6.3 Background Debugger Control

This section describes the hardware requirements that background debugging imposes on the PIC16F877 design as well as the functional description of the background debugger system.

6.3.1 Introduction

Traditionally, embedded system engineers use in circuit emulators to debug system hardware and software. While these emulators work well for many embedded systems, they pose a number of problems that limit their effectiveness:

- High cost of purchasing emulators ($800.00 to $3000+) per system prevents many customers from buying and using them
- Limited high frequency support (10 Mhz limit on PIC16CXX)
- Limited voltage range (+5 Volt system only)
- Serious interconnect problems with surface mount packages
- Inability to diagnose production board problems
- Inability to easily fine tune software for initial prototype systems

Background debugger support solves these limitations of in circuit emulators.

Background debuggers require breakpoint address, peripheral freeze, halt logic and a non-maskable debugger invocation address. A software or hardware communication protocol is established between the Target processor (which runs the background debugger) and the debugger interface that connects to the PC running MPLAB.

A software synchronous protocol can be implemented to communicate between the 16F877 using the ICSP pins (RB6/RB7) and the PC running MPLAB through a debugger interface module. This debugger interface module can also perform In Circuit Serial Programming for automatic download of updated software. No additional communication hardware is needed. However, a background debugger cannot completely be implemented in software - it needs a small amount of hardware assistance with interrupts, peripheral freezing and breakpoints.

The design, build and market the debugger/programmer interface module to MPLAB will be done through a Buy/Re-sell agreement with 3rd party developers; much like ICEPIC. There will also be a 3rd party DLL development.

6.3.2 Enabling Background debugger

To enable the background debugger, there will be an extra configuration bit in the configuration word. The EEPROM configuration bit is used to enable or disable the debugger.

When BKBUG is programmed to “0”, the breakpoint address register and RB6 HIGH->LOW debugger entry are enabled. The single step mode is also enabled. RB6/7 user I/O functionality and interrupt on change functionality is disabled when this bit is set. TRISB<6:7> are forced high, so that RB<6:7> are in input mode.

While in the debug mode, the debug software can access either RB<5:0> of the PORTB pins or just RB<7:6> by switching between banks 0:1 and 2:3. Addresses 006h and 086h will access bits<5:0> of the PORTB and TRISB respectively. Addresses 106h and 186h will only access bits <6:7> of PORTB and TRISB respectively. While running debugger communication, the debugger software should only access addresses 106h and 186h.

If BKBUG is programmed to “0” and the PC equals BKA<12:0> (or the RB6 Halt condition occurs) and the INBUG flag is clear, the background debugger is entered. The INBUG flag is set, the PC is pushed onto the stack and stored in the BKA register, and the PC is set to 1F00h.

Figure 6-3: CONFIGURATION WORD

CP1	CP0	BKBUG	-	WRT	CPD	LVP	BODEN	CP1	CP0	$\overline{\text{PWRTE}}$	WDTE	F0SC1	F0SC0
bit13													bit0

Register: CONFIG
Address 2007h

bit 11: **BKBUG**: Background Debugger Mode (This bit documented as reserved in data sheet)
`1` = Background debugger functions not enabled
`0` = Background debugger functional.

6.3.3 Loading and Reading Program Memory

The ICSP connector establishes the communication channel between the target device and the MPLAB software. The ICSP is also used to load and read the program memory through normal device programming operations.

6.3.4 Reset and Running Programs

In the debug environment, the target device reset pin can be activated by the ICSP connector or by the target system resources. After reset, the target device will immediately begin running the target program. The user must initiate a emulator halt to begin debugger execution.

6.3.5 Halting Execution by I/O Pin

To implement the MPLAB HALT feature, the debugger must be entered when a high to low transition occurs on the RB6 pin and the BKBUG configuration bit is programmed to “0”. A edge detection circuit will generate a HALT signal pulse when RB6 transitions from high to low.

