HI-TECH C Compiler for PIC18 MCUs (PRO) Version 9.66 Release Notes

Produced on: April 20, 2011

Australian Design Centre
45 Colebard Street West
Acacia Ridge QLD 4110
Australia

web: http://microchip.htsoft.com

THIS FILE CONTAINS IMPORTANT INFORMATION RELATING TO THIS COMPILER. PLEASE READ IT BEFORE RUNNING THIS SOFTWARE.

Chapter 1

Introduction

1.1 Description

This is a minor update release for version 9.65 of the HI-TECH C PRO for the PIC18™ MCU Family.

1.2 Updates and Feedback

Microchip welcomes bug reports, suggestions or comments regarding this compiler version. Please direct any bug reports or feature requests via email to (support).

1.3 Previous Versions

The previous version of HI-TECH C PRO for PIC18 MCUs was 9.65 released in November 2010.

Chapter 2

New Features

The following are new features the compiler now supports. The version number in brackets indicates the first compiler version to support the feature. See Section 6.1 for a full list of all supported devices.

2.1 General

New Chips Supported (9.66) No new device support is added.

New Chips Supported (9.65) The following devices are now supported: 18F23K22, 18F24K22, 18F25K22, 18F25K80, 18F26J13, 18F26J53, 18F26K22, 18F26K80, 18F27J13, 18F27J53, 18F43K22, 18F44K22, 18F45K22, 18F45K80, 18F46J13, 18F46J53, 18F46K22, 18F46K80, 18F47J13, 18F47J53, 18F65K22, 18F65K80, 18F65K90, 18F66K22, 18F66K80, 18F66K90, 18F67K22, 18F67K90, 18F85K22, 18F85K90, 18F86J72, 18F86K22, 18F86K90, 18F87J72, 18LF23K22, 18LF24K22, 18LF25K22, 18LF25K80, 18LF26J13, 18LF26J53, 18LF26K22, 18LF26K80, 18LF27J13, 18LF27J53, 18LF43K22, 18LF44K22, 18LF45K22, 18LF45K80, 18LF46J13, 18LF46J53, 18LF46K22, 18LF46K80, 18LF47J13, 18LF47J53 and 18LF65K80.

FETCH Errata (9.63PL4) Work-arounds have been added to the compiler for devices which suffer the "Fetch" errata (e.g. PIC18F8585) where instruction fetches can be corrupted. The recommended work-around is to insert a NOP (0xFFFF) after every TBLRD and return type instruction, at the target of every GOTO and CALL and at the interrupt vector addresses.

New Chips Supported (9.63PL3) The following devices are now supported: 18F13K20, 18F13K22, 18F14K20, 18F14K22, 18F24J11, 18F24J50, 18F25J11, 18F25J50, 18F26J50, 18F44J11, 18F44J50, 18F45J11, 18F45J50, 18F46J50. 18F66J90, 18F66J93, 18F67J90, 18J67J93, 18F86J90, 18F86J93, 18F87J90, 18F87J93, 18F87K22, 18F87K90, 18LF13K22, 18LF13K50, 18LF14K22,

18LF14K50, 18LF24J10, 18LF24J11, 18LF24J50, 18LF25J10, 18LF25J11, 18LF25J50, 18LF26J11, 18LF26J50, 18LF44J10, 18LF44J11, 18LF44J50, 18LF45J10, 18LF45J11, 18LF45J50, 18LF46J11 and 18LF46J50

New Chips Supported (9.63PL2) The following devices are now supported: 18F46J11, 18F26J11

Processing Assembly Auto Variables (9.63PL2) The compiler will now process FNSIZE and FNCALL directives contains in assembly code (or precompiled object files) and include this information into the calculation of the stack size determined by the code generator.

Disabling Function Duplication (9.63PL2) The interrupt_level pragma can now be used to prevent functions being duplicated when the user can guarantee that an interrupt will not occur while the function is executing from main-line code.

Bit types in MPLAB (9.63PL2) Previously the 'bit' type was not a recognized type that could be watched in Microchip's MPLAB IDE. The compiler will now manipulate the COFF output format so that bit types are treated like a bit-field within a structure. Since MPLAB has no problem watching structures and their members, this provides a convenient work-around for this limitation.

ICD3 (9.63PL1) The MPLAB ICD3 is now supported. If the ICD3 is used, the option `--debugger=icd3` should be used.

PICKit debugger (9.63PL1) The MPLAB PICKit 2 & PICKit 3 are now supported. If the PICKit is used, the option `--debugger=pickit2` or`--debugger=pickit3` should be used.

