIDER. Whenever it is connected to the PC-comm port or the Reset Pushbutton is pressed, it will be initialized and a start-up prompt message will be shown in the user's terminal. It would be necessary to reset the INSIDER on those occasions when there is no connection with the target or the sync is lost due to some debug error. Any command and breakpoint setup status are preserved, a few of the INSIDER's commands are available because most of them depends on the target response.

ON-SCAN: (Led Green ON)
After the INSIDER reset, there is no interaction with the target, so it must be started issuing a G cmd. Before, the target code has been started and reached the INSIDER macro, where it waits for the INSIDER's G cmd to start running.
Also, after a Breakpoint, the target waits for the INSIDER's G cmd to start running from the Breakpoint address.
On Both situations, after executing the G cmd, the INSIDER enters to ON-SCAN mode where it waits for a target's breakpoint query. In this mode, the INSIDER will not respond to any command from the User's Terminal. Thus, the Led Green ON indicates: Target Code Run.

ON-BREAK: (Led Red ON)
This is the INSIDER's main mode where all commands are available to the User's terminal.
Whenever the target code reaches any enabled breakpoint address, a breakpoint query is sent to the INSIDER making it enters to ON-BREAK mode showing the BREAK number, WREG and STATUS contents and the pic-freq. After that, the target will wait and respond to any of the INSIDERs debug commands issued form the User's terminal.
On this mode, the User can view/edit all the target's DATA/SFR area and READ/WRITE to any of the pic's ports in order to test any hardware interface connected to it.
Whenever the User is ready reviewing the code/DATA and decides to continue by executing G cmd, the target code will run from the breakpoint, making the INSIDER enters ON-SCAN mode again, until another breakpoint. Thus, the Led Red ON indicates: Target Code Stop.

INSIDER COMMANDS:
Each time you press H from the Hyperterminal keyboard, the INSIDER will show you a quick reference Help for the 10 commands available, each one is explained next.
For any numeric field entered like nnn sss fff dd, only the last needed digits will be entered. For example: for nnn 345ABCD7FE001 only 001 will enter, so **don’t backspace**.
In case of an address range sss.fff: FE67109.CCB2FF only 109.2FF will enter.
(CR), (SP/CR), (+/-), means pressing: ENTER, SPACE or ENTER, + or -, etc.

S[T]{nnn nnn nnn}(CR)
Show Variables: The INSIDER uses hex address to reference your data. It would be useful to store the address of the same variables you view again and again on every breakpoint.
S cmd allows you to store up to 16 nnn Hex data address to show its contents.
On any mode, Snnn nnn nnn nnn(CR) will store the entered address (up to 16).
ST will toggle AUTO ON/OFF, AUTO ON will execute S(CR) after a breakpoint query.
ON-RESET, S(CR) will only show the stored address.
ON-BREAK, S(CR) will show the s-variables contents (HEX and Binary) in the stored order.

V/X/T[N] n n n(CR)
Enable/Disable/Trace Breakpoints Groups: Breakpoints can be placed from 00H to FFH.
It would be useful to enable/disable/trace selected breakpoints as you go forward debugging your code, this will reduce the assembly-flash-cycle of the target pic every time you need to remove/insert breakpoints in your code. Also, you can deploy the breakpoints by groups, for example: breakpoints 00-0FH (group-0) over the code start up, breakpoints 10H-1FH (group-1) over the math routines, breakpoints 20H-2FH (group-2) over the uart routine, and so on. This way by disabling group 0 and 2, only group-1 breakpoints are allowed to halt your code in order to debug the math routines.
You can trace your code execution by tracing the deployed Breakpoints as your code pass through them without halting but showing each breakpoint number in sequence. So, you can see your code execution path on behalf of the fast displaying breakpoints numbers.

```
Executing:  V2 5 3 E(CR)      XA 4 1 F(CR)    TD C 8 9(CR)
```
Will enable: 2-3-5-E, disable: 1-4-A-F, trace: 8-9-C-D break-groups at the same time.
VN(CR) enables, XN(CR) disables, TN(CR) traces -- all break-groups.
V(CR) or X(CR) or T(CR)-- will show the break-groups enable/disable/trace status.