The HALT signal will begin the HALT sequence. The HALT signal must be sampled with Q4 to bound it to a machine cycle.

Processor will go through what appears to be a normal interrupt cycle. However, the normal interrupt logic in the core cannot be used because the debugger would lose the state of the GIE bit. This “debugger interrupt” must not disturb the contents of the GIE bit nor fail to respond if the GIE bit is “0”. The GIE bit must be available for restoration after running debugger code. So, background debugger logic will control sequencing of the key interrupt signals in the core logic. The correctly timed HALT signal will be OR’d with the NINTAKE signal to initiate the sequence. When INBUG bit is set, this will need to block other signals from setting NINTAKE, which will disable interrupts. INBUG also prevents other HALTs from occurring. The INTAK and INTAKD signals will control the vectoring. At the same time, the background debugger logic external to the core will begin counting cycles and sequencing debugger related events. The INBUG bit will be set. Freeze to the peripherals will be asserted, if the FREEZ bit is set. The time that the freeze signal is asserted varies depending on if the instruction finishing is a 1 or 2 cycle instruction. When the interrupt vector is parallel loaded into the PC, the background debugger logic will cause the MSB PC<13> to be set. This will make the logic vector to 2004h instead of 0004h.

During this time, the BKA<12:0> register is updated with the contents of the PC that was also loaded onto the stack. This allows the background debugger to correctly report the current (next to be executed) PC address. Interrupts have to be automatically disabled when the INBUG flag is set. Again, this must be independent of the GIE bit.

As the processor vectors to 2004h, the CPU will fetch and execute a “GOTO Bkg_Debug” instruction. The background debugger logic will send a signal to the CPU core that will cause the output of the PCLATH to assume the value xxx11xxx. This ensures that the GOTO instruction vectors to the highest page of memory without disturbing the contents of the PCLATH.

So instead of taking the first instruction of the interrupt handler a GOTO 1F00h will execute from 2004h. The background debugger routine will commence. PCLATH should be undisturbed, and the PC at breakpoint is at the top of the stack.

The debugger software will use a RETURN instruction to return to the mainline code. The RETURN will cause the INBUG bit to be cleared and will also release the FREEZE signal at the proper time.

The estimate is that the debugger code will take between 256 and 512 program memory words on the target system. The vector for the start of debugger code is programmable by placing the correct instruction in location 2004h. Normally it will be a “GOTO 1F00h”.