REALICE Native Trace (9.63PL1) Preliminary support is available for native trace when using REAL ICE. To do so, you must ensure that the REALICE debugger is selected when compiling under MPLAB or HI-TIDE. If compiling on the command line, used the `--DEBUGGER=realice` option. See also the help files associated with REAL ICE published by Microchip.

New Chip-Specific Macro (9.63) The driver now defines the chip-specific symbols consisting of double underscore followed by the chip name.

e.g. For the `pic18f4580`, the macro *__18F4580* is now defined as well as the pre-existing macro `_18F4580`.

HI-TECH C PRO in Lite mode (9.63) A significant new feature available as of this version is the introduction of Lite mode. This is in addition to the product's existing PRO and 45 day evaluation modes. When operating in Lite mode the compiler will continue to support the complete set of devices available in PRO mode, but all compiler optimizations will now be disabled. No memory restrictions apply in Lite mode.Lite mode can be enabled without activation. Lite

mode will be the compiler's mode of operation when activation is incomplete or evaluation period has expired. Previously, the compiler would have been inoperable in either of these states.The installer allows the compiler to be installed in Lite mode. Doing so allows the one-time 45 day evaluation period to be deferred, however once evaluation has been initiated it will not be prolonged by switching back to Lite mode. Evaluation mode can only be activated once and will expire 45 days after this mode has been initiated.

Sample Code (9.63) Sample code for picdem2 development board and an example bootloader has been supplied with this release of the compiler.

2.2 Command-Line Driver

–runtime=+plib (9.63) This new sub-option (off by default) is used to link in the MPLAB C18 compatible peripheral library.

–opt=space (9.63) This new sub-option (on by default) indicates to the compiler that it should optimize code for space.

–opt=speed (9.63) Intended for future use to indicate the compiler that it should optimize for speed. Use this sub-option in conjunction with other sub-options if you want space specific optimization to be disabled.

__*chipname* predefined macro (9.63) The driver in addition to *_chipname* will create this predefined preprocessor macro for the target device.

–debugger=none (9.63) This new sub-option (selected by default) is used to indicate to the compiler that this build is not intended for use with any hardware debugger.

–debugger=realice (9.63) This new sub-option (off by default) is used to indicate to the compiler that realice debugger is being used. Selecting this option also defines the preprocessor symbol __MPLAB_REALICE=1.

2.3 Code Generator/Parser

Anonymous Members (9.63) The usage of anonymous structures and unions as members of structures and unions is now supported.

2.4 Assembler

FETCH and 4000 Errata (9.63PL4) The assembler now has the ability to insert a NOP (0xFFFF) at the prescribed locations (where possible) according to the work-arounds for the FETCH and 4000 errata. This feature will be automatically enabled when either of these errata are enabled.

Optimize for Space using Procedural Abstraction (9.63) For better and faster code, new level of optimization is being added. If running in Lite mode, this optimization is not available.

2.5 Libraries and Header files

New header files (9.66) Each device now has a separate single header file. The old family specific header files are no more in the distribution.

New header files (9.65) The files `stdint.h` and `stdbool.h` have been added to the distribution.

Peripheral Library (9.65) There is no peripheral library support for the new devices added in version 9.65.

BANKSEL and BANKMASK assembler macro (9.63PL3) Two new assembler macros have been added. The BANKSEL will do a bank selection for the address in parameter and the BANKMASK will mask the bankl of the address.

MPLAB C18 Compatible Peripheral Library (9.63) This release includes a peripheral library which is compatible with MPLAB C18 library.

New Header Files (9.63) A new set of header files for each chip have been added which gives compatibility with Microchip bitfield definitions. These files are automatically included by the file `htc.h`.

Chapter 3

Changes

The following are features that are now handled differently by the compiler. These changes may require modification to your source code if porting code to this compiler version. The version number in brackets indicates the first compiler version to implement the change.

3.1 Parser

Redefining enumerations (9.63PL1) In the past redefining an enumeration type would cause the compiler's parser to emit a warning. The parser will now treat this as an error and emit the error message 1348: enum tag "*" redefined (from *:*).

3.2 Command-Line Driver

`--opt=+space` (9.63PL1) This optimisation sub-option (selected by default) means that all optimising behaviour of the compiler (e.g. the assembly or global optimisers) that has been enabled should favor space over speed.

`--opt=+speed` (9.63PL1) This optimisation sub-option means that all optimising behaviour of the compiler (e.g. the assembly or global optimisers) that has been enabled should favor speed over space.

3.3 Code Generator

Indication of Build Number (9.63) The build number of the compiler is now displayed in code generated output.

Source Level Debugging (9.63) HTC library files, e.g. sin, tan etc, can now be debugged just like implicitly called routines like ftadd, flcmp etc.