Disabling all break-groups by XN(CR) cmd will let your code run without halting, but the break-query is actually made on every masked breakpoint and the INSIDER will rerun your code automatically until any User-key-press will enable the next Breakpoint.
This will add some delay on your code execution and differs from making ‘DBRK EQU 0’ in the INSIDER bootstrap header making all INSIDERs items disappear from your code.
Tracing all break-groups by TN(CR) cmd will let your code run without halting, but on each break-query only the breakpoint number is displayed and the INSIDER will rerun your code automatically until any User-key-press will enable the next Breakpoint.

G{nnnnnn}(CR)
GO command: Allows you to run your code whenever you are ready with your hardware.
ON-RESET, G(CR) initiates your code execution after its power-up and the INSIDER enters ON-SCAN mode waiting for a breakpoint query. When it occurs, the INSIDER will enter ON-BREAK mode, where you can view/edit any DATA/SFR using the INSIDER commands.
Once you are ready to continue, you must issue G(CR) in order to rerun your code from the last breakpoint. On pics 18Fxxxx you can specify a run address different from the last breakpoint by issuing Gnnnnnn(CR) where nnnnnn is the address you want to run from.
On 18Fxxxx the nnnnnn address will always be forced to be even (LSBit=0).
Note that if you issue a Gnnnnnn(CR) cmd on pics type 12F6xx, 16Fxxx the target will ignore the nnnnnn address and will run from the last breakpoint always.

N   Is the same like pressing G(CR) but only works just after a Breakpoint query.
This will speed up the rerun-cycle/auto-show-variables on many successive breakpoints.

E nnn (CR)   (ON-BREAK MODE ONLY)
EEPROM Dump Display: If your code uses the EEPROM DATA AREA, the ‘DBEE EQU 1’ must be set in the INSIDER bootstrap header to enable this cmd, so you can view its contents.
E7F(CR) will display the EPROM-DATA contents, in HEX/ASCII, of your pic from 00 to 7FH.
Note that the start address is always 00H, you only specify the end address nnn.
If ‘DBEE EQU 0’, the target will ignore the command and display the DATA address only, reducing the bootstrap size on pics without EEPROM DATA AREA or when it is not used.

D sss.fff ..sss.fff (CR) (ON-BREAK MODE ONLY)
Data Dump Display: DATA/SFR RAM address space is usually dispersed over several banks with different sizes and gaps between address. It would be useful to display different DATA RAM address ranges using the same command and store this ranges so they can be displayed again without entering them each time, over and over, on every breakpoint.
D20.7F 150.166 120.135 A0.EF(CR) will display the DATA contents, in HEX/ASCII, of the target pic from 20H to 7FH, from 150H to 166H, from 120H to 135H and from A0H to EFH.
Up to four (4) address ranges can be entered and stored in the command buffer.
So, when you press D, it will show the last address ranges used in case you want to display them again, or just type the new address ranges to be displayed.
Note that the ASCII char of the data will be shown for values 20H-FFH, a '.' for < 20H.

R nnn.nnn...nnn (SP/CR) (ON-BREAK MODE ONLY)
Read DATA/PORT: Sometimes the hardware connected to your target pic doesn't work as you expected. And the need arises to read at your will while you are forcing the input to do something, like a: keyboard, encoder, A/D input potentiometer, TIMER running, etc.
On a 16F628: R5.6.1A(CR) will read the PORTA, PORTB and RCREG contents.
R6.F.E(SP) will read the PORTB, TIMER1H-L each time you press SPACE, and if you hold the SPACE key down, the data are read continuously showing the data changing dynamically.
Up to 8 data/ports address, separated by dots, may be read in the same command.

W nnn dd (+/-/SP/CR) (ON-BREAK MODE ONLY)
Write DATA/PORT: Sometimes the hardware connected to your target pic doesn't work as you expected. And the need arises to write any data at your will while you are testing the output hardware to do something, like a: LCD Display, Port expander, Stepper motor, etc.
On a 18F4620: WF82 7A(CR) will write data 7A to PORTC.
WFAD 41(SP)42(SP)43(SP)44(CR) will write 'A' 'B' 'C' 'D' to the UART-TXREG.
WF80 45(+)AA(+)(+)CF(-)73(SP)55(CR) will write 45H-AAH-CFH-73H-55H to Port A-B-D-C-C.
The (+/-) keys are used as a way to inc/dec the address pointed by the Write command.

