Holtek C Compiler V3 FAQ

Revision: V1.80 Date: September 13, 2021

www.holtek.com

Notice

1. This document may be not the latest version. As Holtek's tools and documents will continue to be updated, some dialog boxes and tool descriptions in actual use may differ from the contents of this document. For the most up-to date information, visit the Holtek website at:

   http://www.holtek.com.tw/en/mcu_tools_users_guide

2. It is assumed that the reader already has the following basic qualities:
   - Knows how to write C programs
   - Has already read and understood the target MCU datasheet

Table of Contents

Part I V3 Introduction

1.1 V3 Version

A:

Release date	V3 Compiler version	IDE version
2012/12	C Compiler V3.10	HT-IDE30007.7
2013/10	C Compiler V3.20	HT-IDE30007.8
2014/03	C Compiler V3.30	HT-IDE30007.82
2014/09	C Compiler V3.31	HT-IDE30007.85
2015/01	C Compiler V3.40	HT-IDE30007.86
2015/11	C Compiler V3.41	HT-IDE30007.89
2016/06	C Compiler V3.42	HT-IDE30007.90
2016/12	C Compiler V3.50	HT-IDE30007.93
2017/05	C Compiler V3.51	HT-IDE30007.94
2017/12	C Compiler V3.52	HT-IDE30007.96
2018/07	C Compiler V3.53	HT-IDE30007.97
2018/11	C Compiler V3.54	HT-IDE30007.98
2020/09	C Compiler V3.59	HT-IDE30008.04

1.2 What are the increased functions in the new version

A:

V3.51, V3.52, V3.53, V3.54, V3.59

- Modify bugs.

V3.50

- Support bit data type (more details can be obtained in chapter 2.2.11 of the <C Compiler V3 user's guide>))
- Modify bugs.

V3.42

- Support the function of hardware multiplication and division (when IC has the MDU registers, more details can be obtained in chapter 2.2.10 of the <C Compiler V3 user's guide>)
- Modify bugs.

V3.41

- Modify when without the option –Os, part of MCUs fail to write EEPROM
- Modify bugs

V3.40

- Modify all known bugs of V3.31
- Optimize the RAM space allocation of extended instruction MCU, more details can be obtained in chapter 10.1 of <C Compiler V3 user's guide>

V3.31

- Support for when the entry function and the main function are in different files.
- Modify bug – run error when the function parameter is const array
- Supports the internal function: GCC_DELAY(n), more details can be obtained in chapter 2.2.3 of the <C Compiler V3 user's guide>

V3.30

- Supports to specify the program entry function, more details can be obtained in chapter 2.2.9 of the <C Compiler V3 user's guide>
- Modify the startup function to avoid the use of the TABRD instruction
- Causes an error when a function parameter type is missing

V3.20

- Supports floating/double data type and C Standard libraries
- Supports MCUs that have extended instructions, such as the HT66F70A
- Supports const variable to specify address, more details can be obtained in chapter 2.2.7 of the <C Compiler V3 user's guide>
- Supports function to specify address, more details can be obtained in the <C Compiler V3 user's guide> section 2.2.8

1.3 What are the V3 user's guides?

A: http://www.holtek.com.tw/en/mcu_tools_users_guide

<C Compiler V3 user's guide>

<Holtek C Compiler V3 FAQ>

<Standard library user's guide>

1.4 What are the MCUs that V3 does not support?

A: V3 does not support MCUs that the MP register width less than 8 bits. The following list shows these devices. Extended instruction MCUs are only supported by the V3.20 version or above.