3.4 Assembler

Procedural Abstraction (9.63PL3) For devices affected by errata, the procedural abstraction optimizations will be disabled.

Procedural abstraction of inline assembly (9.63PL1) Any inline assembly in a program will no longer be considered by the assembly optimiser as a candidate for procedural abstraction.

3.5 Libraries and Header Files

Program memory read functions (9.63PL4) Work-arounds for the 4000 and FETCH errata were implemented in the compiler library functions config_read, device_id_read, flash_read and idloc_read. These have been removed since the assembler now adds these work-arounds as required.

Checksum functions (9.63PL4) Work-arounds for the 4000 and FETCH errata were implemented in the checksum functions provided by the compiler. These have been removed since the assembler will now add these work-arounds as required.

__PROG_CONFIG macro function (9.63PL4) This macro function has been added to allow users to conveniently program a literal value into a configuration word register. It was introduced to work-around the limitations of the current __CONFIG macro function which changed in version 9.60PL3.

__CONFIG macro function (9.63PL3) The way in which the configuration words are programmed by the compiler has changed. A consequence of this is using this macro function to program a literal value in a configuration word may produce unexpected results. Only the configuration setting macros provided in the device specific header files should be used with this function.

Device Specific Header Files (9.63PL1) All register definitions in device specific header files are now absolute variables declared “extern”.

Flash and EEPROM Libraries (9.63PL1) The naming convention of Flash and EEPROM libraries has changed to reflect the new method of generation. The new naming convention is described in the relevant section of the manual. Some errata that were previously applied are no longer necessary.

idloc_write Function (9.63) This function has been recoded to be slightly more optimal.

Device Specific Header Files (9.63) Each chip now has their own specific header file. This does away with the header file per family scheme. Previous types and definitions have been maintained to ensure compatibility with previous versions of the compiler. Many new types and definitions have been added to these header files to ensure compatibility with the new peripheral libraries. Please refer to the header files for more details.

Chapter 4

Limitations

The following are limitations in the compiler's operation. These may be general coding restrictions, or deviations from information contained in the user's manual.

4.1 General

Code size In some instances the code size may be larger than that produced by the PICC-18 STD compiler.

Auto/param size In some instances the amount of space taken up by the program's auto/parameter area (`paramx` psect) in RAM (this area also holds temporary variables) is larger than that produced by the PICC-18 STD compiler.

Array pointers In some instances, the size of elements of arrays of pointers may be incorrectly calculated.

Absolute addressed const object The absolute addressing of const qualified objects is not supported in this version of the compiler.

Struct/Union with Pointer Members In some instances, taking address of a structure member and assigning to another pointer member may yield incorrect results.

Const Qualified Local Vars Compiler may allocate const qualified local vars in Data Memory instead of Program Memory.

Mixing C And Assembly Using assembly programs (where data is placed at absolute addresses) with C programs, may result in bank selection issues.

4.2 Parser

4.3 Preprocessor

Sizeof Pointers The result of the sizeof preprocessor operator when applied to a pointer will always return 1. Use of the C sizeof operator will return the correct value.

4.4 C Code

Undefined symbol when only accessing first array element If an array is a member of a structure or union and the only array index used throughout the entire program is the constant 0, an undefined symbol error will be issued by the compiler. The error will not be produced if the structure is qualified as `volatile`.

Indirect calls from ISRs Indirect calls made in interrupt service routines to functions that are also called from main-line code may fail. This does not affect direct calls made for an ISR. A warning is now issued if this situation is encountered.

Degenerate comparisons Additional warnings have been added in places where a degenerate comparison can be identified.

Far Variables Variables qualified as `far` may be defined, but will not be assigned initial values. The usual caveats with `far` variables still apply with regard to the size of the code generated to access them.

Extended PIC18 instruction set Presently the extended PIC18 instruction set and indexed addressing mode are not supported by either the HI-TECH C PRO for the PIC18 MCU Family or PICC-18 STD compilers. If using a device which is equipped with the extended instruction set, it can be configured for legacy mode (extended instructions disabled) with the following configuration setting:

```
__CONFIG(4, XINSTDIS);
```

Other configuration word 4 parameters are inherently ANDed into this directive as required.

Global optimizer The global optimizer should not be used with C code that accesses the `TBLPTR`*x*, `FSR`*xx* or `PROD`*x* special function registers. The global optimizer may use these registers for temporary variable placement and currently cannot detect the use of these registers in user code.