F sss{.fff} dd dd dd (CR) (ON-BREAK MODE ONLY)
Fill DATA RAM: During a debugging session you may need to fill some RAM address range with known data in order to verify some data process in your code.
F20.6F 00(CR) will write data 00 from 20H to 6FH.
F20.11F 12 55 43(CR) will write data 12H-55H-43H-12H-55H-43H-... from 20H to 11FH.
F20 CF 23 10 356 89FE 44(CR) will write data CFH-23H-10H-56H-FEH-44H from 20H.
Up to 16 data bytes can be used in the same cmd.
Be careful specifying the address range to do not overlap any SFR unintentionally.

B {W/S dd}(CR) (ON-BREAK MODE ONLY)
Break command: The breakpoints are only placed by inserting their macros while editing your source code and then flashed to the target pic. But when the running code reaches to a breakpoint and its query will set the INSIDER ON-BREAK mode, it will show you BREAK nn, W and Status contents with the pic osc-freq and if (AUTO ON) then S(CR) will be executed.
During a debugging session, B(CR) will show the same again.
BW 7A(CR) will write 7AH to WREG. BS 03(CR) will write 03H to STATUS REG.
This edited W and STATUS values will be set on the target pic when rerun by G(CR).
This cmd allows you to edit W and STATUS without knowing their HEX address on any pic.

****

Note: On any command the bootstrap target routine will not allow you to read/write over the DB0-DB5 INSIDER bootstrap variables where the users: WREG, STATUS, INTCON, FSR (FSR0H, FSR0L on 18Fxxxx) are preserved.
Also, whenever you point to these mentioned SFRs to edit them, the INSIDER bootstrap will address their respective DB0-DB5 TEMPORAL-REGS to write or read from; this way the correct user's SFRs is preserved or edited and then restored to the target pic on rerun.

The INSIDER bootstrap routine doesn't have any means to determine if any target address sent by any user-command, ON-BREAK mode, exists. The user is responsible of what is addressed on any command issued from the INSIDER and the interpretation of the data response or effect from the target pic-hardware.

So, you can do whatever you want, if you know what you are doing.

NOTES TIPS & HINTS:

- In order to avoid any interruption of the Target-INSIDER link during a debugging session, you must disable the Watch Dog Timer (WDT=OFF) in case your application use it.

- Your code may use any interrupt during its execution, but whenever a breakpoint is reached, the target INTCON value will be saved and INTCON,GIE bit will be cleared in order to avoid any interruption of the Target-INSIDER link during the ON-BREAK mode.
You can view/edit the saved INTCON value at your will; it will only be active after the rerun command (G cmd) when W, STATUS, FSR and INTCON are restored to the pic-target.
Note that if you have any timer interrupting your code, this timer will still be running during the INSIDER's ON-BREAK mode (unless you stop it by editing the TxCON SFR) and it will not interrupt the target-pic because INTCON,GIE bit is 0. The same applies to any peripheral's interrupt enabled; they will be functioning but not interrupting.
You can STOP/START any peripheral before/after any selected BREAK nn to avoid losing any code sync, overrun the UART, false ADC readings, PWM running wild, etc.

- Working with low freq-pics (<4MHz) will slow down the Target-INSIDER link speed, so if you want a better response time of the INSIDER you can change the Speed multiplier value defined in the Header of each bootstrap routine by increasing the DBN EQU 1 (default) to 2, 4, 8 and 16 to increase the link speed by a factor of 2x, 4x, 8x, 16x respectively.
Note that these are power of 2 values, 3, 5, 6, 10, 12, etc will not work.

- You can place (ORG) the INSIDER bootstrap routine in your code at your will; I use to place it at the end of the last Program-Memory-Page to clearly separate it from the code to be debugged in the Program-Memory-Usage-Map displayed in the Assembler list file. Also, by fixing its position at any Prog-Memory page, there will be no page boundaries crossing errors with the INSIDER bootstrap routine when linking-assembling your code.
Note that on each bootstrap routine there is an ORG ENDPROG-nnn directive where nnn is the bootstrap length and ENDPROG is the User-Edited end-code-memory of the target pic. You have to be careful to do not overlap any end of program OSCCAL/CONFIG Byte usually found on many pics. So, check your pic datasheet before setting ENDPROG value.
Please don't change the nnn value because it only defines the bootstrap length.