MCU name			
HT45F2Y	HT46R46	HT48E30	HT66F03M
HT45R04	HT46R46-H	HT48F06E	HT66F03T3
HT45R0G	HT46R47-H	HT48F10E	HT66F13
HT45R34	HT46R48A	HT48F30E	HT66F20
HT45R35	HT46R51	HT48R002	HT66F23D
HT45R35V	HT46R52	HT48R003	HT66F30
HT45R36	HT46R53	HT48R005	HT66FB30
HT45F39	HT46R54	HT48R006	HT66FU30
HT46C22	HT46R71D	HT48R01A	HT68F002
HT46R22	HT46R71D-1	HT48R02	HT68F003
HT46C46E	HT46R72D-1	HT48R063	HT68F03
HT46R46E	HT46R72D-1A	HT48R063B	HT68F03C
HT46C47	HT46R73D-1	HT48R064	HT68F03M
HT46R47	HT46R73D-1A	HT48R064B	HT68F03T3
HT46C47E	HT46R74D-1	HT48R064D	HT68F13
HT46R47E	HT46RU22	HT48R064G	HT68F20
HT46C48AE	HT46R92	HT48R07A-1	HT68F30
HT46R48AE	HT48C05	HT48R08A-1	HT68FB30
HT46C62	HT48R05A-1	HT48R09A-1	HT68FU30
HT46R62	HT48C06	HT48R52	HT82J97A
HT46F46E	HT48R06A-1	HT48R52A	HT82J97E
HT46F47E	HT48C062	HT48R53	HT82K72A
HT46F48E	HT48R062	HT48RA0-5	HT82M39
HT46R002	HT48C10-1	HT48RA0-6	HT82M39B
HT46R003	HT48R10A-1	HT49C10-1	HT82M72A
HT46R003B	HT48C30-1	HT49C30-1	HT82M98
HT46R02	HT48R30A-1	HT49R30A-1	HT82M99A
HT46R004	HT48CA0	HT49C30L	HT82M99E
HT46R005	HT48RA0A	HT49CA0	HT82M99AE
HT46R01A	HT48CA0-1	HT49RA0	HT82M99EE
HT46R064	HT48RA0-1	HT49RA0-6	HT83020
HT46R064B	HT48CA0-2	HT49R10A-1	HT83F10
HT46R064D	HT48RA0-2	HT56R22	HT83F20
HT46R064G	HT48CA0-3	HT56R62	HT83F40
HT46R12A	HT48RA0-3	HT66F002	HT83F60
HT46R32	HT48CA6	HT66F003	HT83F80
HT46R321	HT48E06	HT66F03	HT83P00-1
HT46R322	HT48E10	HT66F03C	HT83R00

1.5 What are extended instructions?

A: Any the extended instructions will be preceded by the letter ‘L’. For example: LMOV and LSET which have a length of 2 words. Whether an MCU has extended instructions or not can be determined by looking at the datasheet. Each extended instruction occupies one cycle more than a general instruction.

Part II Differences between V3 and V2

2.1 What are the syntax differences between V3 and V2 compared with V1, V2 and standard C?

A: The syntax differences between V3 and V2 are absolute address variables, interrupt syntax and integrated assembler. More details can be obtained in the user's guide <C Compiler V3 user's guide> section 2.2

The comparison table for V3, V2, V1 and C are in the user's guide <C Compiler V3 user's guide> chapter 4.

2.2 What are the advanced functions of V3 over V2?

A:

	V3	V2
Global Variables	Supports initialization, refer to <C Compiler V3 user's guide> section 2.2.4	Does not support initialization
Const Variables	supports a maximum size of 64 pages supports extern const supports Const Variables Specified Address,refer to <C Compiler V3 user's guide> section 2.2.7	There may be an error when the size is more than 1 page.
Array	supports more than three-dimensional arrays	Only supports less than two-dimensional arrays
ISR	can call a function, refer to <C Compiler V3 user's guide> section 2.2.1	Is not able to call a function
Function	Support specify the program entry function,refer to <C Compiler V3 user's guide> section 2.2.9	Can not specify the entry function

2.3 Common errors when changing V2 programs to V3 programs

2.3.1 ISR warning

e.g.

```
#pragma vector Int_isr @ 0x04
void Int_isr() {}
```

warning: ignoring #pragma vector Int_isr [-Wunknown-pragmas]

Solution:

Use the correct interrupt grammar:

```
void __attribute((interrupt(0x04))) Int_isr() {}
```

More details can be obtained in the user's guide <C Compiler V3 user's guide> section 2.2.1

Note:

i: If the warning is not amended, the program can continue to be compiled, but the compiler will process the function as a normal function, not as an interrupt service program.

ii: If the other keywords of #pragma, such as rambank/function etc., are used in V3, it will issue a warning and indicate the function invalid.

2.3.2 Inline assembly error

e.g.

```
#asm
nop
#endasm
```

error: invalid preprocessing directive #asm

error: invalid preprocessing directive #endasm

Solution:

```
Use the correct inline assembly grammar:asm("nop");
```

More details can be obtained in <C Compiler V3 user's guide> section 2.2.5

2.3.3 Bit variable error

e.g.bit a;

error: unknown type name 'bit'

Solution: use the HT-IDE3000 7.93 version or above

2.3.4 Bit flag error

e.g. _40_1 = 1;

error: '_40_1' undeclared (first use in this function)

Solution:

a. Use the structure bit-field to define the bit flag

```
bit_type bit_var __attribute__ ((at(0x40)));
#define _40_1 bit_var.bit1
```

More details can be obtained in the user's guide <C Compiler V3 user's guide> section 2.2.3

b. Use the bit type:

```
static volatile bit flag1 __attribute__ ((at(0x40),bitoffset(1)));
```