Error on initialization of complex bitfields Any structure using bitfields within a structure or union cannot be initialized. For example, using the following types where a structure is a member of a union, initialization of the structure's bitfields will generate an error.

```
typedef struct {
    unsigned b0:1, b1:1, b2:1, b3:1, b4:1, b5:1, b6:1, b7:1
} byte_bits;
typedef union {
    byte_bits    bits;
    char         byte;
} byte_or_bits;
```

This will generate a compiler error:

```
byte_or_bits example = { {1,0,1,1,0,1,0,0 } };
```

Instead, use a single value:

```
byte_or_bits example = { 0b10110100 };
```

Missing parenthesis not detected A missing end parenthesis is not detected in variable declarations when the variable is type cast. For example,

```
static char C @ ((unsigned)&PORTA;
```

This problem does not appear to affect any output code, but may cause problems when using third-party software tools.

Qualified arrays generate error The compiler generates an error when compiling complicated qualified arrays, e.g.:

```
const char * const * const listnames[] = {menu0, menu1};
```

The error is not issued if the array is not qualified, or for arrays of more basic types.

Multiple block variable declaration Functions which have variables declared within multiple blocks where code within the block requires the use of temporary memory, may get corrupted when compiled with global optimizations. A workaround is to move the inner variable declaration outside of the block.

Structure initialization Locally defined structures cannot be initialized.

Returning structures A function cannot return a copy of structure which resides in ROM.

Float to short long cast Type casting a float to short long will result in an error.

Initialised Far Structs Vars Far struct declared with initialized values and accessed through pointers may yield incorrect results.

XOR Operation XORing a long and char type variables and storing the result back to same long type variable may result in incorrect output, eg.:

```
volatile long lvar;
volatile char char;
main() {
lvar = 14; cvar = 7;
lvar ^= cvar;
}
In the above example, lvar will be left with incorrect value.
```

Operation on long long types Subtraction and Logical OR on long long int or unsigned long long int types of variables may result in undefined behaviour of compiler.

Indirect Function call Use of indirect function calls when the function parameters include pointers, may result in compiler error. Try to avoid using pointers as parameters to the functions called indirectly.

Modifying const objects Attempt to modify const qualified objects may give warning along with compiler error. Avoid modifying const objects.

Shift operation on signed type Shifting operation on signed integer types does not extend sign and hence will result in unexpected results, eg.:

```
volatile int a,b;
main() {
b = 0xFFEE;
a = b >> 8;
}
In the above example, a will have value 0x00FF, instead of expected value bei
```

Function pointer as structure member If object of a structure with function pointer as member, is initialized along with the declaration, then accessing this object may result in error.

4.5 Libraries

Flash Write Routine The flash write routine will fail if the memory being written is in the same 400h byte aligned block as the functions associated with writing flash.

Fast Double Libraries The fast form of the floating-point library routines have not been implemented. The sub-option in the –double options has been removed to that effect.

Floating point accuracy The floating point math routines, despite `DBL_MIN` being defined as 1.17549435E-38, cannot perform operations predictably on numbers below the order of 1e-35. The result is either correct OR zero.

4.6 Linker and Psects

Psect Pragma The `#pragma psect` directive, which can be used to redirect output into a psect with a unique name, will not function as expected with HI-TECH C PRO for the PIC18 MCU Family. Although the pragma will perform a redirection, it will do so for the entire program, not just the module in which it is used. An alternative, more user-friendly mechanism is being developed to allow placement of certain objects. Non-`const` global objects can be placed at specific addresses by making them absolute.

Local/auto variables The total size of data that can be used for local/auto variables at any time must be less than the size of a RAM bank. If this limit is exceeded, a *"Can't find ??? words for psect param"* error will result. The *critical path* of function calls that caused the error can be identified in the *call graph* section of a map file. An asterisk denotes a function is on the critical path.

4.7 Microchip Debug Limitations

ICE2000/4000 Users of the Microchip ICE2000/4000 should be aware of a documented limitation of these tools concerning side-effects when reading particular SFRs. Reading the following SFRs can affect a STATUS bit, but due to a limitation of the ICE some general purpose registers will also be affected. Reading from the following SFRs can affect the GPRs as follows:

- PORTB - x81 (081, 181, 281, 391 etc.)
- RCREG1 - xAE
- RCREG2 - x6E
- PORTD - x83
- SSPBUF - xC9

  It is advised to reserve the corresponding locations using a –RAM option (while debugging) if your program reads from the above registers. More information can be found in MPLAB via the ICE2000/4000 help topics under Limitations: PIC18XXXX General Limitations.

COF debug limitation For users of MPLAB debugging tools please be aware that due to a limitation in the COF debugging format, pointers are assumed to be 16-bits in length. This limitation will mean that larger pointers will appear truncated in the variable watch window. The actual contents of the pointer and program execution are not affected.

Chapter 5

Bugfixes

The following are corrections that have been made to the compiler. These may fix bugs in the generated code or alter the operation of the compiler to that which was intended or specified by the user's manual. The version number in brackets indicates the first compiler version to implement the fix.