Figure 6-4: Background Debugger Execution Sequence

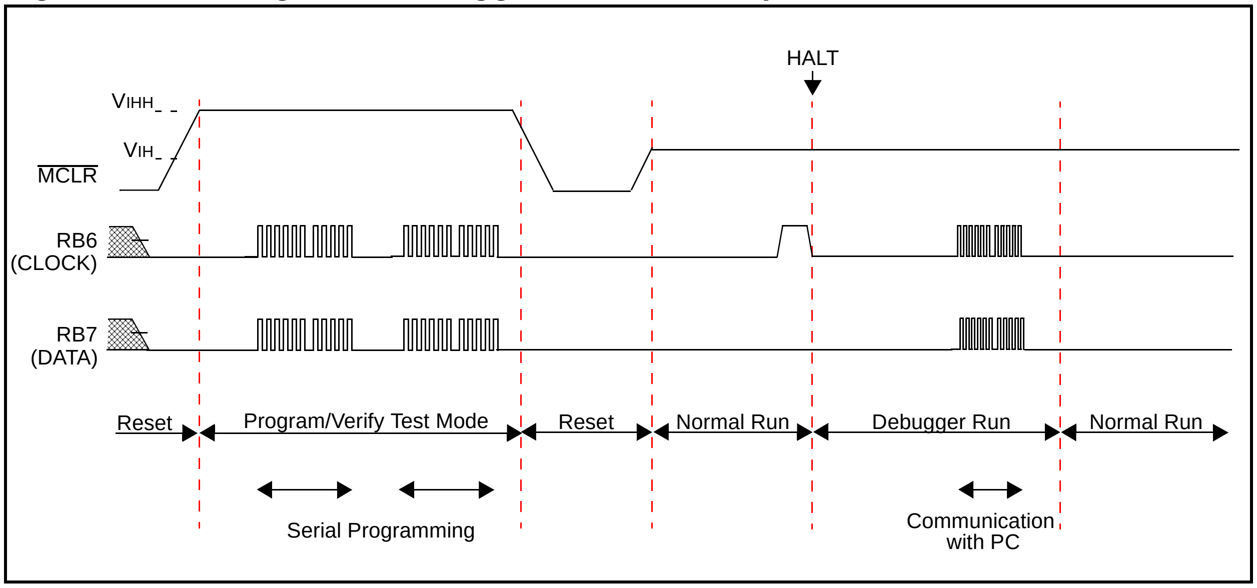