- INSIDERs bootstraps routines define some data-ram variables for its internal use.
On pics 12F6xx and 16Fxxx DB0-DB5 must reside in Bank-0 data RAM and DB0-DB1 must be at Bank-0 common-access area, so they can be reached from any bank during the break-query.
On pics 18Fxxxx DB0-DB6 may be placed as a group on any data-RAM-Bank.
DB0-DB6 must be set as global variables when used to debug any C-compiled pic-code.

- INSIDER MACRO clears STATUS and W contents on 12F6DBUG and 16FDBUG bootstraps.
BREAK MACRO will preserve the user-code W, STATUS, INTCON and FSR on all bootstraps.

- Pic frequency shown at every breakpoint query has the following tolerances:
32MHz +/- 2.4% : 16MHz +/- 1.2% : 8MHz +/- 0.6% : 4MHz +/- 0.3% : 1MHz +/- 0.07%

- To debug 3.3V pic-hardware you must change the INSIDER's (LM2936Z-5 or 78L05) to a 3.3V regulator like LM2936Z-3.3 and resistor R8 from 750 to 430 ohms.

- INSIDER 1Bit-Interface-Pin must be defined by the user in the bootstrap header, you have to select the Port-Bit-number and Port-Tris Reg. Note that only I/O pins will work.
The INSIDER bootstrap will set as input the selected I/O-pin 1Bit-Interface after every breakpoint query; take it into account if you share this pin with your pic-hardware.

- In case you have to share the I/O 1Bit INSIDER connection, consider to share it with any low importance I/O hardware like any led output, pushbutton input, etc. Where the '1' and '0' level will not be affected so much by the operation of the connected hardware.
The INSIDER is able to differentiate the hardware activity from its data link, but you may notice activity in your hardware due to the INSIDER data link transfer.
If there is not any "low" importance I/O-hardware or any spare I/O pin to connect the INSIDER. You will have to consider edit any part of your code that would allow you to free any I/O pin for the INSIDER 1Bit Data link.

- There is no bootstrap for 12-Bit-opcodes 12F5xx pics (like: 12F509, 12F519, etc)
You can use a similar 14-Bit-opcode 12F6xx to run your (modified) 12F5xx target code.

CIRCUIT & FILES:

```
In the INSIDER zip files, there are three folders:
MAIN:        Contains the INSIDER_V1_628.HEX file to be flashed on the INSIDER cpu
             And the INSIDER_User_manual.pdf acrobat reader file.
BOOTSTRAP:   Contains the bootstrap routines for each pic family:
             12F6DBUG.asm – 16FDBUG.asm – 18FDBUG.asm. And three demo sample code.
EAGLE:       Contains all the Schematics and PCB files made on Eagle PCB cad, it can be
             Free downloaded from www.cadsoftusa.com so you can edit the INSIDER circuit.
```

The INSIDER PCB is a 1.9” x 1.2” double layer board to be placed-glued over a 9V-battery holder (DIGIKEY 1294K-ND) with its positive and negative legs soldered to J2 and J3 connection pads on the board at each side of the DB9-female connector
The DB9-female connector is the comm-port to the PC with its wire-solder pins placed over the edge of the board between the jumpers J2 and J3.