5.1 Command-Line Driver

Access bank allocation (9.63PL4) If the –RAM option was used and specified memory in the access bank, e.g. –RAM=0-20,50-100, the memory allocated to the access bank linker classes may not have been correctly defined.

Undefined Symbols (9.63PL1) Applications using the flash/eeprom libraries may have produced undefined symbols.

Memory Summary (9.63) Some memory summary results for psects of a unit size greater than one byte were being incorrectly calculated.

5.2 Code Generator/Parser

Wrong bank selection in comparing unsigned long with literal (9.66) Bank was not being selected for temporary variables used in computation.

Right shift of long by zero (9.66) Right shifting of a long variable by zero was incorrectly handled.

Incorrect result for xor_between long and char (9.66) XOR between a long with char was resulting into incorrect results.

Right shift of integers (9.66) Logical shift was being performed instead of arithmetic shift.

Wrong value assigned to union variable (9.66) Assigning value to union variable was incorectly done.

Banksel missing when assigning bitfield from temporary variable (9.66) Banksel was added in assignment to bitfield from a temporary value.

Pointer comparison to NULL (9.65) If a 2-byte pointer was compared to NULL, the comparison may result a true value if the LSB of the pointer's content was 0. A full 2-byte comparison is now performed and this comparison will return false where expected.

Incorrect bit-field initialization (9.65) If a structure with bit-fields was initialized, but the number of initial values was less than the number of bit-field members contained in the structure, the compiler was not padding the initial values with extra 0 values.

"Undefined symbol 'null"' error (9.65) In some instances pointer variables may not have acquired the correct size which would have lead to this error. The size of pointers is now always performed correctly.

"Looping around MACH_ALLOC" error (9.65) This error may have been produced in some situations where memory allocation of variables failed. Better sorting has been implemented.

Wrong bank selection after pointer access (9.65) One of two banks might have been selected after executing code that performed a dereference of a pointer that has targets to both data and program memory. After executing this code, the compiler would have assumed only one bank selection was possible and this may have caused subsequent code to fail.

Wrong bank selection after bit-field assignment (9.65) One of two banks might have been selected after executing code that performed an assignment of a one-bit wide bit-field to another, but the compiler would have assumed only one selection was possible. This may have caused subsequent code to fail.

Bad code with shift (9.63PL4) When shifting by a constant amount an indirect location, e.g. `*ptr <<= 6;` the produced code may fail.

Fixup overflow with increment (9.63PL4) This error may occur for code accessing some bitfield variables.

Initialization of large auto objects (9.63PL4) Auto objects larger than 4 bytes in size could not be initialized. This has been corrected. Large global or static objects were initialized correctly.

Bank selection issue with bit assignment (9.63PL4) The wrong bank may have been selected when assigning a boolean to a bit variable resulting in the wrong memory location being changed.

Logical operations creating temporary bit objects (9.63PL4) Some optimizations of the && and || operators were creating temporary bit variable to hold intermediate results and which would result in code failure.

Better memory allocation (9.63PL4) Space for the stack in common memory is now allocated from the top down. This may help in better placement of

Overwrite of variable in WREG (9.63PL4) In rare instances a variable cached in WREG may be overwritten by code which loads the table pointer registers.

Interrupt function context (9.63PL4) Interrupt functions which used or clobbered the TABLAT register weren't including it in context saving. This has now been corrected.

Accessing local const arrays (9.63PL4) In some cases the code generator would emit code that incorrectly accessed the members of local const arrays. This has been corrected.

Incorrect comparison code (9.63PL4) Code may fail when comparing an unsigned byte and an unsigned word.

TABLAT register correctly saved on interrupt (9.63PL4) This register is now saved if any of the TBLPTR registers are also scheduled for saving by the context switch code.

Missing basic type (9.63PL4) Object declarators which are missing a basic type specifier default to type `int`. The parser would fail to emit a warning message when this occurred if the object declarator also contained a qualifier or storage class specifier. This has now been corrected.

Mixed memcpy sources (9.63PL4) Under some circumstances the conversion of a 16 bit data pointer to a 24 bit mixed pointer would be incorrect. For example in programs which used the standard library function memcpy with source addresses in both the code and data address spaces. This has been now been corrected.

Undefined symbol (9.63PL3) In some arrangements of code involving multiply-assignment for byte objects, this resulted in an undefined symbol: bmul.

Lite mode (9.63PL2) The level of optimization has been improved for Lite mode for certain expressions involving SFRs. This is to ensure hardware requirements are met when certain sequence of instructions have to occur within a set time frame (eg: EEPROM read/write).