Figure 6-5: External HALT Flow

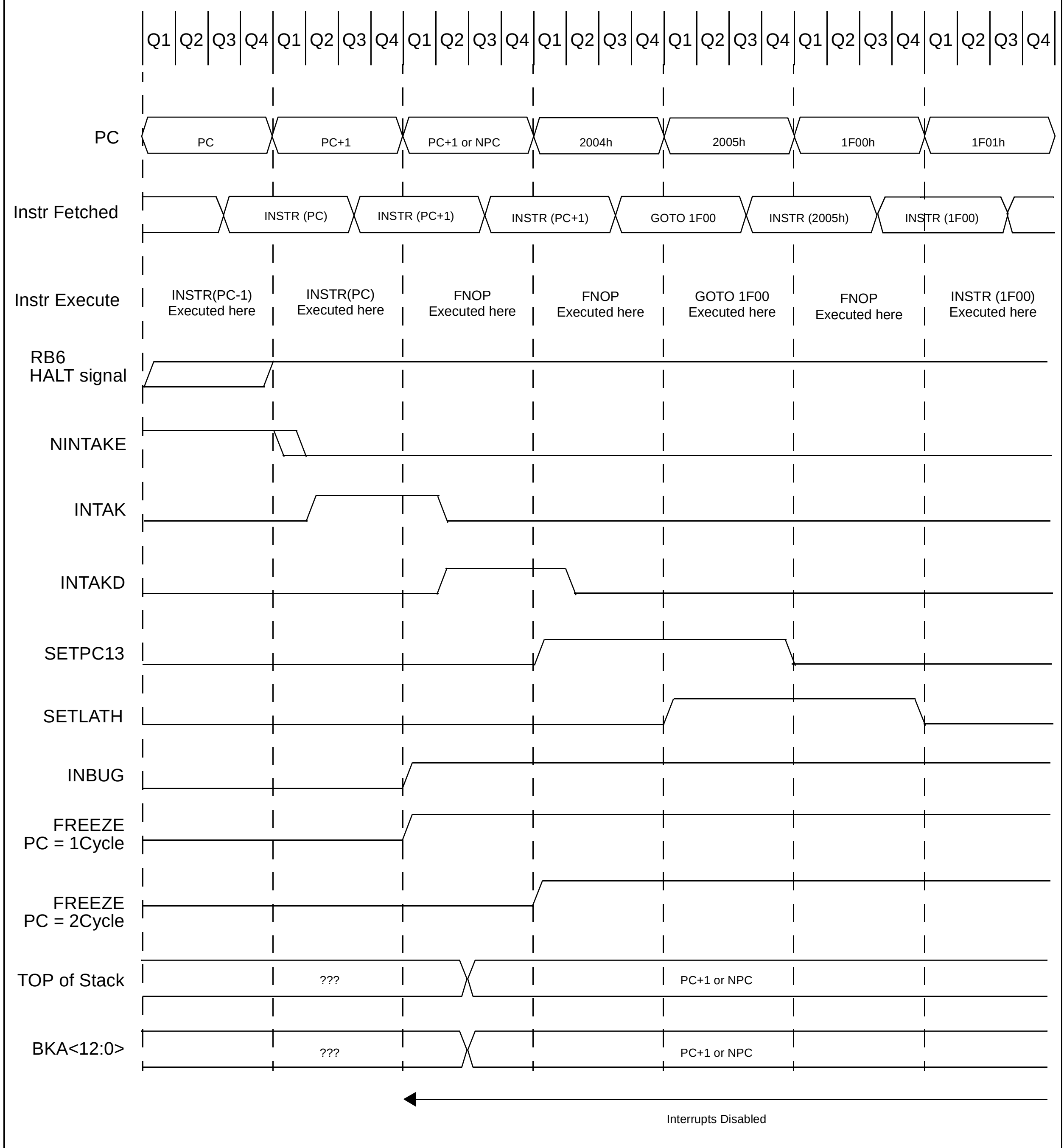
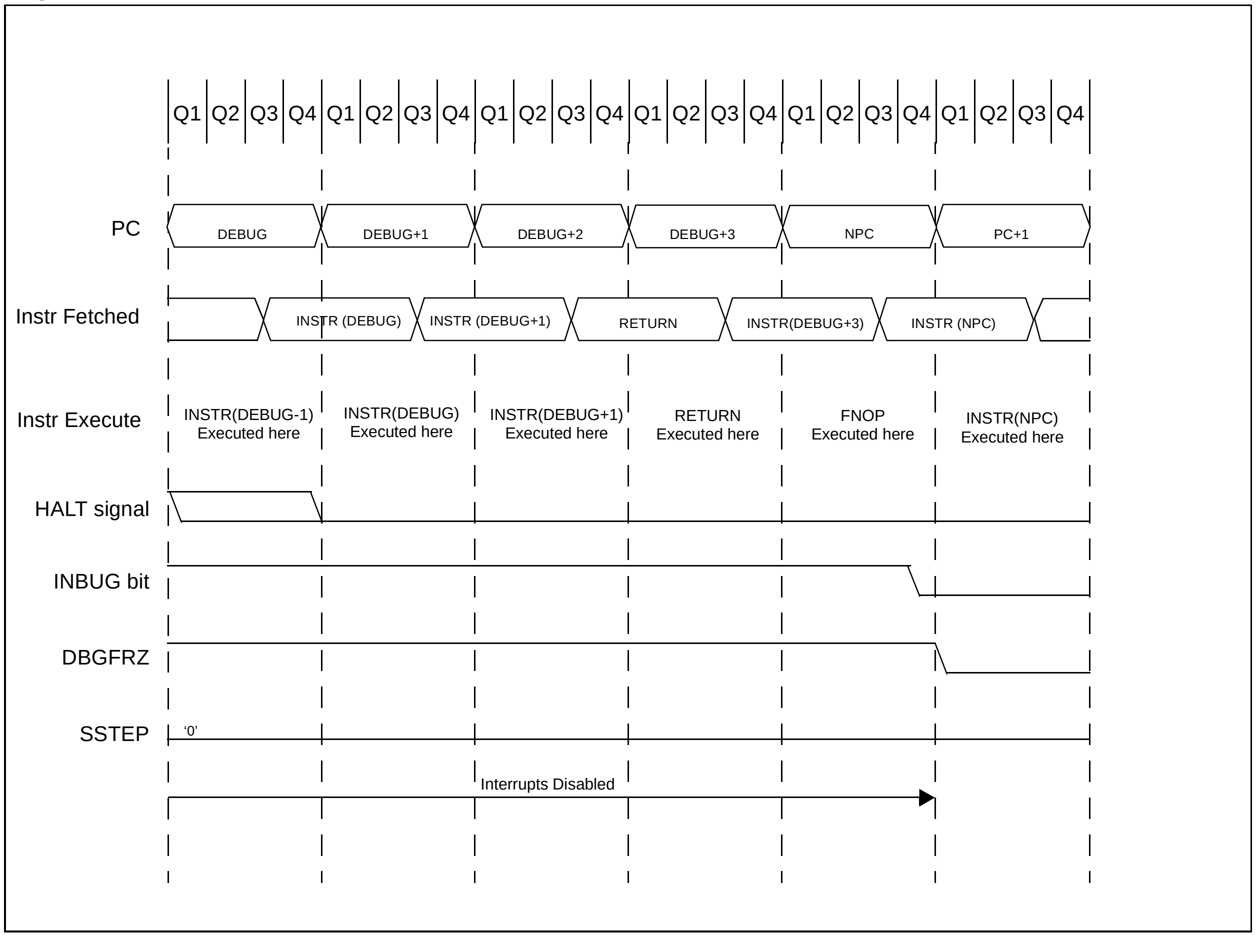