More details can be obtained in the user's guide <C Compiler V3 user's guide> section 2.2.11

2.3.5 Internal function error

e.g. _delay(2);

Error(L2001): Unresolved external symbol '__delay' in file

Solution:

Modify it to:

```
#include "ht66f50.h"
GCC_DELAY(2);
```

More details can be obtained in the user's guide <C Compiler V3 user's guide> section 2.2.3

2.3.6 Absolute address variable error

e.g. unsigned char a @ 0x40;

error: stray '@' in program

error: expected '=', ',', ';', 'asm' or '__attribute__' before numeric constant

Solution:

Modify it to:

```
volatile static unsigned char var_name __attribute__ ((at(0x40)));
```

More details can be obtained in the user's guide <C Compiler V3 user's guide> section 2.2.2

2.3.7 Function pointer error

e.g.

```
void FileFunc(){}
void EditFunc(){}
void main()
{
    typedef void (*funcp)(void);
    funcp pfun= FileFunc;
    pfun();
    pfun = EditFunc;
    pfun();
}
```

error: incompatible types when initializing type ‘funcp’ using type ‘void()’
error: incompatible types when assigning to type ‘funcp’ from type ‘void()’

Solution: V3 does not presently support function pointer.

Part III Special syntax and usage of V3

3.1 How to define a variable for the specified bank?

A: If the MCU without extended instructions, it can only define a variable for the specified address, such as:

volatile static unsigned char var_name __attribute__ ((at(0x140))); More details can be obtained in the user's guide <C Compiler V3 user's guide> section 2.2.2

If the MCU with extended instructions, then there is no need to specify the bank, the linker will assign an arbitrary bank automatically for a variable.

3.2 How to define a function for the specified address?

A: This function is only supported by the IDE 7.8 version or above, grammar:

```
char __attribute__((at(0x373))) foo (char parm){}
```

This means that to specify the function foo at the address 0x373

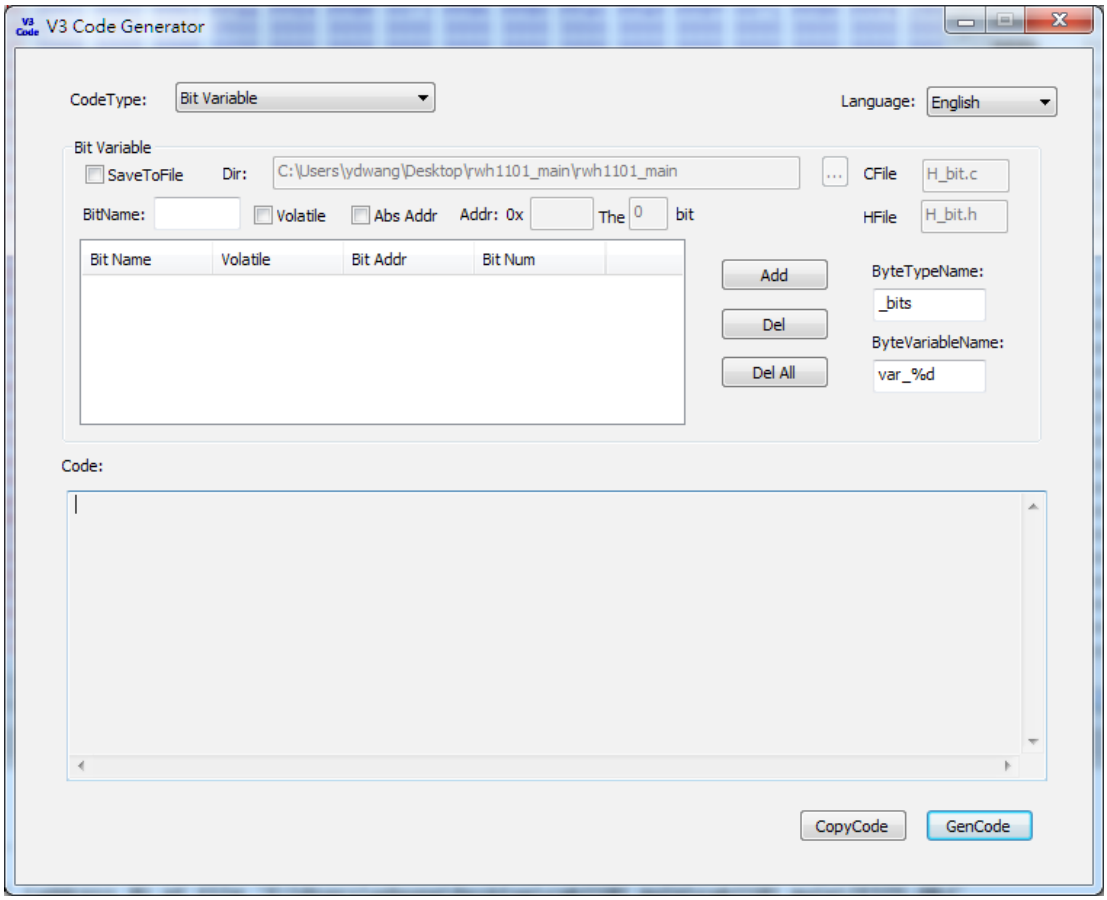
More details can be obtained in the user's guide <C Compiler V3 user's guide> section 2.2.6