Long addition (9.63PL2) An issue was identified involving expressions that did a long integer addition between a byte and a literal. This was corrected and a more optimal solution was implemented.

Debugging with duplicated functions (9.63PL2) For duplicated functions (those being called from an interrupt and main-line code) some symbol information may have caused difficulties when viewing them in a debugger (eg: MPLAB).

Absolute functions (9.63PL2) It was found that a function that followed an absolute function had its psect definition incorrect. This has been corrected.

Typedef declarations (9.63PL1) In some cases typedefs made explicitly in a program or implicitly by the parser were being output in the intermediate p-code later than required by the code generator. This would cause the code generator to fail or in some instances crash. This has now been corrected.Can't generate code (9.63PL1) For some statements involving the additive assignment of floating-point variables the code generator was unable to generate code. This has been corrected.

TBLPTR corruption accessing const data (9.63PL1) In some instances where programs have less than 64k of constant data the TBLPTR could be corrupted whilst accessing said data. This would only happen when accessing data at a program address before a 100h or 10000h byte boundary for programs with less than 255 bytes (smallconst) or greater than 255 bytes and less than 64k bytes (mediumconst) of constant data respectively. This has been corrected by programming a dummy byte at the end of either the smallconst or mediumconst psects.

Syntax Error (9.63PL1) Some projects may have failed compilation with a syntax error in the .SDB file.

Constant Declaration (9.63) The parser would mistakenly emit a warning message "attempt to modify object qualified const" for const definitions in block scope.

Benign Redefinitions (9.63) Benign function redefinitions, (e.g. void fred(); vs void fred(void)) are now permitted by the parser.Symbols Clashing (9.63) Changes made to avoid clashes between existing symbols and those formed from a function_name + local_name construct.

Non-use of Access Bank Memory for autos (9.63) The code generator was not allocating the auto/parameter block to the access bank in situations where it would normally be able to do so. The resulting code would work correctly, but would leave a section of the access bank unallocated, place more variables in the banked memory, and the code size would increase as a result. This issue does not apply to programs which have a large auto/param block that does not fit in the access bank.

Skipping Bank Selection (9.63) With bit skip instructions, it is possible that the operand of the next instruction lies in different bank. In this scenario it would skip the bank selection and not the instruction itself.

Logical Left Shift by 7 (9.63) In some instances the code generator would produce incorrect code while shifting byte sized unsigned by 7.

Incorrect Symbols in Duplicated Functions (9.63) The output of the code generator when it was processing duplicated functions (called from both main-line and interrupt code) was not using the special symbol prefix while referencing symbols for temporary variables. Other local symbols such as autos and parameter variables were using the correct symbols and this issue will only affect code that has reentrantly called functions that allocate temporary variables.

Unnecessary Modification of TBLPTRH (9.63) In some circumstances while using mixed 16 bit pointer to access the program space, the TBLPTRH would be modified unnecessarily.

5.3 Assembler

Bank selection optimization (9.63PL4) The assembler optimizer may remove required bank selection instructions when checking code sequences after a btfsx instruction.

Procedural Abstraction (9.63PL3) An error was detected in the algorithm used to determine if this optimization is worth doing. In some cases the optimization may have occurred and resulted in functional, but larger code.

Code removal (9.63PL3) In some circumstances code which was procedurally abstracted may have later been incorrectly removed by another optimization.

Psect Alignment (9.63PL3) The psect into which code that was able to be procedurally abstracted was missing its delta flag. This resulted in this psect having the potential to not be word aligned. When this occurred, code in this psect would not execute correctly.

Movff Removal (9.63PL2) Corrected an optimizations that in some circumstances was removing what it believed was a redundant movff instruction.

Common subexpression (9.63PL2) An optimization that looked for common code within if/else statements was in some instances incorrectly matching differing code, as being the same.

5.4 Linker

mediumconst psect (9.63PL1) The map file would erroneously report that this psect was linked into the CODE linker class and not the MEDIUMCONST class. This has been corrected.

5.5 HEXMATE

5.6 Cromwell

Break points in interrupts (9.63PL3) An issue was identified which caused difficulty in setting break points in interrupt functions.

5.7 Libraries

ReadSPI2 (9.63PL4) This function was using the BF bit instead of BF2.

TCP/IP Stack Work-around (9.63PL1) When the preprocessor macros __TCP_H or __TCPIP_H are defined, the new Microchip-compatible SFR definitions are not automatically included. This avoids multiple definitions

Undefined Symbols (9.63PL1) Using the Flash / EEPROM libraries could cause undefined symbol errors to appear.

idloc_read Function (9.63) A post increment on a table read operation caused the TBLPTR to return a +1 address value which caused a problem when invoked from idloc_write function.