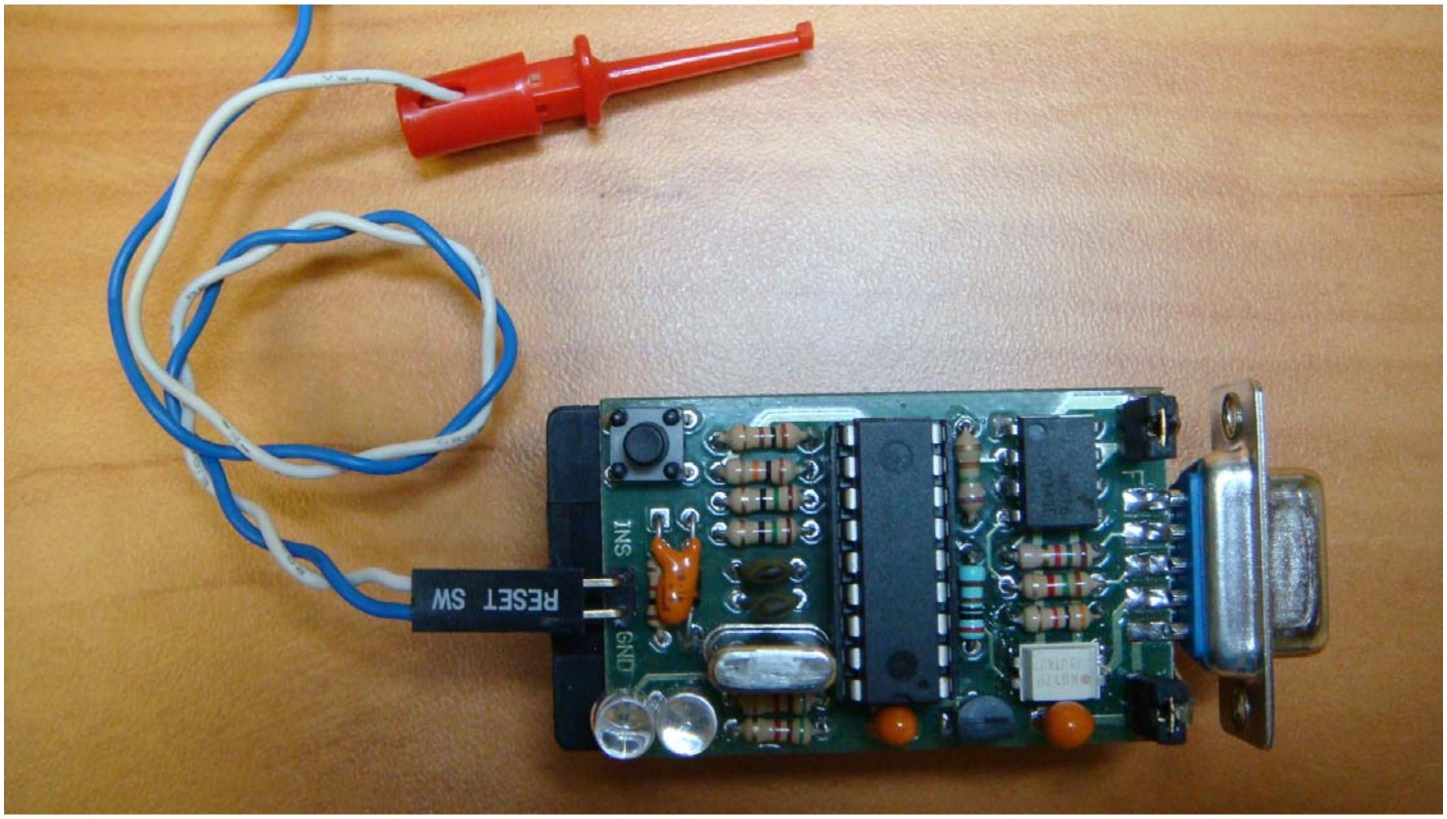
The Dual optocoupler MCT6 (OC1) works as the RS232 interface to the PC-comm-port and provides an isolated high speed path to TXD and RXD PC signals biased from the PC-DTR and GND pin. You can use other dual optocoupler as well but you have to test it and adjust R8-9-10-11 resistors values in order to make it work at 38400BPS.
The single optocoupler OC2 will be needed by a future INSIDER IDE PC program that uses it to reset the INSIDER with the PC-RTS signal. By now OC2 and R12 are not used.