3.3 How to use mixed language in V3?

A: Refer to <C Compiler V3 user's guide> section 2.5

3.4 V3 Code Generator

In order to make it easier for users to use the V3 specific syntax, the IDE3000 v7.83 or later versions supports a "V3 code generator" tool. This is located in the menu → tool → V3 code generator. It can output the bit variables, interrupt grammar, absolute addresses of variables, the internal assembler, delay function and bank specify variables. More details can be obtained in the <HT-IDE3000 manual> section "V3 code generator".


V3 Code Generator
CodeType:
Bit Variable
Language:
English
Bit Variable
SaveToFile
Dir:
C:\Users\ydwang\Desktop\rwh1101_main\rwh1101_main
CFile
H_bit.c
BitName:
Volatile
Abs Addr
Addr: 0x
The
0
bit
HFile
H_bit.h
Bit Name
Volatile
Bit Addr
Bit Num
Add
Del
Del All
ByteTypeName:
_bits
ByteVariableName:
var_%d
Code:
CopyCode
GenCode

Part IV Common errors, warnings and solutions in V3

4.1 error “multi-ram-bank should be equipped with mp1”

A: Confirm that that the MCU has extended instructions. If it is, then use the IDE 7.8 version or above.

4.2 error "internal compiler error:xxxx”

A: Compiler internal error - contact Holtek.

4.3 error (L1038) “RAM (bank0) overflow, memory allocation fails for section ….”

A: For without extended instruction architecture MCUs, the C Compiler will assign the variables to RAM bank0 (extended instruction MCUs can assign the variables to any bank automatically) by default. When bank0 is full, RAM bank 0 overflows and the following message will be generated:

- Check the data type is correct or not, especially the programs from V1 C Compiler
- If it is a multiple RAM bank MCU, locate the global variables to other banks manually - refer to 3.1

4.4 error (L1038) “ROM/RAM (bank*) overflow, memory allocation fails for section ….”

A: When there is not enough ROM or RAM space, the solution is as follows:

- Check if the optimised parameter -Os is enabled or not, refer to the <C Compiler V3 user’s guide> section 2.1.4
- Delete unnecessary programs.

4.5 warning(L3010) (absolute address: xxh, length:x) is overlay with (absolute address: xxh, length: x)

A: There are two situations which may cause these warnings:

- The same absolute address variables are defined many times in different files, such as the variable var is defined in a.h:
  static volatile unsigned char var __attribute__ ((at(0x180)));
  When t1.c and t2.c both include a.h simultaneously, then a warning message will be generated. In this case, the warning message can be ignored or set the option to avoid the warning message. Refer to <C Compiler V3 user’s guide> section 2.1.5
- The defined addresses of different variables overlap, shown as follows, the addresses of _b and _a overlap, _b needs to be defined in the address 0x0142.

```
DEFINE_SFR(unsigned int _a, 0x0140);
DEFINE_SFR(unsigned char _b, 0x0141);    //error
```

4.6 warning (L3009): Same sub function exists between ISR(04H) CMG and MAIN CMG: _func

A: Exist the same sub function(_func) between the interrupt service routine (04H) and the main function, solution:

- Avoid the common calling

More details can be obtained in chapter 2.2.1 of <C Compiler V3 user’s guide>

Part V Common questions and solutions in V3

5.1 How to use bit variables in V3?

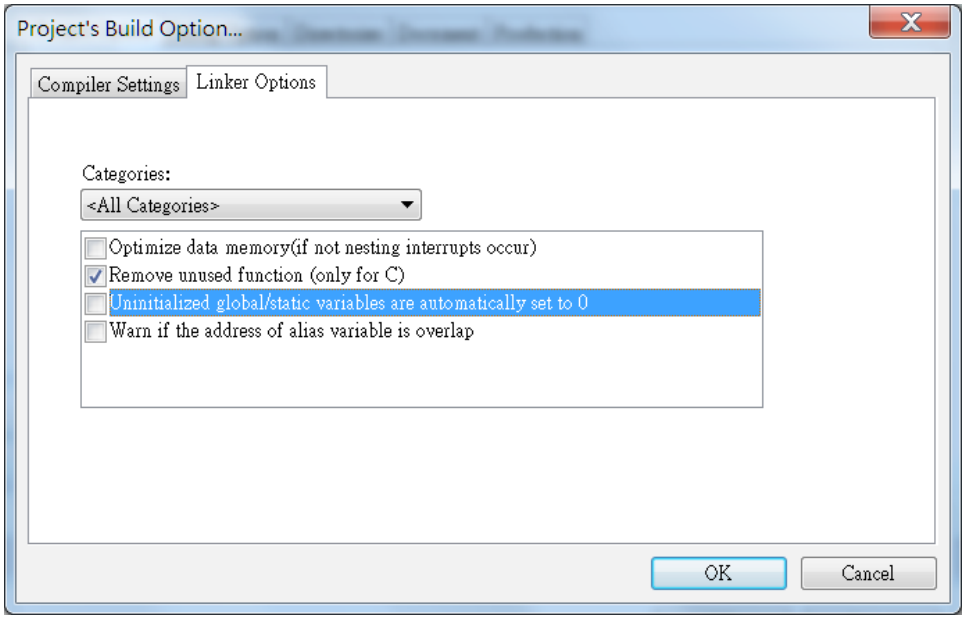
A: bit flag1; (More details can be obtained in the user's guide <C Compiler V3 user's guide> section 2.2.11).

5.2 How to use external defined bit variables in V3?

A: extern bit flag1;

5.3 Solution for when variables are cleared to 0 after a program reset?

A: IDE7.8 version supports a way in which variables are not initialized: the option "Uninitialized global/static…" does not need to be checked.



5.4 How to quote the specified address in other files?

A: The variables (not const) which are specified addresses need to be defined as “static”. If the action scope is only in the current file, it can be defined in the header file. If other files need to use it, then this header file needs to be included directly. Such as:

```
//Define_var.h
static volatile unsigned  var1  __attribute__ ((at(0x180)));
//test1.c
#include "Define_var.h"
void foo1()
{
   var1 = 1;
}
//test2.c
#include "Define_var.h"
void foo2()
{
   var1 = 2;
}
```

Note: If it is a const variable, then there is no need for it to be defined as static, extern can be used instead, such as:

```
//test1.c
const int __attribute__((at(0x3400))) bb[3]={1,2,3};
//test2.c
extern const int  bb[3];
int b;
void fun()
{
   b=bb[2];
}
```

5.5 For MCUs which have an EEPROM write limitation (need to write “set wren, wr, flag” continuously), how to use V3 to write to the EEPROM?

A:

i: In V3, _rden and _rd are in bank1, using extended instructions, is different from the specification described in the datasheet.

ii: For this function that has strict requests to instructions, it is recommend to use the internal assembler for its implementation. This is because it is not confirmed that, if C language is used, it will translate the programs is in a specific way.

iii: To get consecutive instructions, the program in V3 should be changed as follows:

```
typedef struct {
    unsigned char bit0 : 1;
    unsigned char bit1 : 1;
    unsigned char bit2 : 1;
    unsigned char bit3 : 1;
    unsigned char bit4: 1;
    unsigned char bit5 : 1;
    unsigned char bit6 : 1;
    unsigned char bit7: 1;
}iar_bits;
DEFINE_SFR(iar_bits, iar1, 0x02);
#define     iar1_3          iar1.bit3
#define     iar1_2          iar1.bit2
#define     iar1_1          iar1.bit1
#define     iar1_0          iar1.bit0  //rd
```

```
unsigned short EERead(void)
{   unsigned short Backup;
    _mp1=0x40;
    Backup=_bp; _bp=1;
    iar1_1=1; iar1_0=1;
    while(iar1_0);
    _iar1=0;
    _bp=Backup;
    return _eed;
}

void EEWrite(unsigned short EEdata)
{   unsigned short Backup;
    _eed=EEdata;
    _mp1=0x40;
    Backup=_bp; _bp=1;
    _emi=0;
    iar1_3=1; iar1_2=1;
    _emi=1;
    while(iar1_2);
    _iar1=0;
    _bp=Backup;
}
```

5.6 Notes for assigning a variable to a bit flag using the V3 Compiler

Example:

```
unsigned char flag;
_pa2=flag;
```

The asm code:	not:
CLR PA2 SZ _flag SET PA2	SZ _flag JMP L1 CLR PA2 JMP L2 L1: SET PA2 L2:

Description:

The compiler is only interested in the results of the calculation process, to reduce the output of instructions, the compiler will translate the left instructions.

C and assembly language are different, a statement not only translation of an instruction, so before the end of the statement is executed, the calculation is not complete.