5.8 Chip Configuration File

Memory sizes (9.63PL2) Flash erase/write size and EEPROM memory size for 18F46K20 and 18F26K20 has been corrected.

5.9 Header Files

18F87K22 configuration bit settings (9.63PL4) The unprogrammed value for the first configuration word on this device was incorrect. This would mean that if the values for some settings in this word were not specified (left at default) they would program incorrect in the device.

Incorrect macro name (9.63PL3) A macro (MODE0) in `sw_spi.h` was clobbering a bit of the same name in 18F87Jxx devices.

Configuration bits (9.63PL2) An issue was identified in J devices which left unimplemented configuration bits unprogrammed. This resulted in device programmer errors. The unimplemented bit (config1H bit 3) now gets set to zero to avoid this issue.

Assembler header files (9.63PL2) An issue was identified in the process used to create assembler header files which may have caused some invalid definitions to be created.

Configuration Masks (9.63) WDTDIS, EXTOSC and INTOSC configuration mask have been corrected.

Chapter 6

Addendum

6.1 List of Supported Devices

The following is the full list of all devices supported by this version of HI-TECH C Compiler for PIC18 MCUs (PRO) sorted alpha-numerically.

18C242, 18C252, 18C442, 18C452, 18C601, 18C658, 18C801, 18C858, 18F1220, 18F1230, 18F1320, 18F1330, 18F13K20, 18F13K22, 18F13K50, 18F14K20, 18F14K22, 18F14K50, 18F2220, 18F2221, 18F2320, 18F2321, 18F2331, 18F23K20, 18F23K22, 18F2410, 18F242, 18F2420, 18F2423, 18F2431, 18F2439, 18F2450, 18F2455, 18F2458, 18F248, 18F2480, 18F24J10, 18F24J11, 18F24J50, 18F24K20, 18F24K22, 18F2510, 18F2515, 18F252, 18F2520, 18F2523, 18F2525, 18F2539, 18F2550, 18F2553, 18F258, 18F2580, 18F2585, 18F25J10, 18F25J11, 18F25J50, 18F25K20, 18F25K22, 18F25K80, 18F2610, 18F2620, 18F2680, 18F2682, 18F2685, 18F26J11, 18F26J13, 18F26J50, 18F26J53, 18F26K20, 18F26K22, 18F26K80, 18F27J13, 18F27J53, 18F4220, 18F4221, 18F4320, 18F4321, 18F4331, 18F43K20, 18F43K22, 18F4410, 18F442, 18F4420, 18F4423, 18F4431, 18F4439, 18F4450, 18F4455, 18F4458, 18F448, 18F4480, 18F44J10, 18F44J11, 18F44J50, 18F44K20, 18F44K22, 18F4510, 18F4515, 18F452, 18F4520, 18F4523, 18F4525, 18F4539, 18F4550, 18F4553, 18F458, 18F4580, 18F4585, 18F45J10, 18F45J11, 18F45J50, 18F45K20, 18F45K22, 18F45K80, 18F4610, 18F4620, 18F4680, 18F4682, 18F4685, 18F46J11, 18F46J13, 18F46J50, 18F46J53, 18F46K20, 18F46K22, 18F46K80, 18F47J13, 18F47J53, 18F6310, 18F6390, 18F6393, 18F63J11, 18F63J90, 18F6410, 18F6490, 18F6493, 18F64J11, 18F64J90, 18F6520, 18F6525, 18F6527, 18F6585, 18F65J10, 18F65J11, 18F65J15, 18F65J50, 18F65J90, 18F65K22, 18F65K80, 18F65K90, 18F6620, 18F6621, 18F6622, 18F6627, 18F6628, 18F6680, 18F66J10, 18F66J11, 18F66J15,