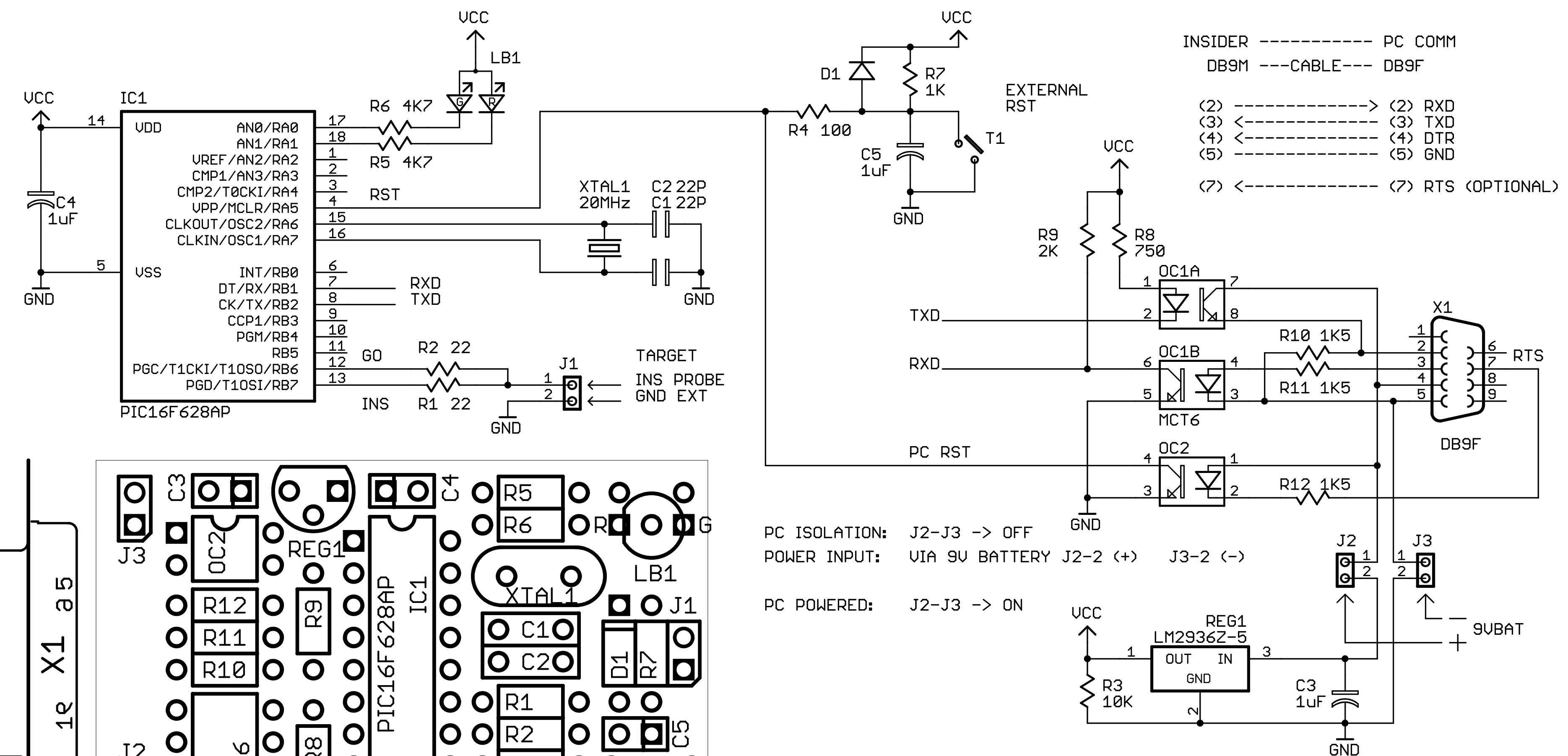
The INSIDER’s voltage regulator (REG1) is a LM2936Z-5(TO-92) 5volt low drop-out, you can use a 78L05. If you plan to work on 3.3v pic, replace it with a LM2936Z-3.3 and R8=430.

J1 is a 2-pin right angle pin header for the 2-wire hook-test-probes used to connect the INSIDER J1 INS-GND to the target pic selected I/O port-pin and GND.

The INSIDER’s cpu is a 16F628 18-pin DIP with an external 20MHz parallel cut crystal.
The Bicolor led LB1 can be replaced by two leds GREEN/RED with their anodes tied to VCC.
The pushbutton T1 (DIGIKEY EG4369-ND) is used to reset the INSIDER.

The INSIDER to the PC-comm-port cable is made by connecting DB9-female 2-3-4-5 pins to a DB9-male 2-3-4-5 pins wire connectors respectively.