Impact:

No matter what the value of the flag is, PA2 will be the first CLR, if an interrupt occurs and the interrupt is useful to the PA, it will affect the results.

Solution:

- Disable the interrupt before assigning to a bit flag, then enable it at the end of the calculation

More generally:

For calculation of a multi-byte variable, if an interrupt is useful to it, before calculation unfinished are allowed to enter an interrupt.

5.7 Notes for using the ROM BP in the V3 Compiler

For multi ROM BANK MCUs:

When using C language, users do not need to set the ROM BP. The Linker will set the ROM BP automatically. If users modify the ROM BP in the project, the programs will probably have an error. When setting up the RAM BP, users should also be careful not to modify the ROM BP.

When use mixed language:

In the C function call assembly section, it is necessary to use C language or inline asm (fcall),

In the assembly section call C function, it is necessary to setup the ROM BP before the call function and restore it later.

Example:

```
;;Test1.asm
extern _fun2:near
public _fun1
_fun1 .section  'code'
_fun1 proc
mov a, bank _fun2
mov [04H],a  ;;if ROM BP at 04h
call _fun2
mov a,bank_fun1
mov [04H],a
_fun1 endp
//Test2.c
extern void FUN1();
//or  asm("extern _FUN1:near");
void main()
{
   FUN1();
   //or asm("fcall _FUN1");
}
void fun2()
{}
```

5.8 Mixed language using ROM BP notes

Refer to section 5.7.

5.9 How to use the DOS command to compiler the C project?

The options of compiler, assembler, linker refer to the appendix C of <V3 C Compiler user's guide>

Example:

a. environment variable settings:

```
set HTCFG=C:\Program Files\Holtek MCU Development Tools\HT-IDE3000V7.x\MCU
set HTBIN=C:\Program Files\Holtek MCU Development Tools\HT-IDE3000V7.x\BIN
set HTINCLUDE=C:\Program Files\Holtek MCU Development Tools\HT-IDE3000V7.x\INCLUDE_V3
set HTLIB=C:\Program Files\Holtek MCU Development Tools\HT-IDE3000V7.x\LIB
```

b. compile the .c files

```
…\hgcc32.exe  t1.c -g -Os -I "%HTINCLUDE%" -o t1.asm
…\hgcc32.exe  t2.c -g -Os -I "%HTINCLUDE%" -o t2.asm
```

c. assemble the .asm files

```
…\hasmgcc32.exe /hide=12345678 /chip=HT66F50 /case /z "t1.asm"
…\hasmgcc32.exe /hide=12345678 /chip=HT66F50 /case /z "t2.asm"
```

d. link the all .obj,.lib files to .tsk

```
…\hlinkw32.exe /MCU=HT66F50 @ "C:\link-test.bat"
```

The content of link-test.bat:

```
"t1.obj"+
"t2.obj",
"test.tsk",
"test.map",
"test.dbg",
"libholtekgcc.lib";
```

5.10 Notes for using the table read in the ASM files of mixed language program

If a project has C file and ASM file, then the EMI flag should be clear during the table read in the ASM file .For example:

```
clr emi
tabrd r0
inc tblp
mov a,tblh
…
set emi
```

5.11 Notes for using inline assembly in interrupt

If there is inline assembly in the interrupt functions and the inline assembly use the special registers(such as MP, TBLP, TBHP, TBLH etc.),the user needs to save the register as follows:

```
DEFINE_ISR(isr04,0x04)
{
  asm("mov a,[01h]"); // mp0 = [01h]
  asm("mov temp_mp0,a");
  asm("mov a,80h");
  asm("mov [01h],a ");
  asm("mov a,[00h]");
  asm("mov a,temp_mp0");
  asm("mov [01h],a");
}
```

5.12 How to solve when modifying the const values in other ways (such as programming), the result of execution unchanged?

Example:

```
__attribut__((at(0x400)))
const unsigned char array[] = {0,1,2,3,4,5,6,7};
```

Clear the area 400H~410H when programming, then execute temp=array[7]; the result of temp is 7.

Solution:

Define array[] and temp = array[7]; in different C files.

5.13 Notes on the use of inline assembly language

The variables/functions/registers/flags used in inline assembly language should follow the definition of assembly language.

1. If the global variable/function is only used in the inline assembly language, the declaration should be added, such as:

```
asm("extern _a: byte");
asm("extern _func: near");
void main()
{
   asm("clr _a");
   asm("call _func");
}
```

2. Register/flag should be defined before use, you can include the INC document, such as:

```
asm("#include HT66F60.INC")
void main()
{
   asm("CLR ACC");
   asm("MOV TBHP,A");
   asm("CLR C");
}
```

3. Inline assembly language is case sensitive.

5.14 The address of the absolute address variable is occupied by other variables

If the absolute address variable is not used in the program, linker will assign other variables to this address.