Figure 6-6: Return Flow



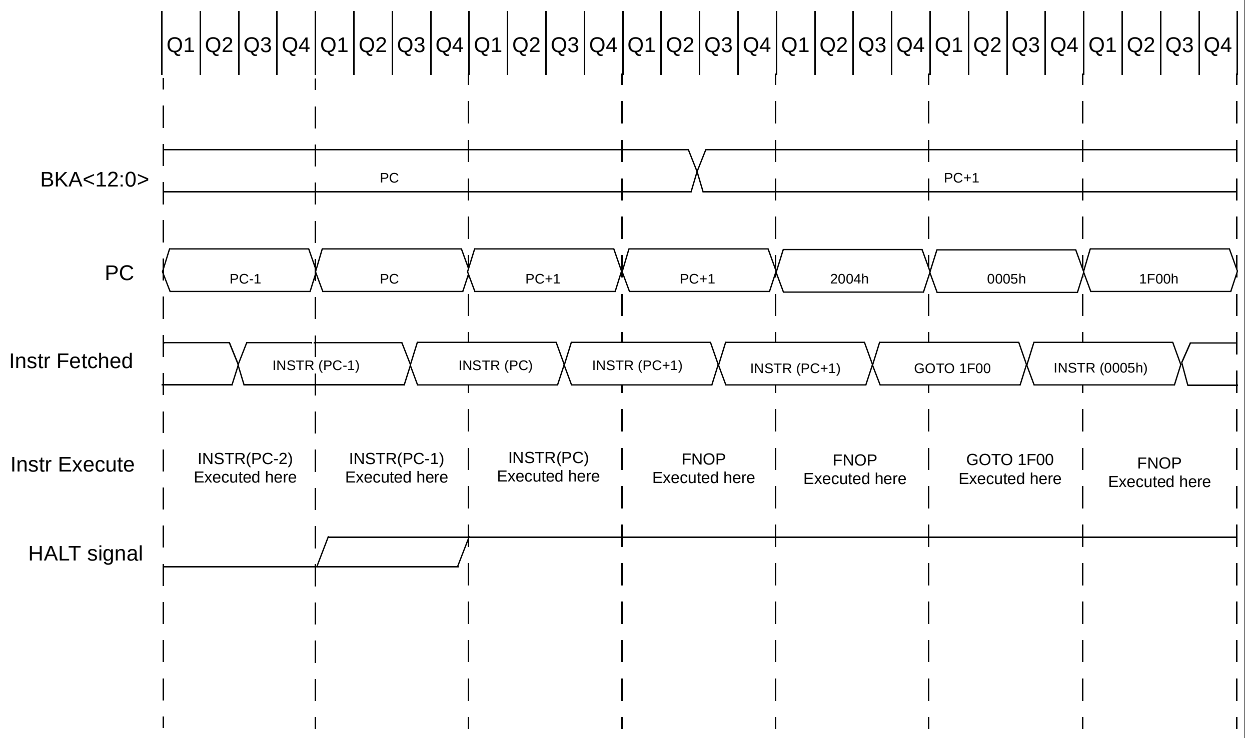
6.3.6 Halting Execution by Breakpoint

Another HALT method is by breakpoint. As with any halt, BKBUG configuration bit must be programmed to “0”. The ICKBUG and BIGBUG registers contain a 13-bit value that is compared against the current PC value. When the values are equal and INBUG is not set, then the circuit will generate a halt signal on that cycle. The HALT will commence as with the external halt cycle.