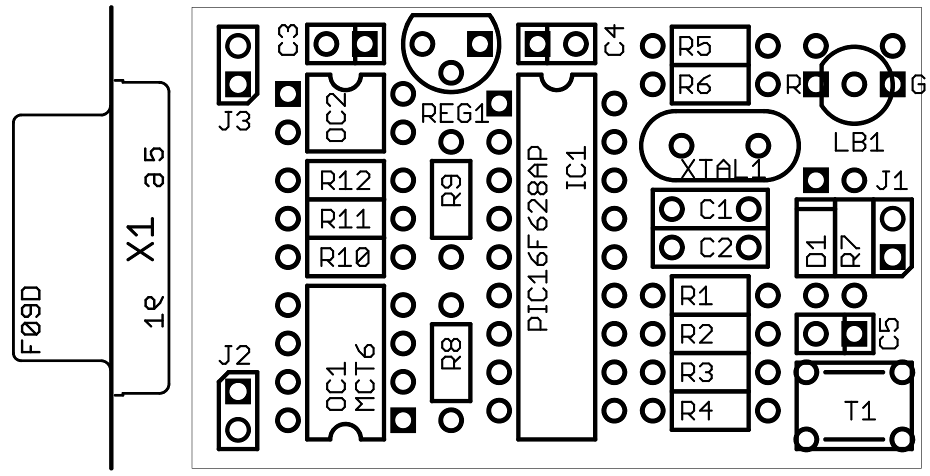
INSIDER ----------- PC COMM
DB9M ---CABLE--- DB9F

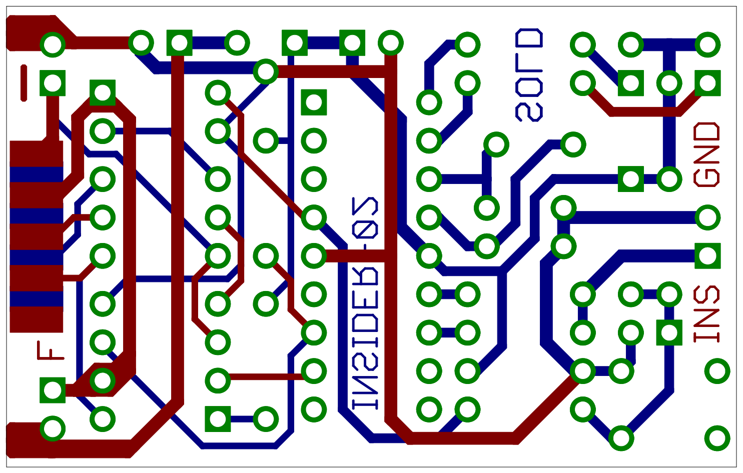
```
(2) -------------> (2) RXD
(3) <------------- (3) TXD
(4) <------------- (4) DTR
(5) -------------- (5) GND

(7) <------------- (7) RTS (OPTIONAL)
```

```
PC ISOLATION:   J2-J3 -> OFF
POWER INPUT:    VIA 9V BATTERY J2-2 (+)    J3-2 (-)

PC POWERED:     J2-J3 -> ON
```





R1-2:	22 1/4W 5%
R3:	10K 1/4W 5%
R4:	100 1/4W 5%
R5-6:	4K7 1/4W 5% (DEPENDS ON LED INTENSITY)
R7:	1K 1/4W 5%
R8:	750 1/4W 5%
R9:	2K 1/4W 5%
R10-11-12:	1K5 1/4W 5%
C1-2:	22pF CERAMIC
C3-4-5:	10uF/16V TANTALUM
X1:	FEMALE DB9 CONNECTOR
REG1:	LM2936Z-5 (TO-92) OR 78L05
OC1:	MCT6 DUAL OPTOCOUPLER (DIP8)
OC2:	SINGLE OPTOCOUPLER (DIP4) OPTIONAL
IC1:	16F628(A) DIP 18 MPU
XTAL1:	XTAL 20MHz
J1-2-3:	2 PIN JUMPER PIN HEADER
1X	9-VOLT BATTERY HOLDER
T1:	PCB N.O PUSH BUTTON
LB1:	COMMON ANODE GREEN/RED BICOLOR LED OR 2 SINGLE LEDS

pic.insider@gmail.com		INGENIERO: ATANASIOS MELIMOPOULOS	
PROYECTO: PIC-INSIDER			
ARCHIVO: INSIDER02	FECHA: 10/09/2008 09:41:15	REV:	PAG: 1/1