Part VI Common optimization problems in V3

6.1 Variables debug messages cannot be seen on the watch window after using the V3 optimization parameters?

A: When using optimization parameters, variables may be deleted during optimization, therefore they will not be shown in the debug messages. To view the variable values when debugging, the variables can be defined as volatile temporarily, then deleted when debug is complete, such as:

```
volatile int i, j, k;
```

6.2 For interrupts and the general function access of the same global variable are the related statements of this global variable optimized?

A: There is no call relationship between the general function and interrupt. The compiler does not know when the interrupt occurs so it will influence the variables in the general function. Therefore it is recommend to define this kind of variables as volatile, such as:

Flag is used in the interrupt ISR_INT0 and the main function, then to define it as volatile:

```
volatile unsigned char flag;
```

```
    do
     {
     _emi=1;
     }while (flag==0);

//    _nop();
//    _nop();
    GCC_NOP();
    while(1);
}


void ISR_INT0()
{
  flag=1;
  _pb=0xff;          //0b00000001;
  _int0f=0;                 // clr interrput
}
```

Description: volatile: a type specifier. Designed to qualify the variables which are accessed or modified by different functions. Variables defined using volatile cannot be omitted because of compiler optimization.

Variables recommended to be defined with volatile: special registers, variables used in the interrupt functions, variables defined for some certain function codes (such as a delay function)

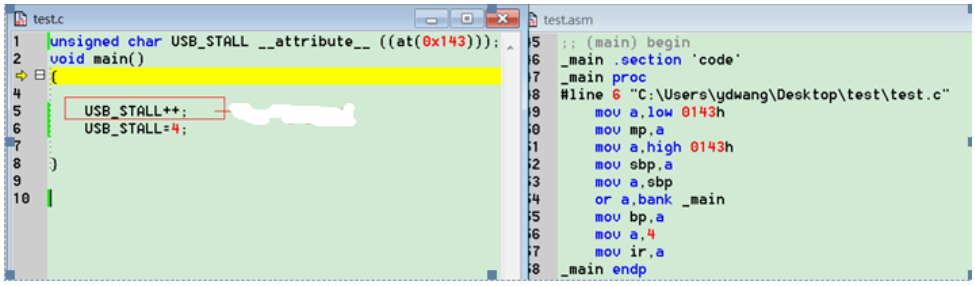
6.3 V3 optimization functions and its effect on debug?

A: More details can be obtained in the <C Compiler V3 user's guide> chapter 3

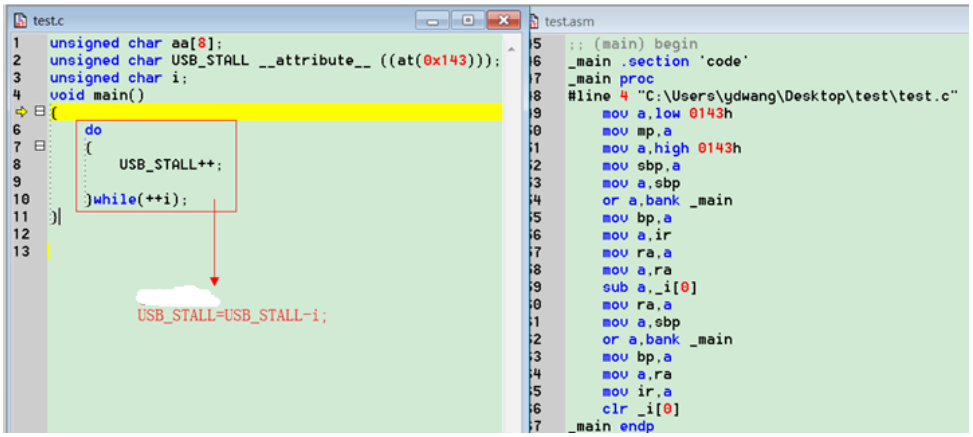
6.4 Line number error when using V3 compiler to debug?

A: The following cases may be shown:

a. Some statements may not be translated into code because of being optimized; there will also be no debug messages.



b. Several statements are translated into the same code, but only one line number is shown.



In this case, it may affect debug, but the execution results are without errors. If it is not in above two cases, then report.