Once the background debugger is entered, the BKA<12:0> bits in the ICKBUG and BIGBUG registers are loaded with the contents of the PC that was also loaded onto the stack. This allows proper reporting of the current state of the PC (next instruction to be executed) when the background debugger is entered. The power up and MCLR initialization state of these registers will be 0x0000, equal to the reset vector.

Disabling the breakpoints is implemented by setting breakpoint address that lies in the address space of the background debugger (like 0x1F00).

Figure 6-7: Breakpoint Flow

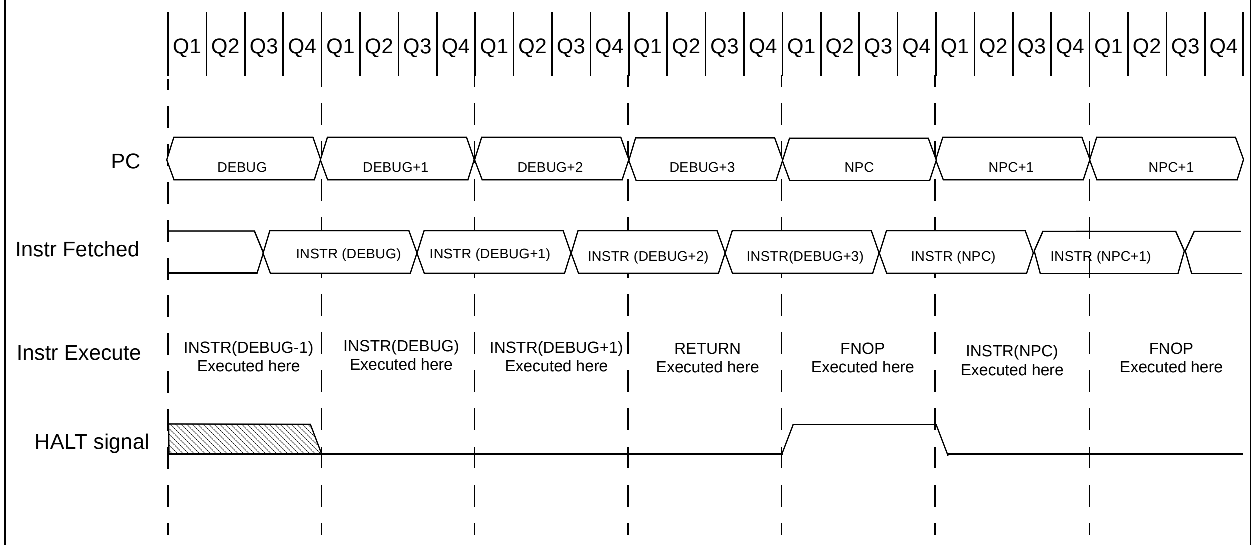


6.3.7 Single Stepping

Another HALT method is by single step. As with any halt, BKBUG configuration bit must be programmed to “0”. When the SSTEP bit is set, and the INBUG bit is cleared signaling an exit of the debugger routine, the background debugger logic will generate a HALT signal timed to allow one instruction execution. The HALT will occur in the FNOP cycle of the RETURN instruction. One user instruction will be executed, then the device will re-enter the background debugger routine in the same fashion as the external halt.

To first get to a single step mode, the debugger must enter normal run mode from reset, then enter the debugger with an external halt. The SSTEP bit can be set and then subsequently, any RETURN from the debugger will be a single step execution.