18F66J16, 18F66J50, 18F66J55, 18F66J60, 18F66J65, 18F66J90, 18F66J93, 18F66K22, 18F66K80, 18F66K90, 18F6720, 18F6722, 18F6723, 18F67J10, 18F67J11, 18F67J50, 18F67J60, 18F67J90, 18F67J93, 18F67K22, 18F67K90, 18F8310, 18F8390, 18F8393, 18F83J11, 18F83J90, 18F8410, 18F8490, 18F8493, 18F84J11, 18F84J90, 18F8520, 18F8525, 18F8527, 18F8585, 18F85J10, 18F85J11, 18F85J15, 18F85J50, 18F85J90, 18F85K22, 18F85K90, 18F8620, 18F8621, 18F8622, 18F8627, 18F8628, 18F8680, 18F86J10, 18F86J11, 18F86J15, 18F86J16, 18F86J50, 18F86J55, 18F86J60, 18F86J65, 18F86J72, 18F86J90, 18F86J93, 18F86K22, 18F86K90, 18F8720, 18F8722, 18F8723, 18F87J10, 18F87J11, 18F87J50, 18F87J60, 18F87J72, 18F87J90, 18F87J93, 18F87K22, 18F87K90, 18F96J60, 18F96J65, 18F97J60, 18LF13K22, 18LF13K50, 18LF14K22, 18LF14K50, 18LF23K22, 18LF24J10, 18LF24J11, 18LF24J50, 18LF24K22, 18LF25J10, 18LF25J11, 18LF25J50, 18LF25K22, 18LF25K80, 18LF26J11, 18LF26J13, 18LF26J50, 18LF26J53, 18LF26K22, 18LF26K80, 18LF27J13, 18LF27J53, 18LF43K22, 18LF44J10, 18LF44J11, 18LF44J50, 18LF44K22, 18LF45J10, 18LF45J11, 18LF45J50, 18LF45K22, 18LF45K80, 18LF46J11, 18LF46J13, 18LF46J50, 18LF46J53, 18LF46K22, 18LF46K80, 18LF47J13, 18LF47J53, 18LF65K80, 18LF66K80

6.2 Microchip Errata

This release of the PICC-18 compiler recognises the published silicon errata issues listed in the table below. Some of these issues have been corrected and no longer apply in recent silicon revisions. Refer to Microchip's device errata documents for details on which issues are still pertinent for your silicon revision. The compiler's chip configuration file records which issues are applicable to each device. Specific errata workarounds can be selectively enabled or disabled via the driver's `--ERRATA` command line option.

PICC-18 Errata Workarounds

Name	Description	Workaround details
`4000`	Execution of some flow control operations may yeild unexpected results when certain instructions vector code execution across the 4000h address boundary.	A continuous block of program code is not allowed to grow over the 4000h address boundary. Additional `NOP` instructions are inserted at prescribed locations.
Continued...		

PICC-18 Errata Workarounds

Name	Description	Workaround details
FETCH	Some instruction fetches can be corrupted.	Insert a NOP (0xFFFF) after every TBLRD and return type instruction, at the target of every GOTO and CALL and at the interrupt vector addresses
LFSR	Using the LFSR instruction to load a value into a specified FSR register may also corrupt a RAM location.	The compiler will load FSR registers without using the LFSR instruction.
DAW	The DAW instruction may improperly clear the CARRY bit (STATUS<0>) when executed.	The compiler is not affected by this issue.
MINUS40	Table read operations above the user program space (>1FFFFFh) may yield erroneous results at the extreme low end of the device's rated temperature range (-40^{o}C).	Affected library sources *config.c*, *idloc.c* and *devread.c* employ additional NOP instructions at prescribed locations.
EEDAT	When reading EEPROM, the contents of the EEDATA register may become corrupted in the second instruction cycle after setting the RD bit (EECON1<0>).	The EEPROM_READ macro and eeprom_read library function read EEDATA immeadiately.
EEADR	The result returned from an EEPROM read operation can be corrupted if the RD bit is set immediately following the loading of the EEADR register.	The compiler is not affected by this issue.
EELVD	Writes to EEPROM memory may not succeed if the internal voltage reference is not set.	No workaround applied
FLLVD	Writes to program memory may not succeed if the internal voltage reference is not set.	No workaround applied
RESET	A GOTO instruction placed at the reset vector may not execute.	Additional NOP instruction inserted at reset vector if following instruction is GOTO.

Continued...

PICC-18 Errata Workarounds

Name	Description	Workaround details
FASTINTS	If a high-priority interrupt occurs during a two-cycle instruction which modifies WREG, BSR or STATUS, the fast-interrupt return mechanism (via shadow registers) will restore the value held by the register before the instruction.	Additional code reloads the shadow registers with the correct values of WREG, STATUS and BSR.
BSR15	Peripheral flags may be erroneously affected if the BSR register holds the value 15, and an instruction is executed that holds the value C9h in its 8 least significant bits.	Compiler avoids generating MOVLB 15 instructions. A warning is issued if this instruction is detected.
TBLWTINT	If a peripheral interrupt occurs during a TBLWT operation, data can be corrupted.	Library routine *flash_write()* will temporarily disable all applicable interrupt-enable bits before execution of a TBLWT instruction.
FW4000	Self write operations initiated from and acting upon a range within the same side of the 4000h boundary may fail based on sequences of instructions executed following the write.	No workaround applied
RESETRAM	Data in a RAM location can become corrupted if an asynchronous reset (eg. WDT, MCLR event) occurs during a write operation to that location.	A warning will be issued if the length *nvram* psect is greater than zero bytes (*persistent* variables populate this psect).