Figure 6-8: Single Step Flow



6.3.8 Peripheral Freeze considerations

The background debugger logic will output a signal that will be OR'd into the FRZMOD signal. When NEMUL signal is low, the freeze signal from the background debugger logic must be disabled. Similarly, the FRZIN pin signal is disabled from FRZMOD when NEMUL is high. The FREEZ bit in the ICKBUG register determines if peripherals will freeze or not when the background debugger code is entered.

The existing functionality of the freeze in the -ME will be used without change. It is not necessary to improve the -ME hardware freeze signal functionality.

Port B does not have a freeze function because when used as a -ME device, the Port B is on the master.

Since the debugger will issue read modify instructions (BSF, BCF) on PORTB for communication with the debugger interface module, addresses 106h and 186h should be used to modify PORTB<6:7> and TRISB<6:7> so that bits <0:5> are not corrupted. If the user requests a read or a write of PORTB or TRISB, addresses 006h and 086h can be used to access the RB<5:0> portions of the register.

So when the BKBUG fuse =0 and the INBUG bit is set, the debugger code can use 106h and 186h to access port B for communication functions, and use 006h and 086h to access the remaining part of port B.

6.3.9 Hardware support for Background Debugger

The background debugger logic will consist of:

- Extra configuration bit BKBUG and logic to disable RB6,RB7 user pin functions.
- Circuit to generate a Q4 sampled HALT signal from falling RB6 pin.
- ICKBUG and BIGBUG registers. These registers will power-up/reset to 0x00.
- Comparator logic for PC and BKA<12:0>. Latch for PC value to be read in BKA<12:0> after HALT.
- Logic to generate HALT sequence in CPU core.
- Logic to time and generate freeze and FRZMOD.
- Logic to time and generate INBUG bit and disable other interrupts and halts.
- Logic to re-create the PortB functions for the debugger.

In addition to the background debugger logic required to support the background debugger as described in the above sections, there are several issues to address.

- The program memory decode must wrap the program memory on less than 8K devices to allow 0x1F00 as the debugger address for all metal options.
- Background debugger functions must be disabled when device is in -ME mode.
- To prevent a deadlock condition, the background debugger is never entered if the INBUG flag is set.
- Watchdog timer will still be functional, there are no special modes associated with background debugger.
- The reset, POR and BOR logic will still be functional with no special modes associated with the background debugger.

The definitions of the ICKBUG and BIGBUG registers are in Figure 6-10 and Figure 6-10. There are 3 control bits and 13 bits for breakpoint address.

INBUG - Set automatically when entering the debugger routine. Must be cleared by the debugger just before exit. A 4 instruction cycle delay will prevent the new state of zero to be recognized by the core to allow a safe debugger exit. When set, interrupts are disabled (regardless of the state of the GIE bit) and the debugger HALT is disabled. This prevents reentry into the debugger while communicating with MPLAB.

FREEZ - MPLAB will instruct the debugger to set or clear this bit. When FREEZ is set, TMR0, TMR1 and TMR2 and their prescalers freeze (suspend counting). The SSP, A/D converter and USART state machines will also suspend upon the assertion of the FREEZ bit. Status flag bits that are altered by register read operations are frozen in their current state while FREEZ is asserted. The debugger logic will compensate for all instruction cycles necessary for the debugger entry and RETURN so that the timers and their respective prescalers read the same value as they would if the breakpoint (and debugger code execution) did not occur. It is acceptable to limit the freeze functionality to that which is offered on PICMASTER/ MPLAB-ICE by the -ME chip freeze signal.

SSTEP - Enables single step operation. When SSTEP is set, after INBUG is cleared, the logic will allow only one user instruction to execute, then the background debugger is re-entered.

BKA - Break address, these bits are cleared at POR. No other reset clears the bits. I.E. a MCLR cannot change the breakpoint address value.

Figure 6-9: ICKBUG Register (Address 18Eh)

R-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
INBUG	FREEZ	SSTEP	BKA12	BKA11	BKA10	BKA09	BKA08
bit7							bit0

R	= Readable bit
W	= Writable bit
S	= Settable bit
U	= Unimplemented bit, read as ‘0’
- n	= Value at POR reset

bit 7: **INBUG**: Background debugger activity status
1 = Device is executing background debugger code
0 = Device is executing user code
(This bit is read only; set from background halt or breakpoint occurrence; only clear by RETURN)

bit 6: **FREEZ**: Background debugger freeze mode
1 = Peripherals will freeze when INBUG=1
0 = Peripherals will not freeze when INBUG=1

bit 5: **SSTEP**: Single Step Enable
1 = Program will execute 1 instruction word of user code upon RETURN from debug code
0 = Program will execute multiple instruction words of user code

bit 4-0: **BKA**: Breakpoint Address
When writing, represents the requested breakpoint address.
When reading, represents the value of the PC at last breakpoint, halt, or single step operation.

Figure 6-10: BIGBUG Register (Address 18Fh)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
BKA07	BKA06	BKA05	BKA04	BKA03	BKA02	BKA01	BKA00
bit7							bit0

R	= Readable bit
W	= Writable bit
S	= Settable bit
U	= Unimplemented bit, read as ‘0’
- n	= Value at POR reset

bit 7-0: **BKA**: Breakpoint Address
When writing, represents the requested breakpoint address.
When reading, represents the value of the PC at last breakpoint, halt, or single step operation.

6.3.10 Background Debugger Software

The following MPLAB API commands will be implemented in the background debugger:

- DbgInterrogate - Read all internal file address locations and report them to MPLAB
- DbgSelectBreakMemory - Set breakpoint memory address
- DbgReset - Re-loads DLL, apply reset to PIC, reset PC, Stack
- DbgIssueMCLR - Apply MCLR assertion
- DbgHack - Check to see if processor is halted
- DbgForce - Execute a PIC16/17 opcode
- DbgRun - Run main code (exit background debugger)
- DbgStep - Single Step one instruction
- DbgHalt - Force a processor halt
- DbgGetPC - Return current PC address (from BKA register) that it to be executed next after the debugger returns.
- DbgPutFile - Force a new value into a file register address
- DbgGetFile - Read value of file register at specified address.
- dbgWriteMemory - Download program memory
- dbgReadMemory - Read program memory

All debugger software must not write PCL until the status of PCLATH is saved.

DbgGetFile needs to report the save_context values for W, STATUS, FSR and PCLATH.

The watchdog timer will still be functional, so it is necessary that the debugger software reset the watchdog.

Currently, probes are identified by data stored in an EE memory. PICMASTER can write to the probe for configuration and also read this data through some complex patched EE protocol code. The debugger software will need to store a device processor code in program memory and return this when asked what DLL to load. The processor return codes for identification need to be defined.

In PICMASTER, the WDT state is disassociated with the user setting of the configuration bits in the source file. With the background debugger, it would be easiest if the configuration file selected this state automatically when the PIC16F877 is programmed.

6.3.11 Loss of Target System Resources

Using the background debugger causes some loss of functionality in the users target system.

- RB6 and RB7 are lost to the user.
- The user must implement the ICSP logic in the system.
- One level of stack is lost to the debugger entry and exit.
- Very low frequency systems may not work with the debugger due to watchdog timer interrupts during debugger code execution.
- User must allocate program space for the debug monitor.

6.3.12 MPLAB Functions not Supported

Due to the limitations of the background debugger, several MPLAB functions found while executing the emulator system are not present when executing the background debugger.

- There will be no trace file or trace instruction capability.
- There will be no pass counter capability.
- There is no report of the state of the Stack to MPLAB?
- We will be limited to one breakpoint.
- Change Program Counter
- Execute an Opcode
- Stopwatch
- Stack Overflow Break Enable
- dbgPutStack cannot be implemented. There is no way to modify the stack.
- dbgReadTargetMemory and dbgWriteTargetMemory are not implemented. These seem unneeded with FLASH memory downloads through ICSP.