6.5 How to solve the problem when code which is used for delay is optimized when using the V3 compiler?

A: As follows:

```
1   void delay2(unsigned int count)
2   {
3           unsigned int counts = count;
4           if(counts)
5           {
6                   counts--;
7           }
8   }
9
10  void main()
11  {
12      while(1)
13      {
14          delay2(10);
15      }
16  }
```

```
include HT66F50.inc
@code .SECTION 'CODE'
;1  void delay2(unsigned int count)
;2  {
;3          unsigned int counts = count;
;4          if(counts)
;5          {
;6                  counts--;
;7          }
;8  }
;9
;10 void main()
;11 {
    JMP _main_startup1
@start .SECTION 'CODE'
_main_startup1:
@start .SECTION 'CODE'
    MOV A,  00H
    MOV BP, A
    JMP _L3
_main:
_main:
_L3:
    JMP $
data .SECTION 'DATA'
```

Solution: Define the variable as volatile, as follows:

```
1   void delay2(unsigned int count)
2   {
3       volatile unsigned int counts = count;
4       if(counts)
5       {
6               counts--;
7       }
8   }
9
10  void main()
11  {
12      while(1)
13      {
14          delay2(10);
15      }
16  }
```

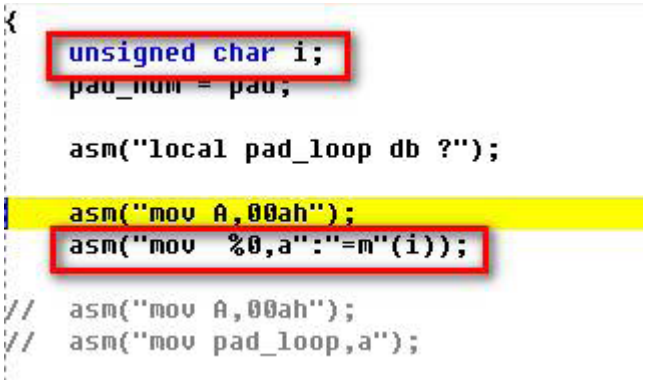
```
_main:
_main:
_L4:
;12     while(1)
;13     {
;14         delay2(10);
    MOV A,  0AH
    MOV _delay2_2,  A
    CLR _delay2_2[1]
    CALL    _delay2
    JMP _L4
data .SECTION 'DATA'
_delay2_2 DB 2 DUP (?) ; count
```

6.6 How to deal with the situation when inline assembly is optimized?

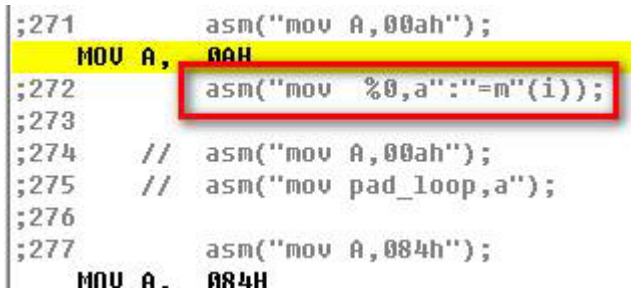
Example:

```
asm("mov %0,a":"=m"(i));//asgn the ACC register to i.
```

The variable i is unused in the following calculation, so the statement is optimised by the compiler.



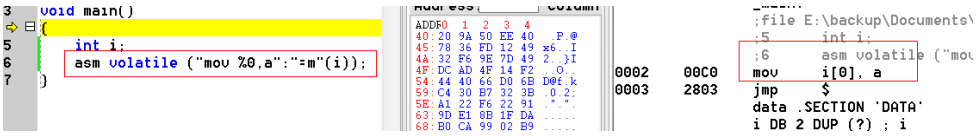
After compiled:



Solution:

Use the volatile keyword:

```
asm volatile ("mov %0,a":"=m"(i));
```



6.7 When select the optimization parameters , the delay time is changed?

A: The execution time of the delay function depends on the numbers of instructions executed. When select the optimization parameters, the instruction is reduced and then affect the delay time. The program developers should pay attention to this and adjust the delay function,or use the built-in function GCC_DELAY(n).

Part VII Fixing the Known Problems in C Compiler

7.1 There was an error when the total size of the const variable used by the program was 64 pages.

Problem Description

V3 Compiler supports const variables with a total size of no more than 64 pages, but it has been found to execute errors when the size is equal to 64 pages.

How to avoid the problem

Most programs use less than 64 pages of const variables, and if need, define a small number of const variables in assembly language.

7.2 When the range of the address specified by the Const exceeds 64pages, the address specified by the _CROM2PROM is invalid.

Problem Description

For example, define table1 at the address 0x100, table2 at the address 0x7000. If specify the _CROM2PROM at the address 0x7f00 in the project settings, the _CROM2PROM actual address will be not at 0x7f00.

```
__attribute__((at(0x100)))
const unsigned char table1[100] = {1,2};
__attribute__((at(0x7000)))
const unsigned char table2[100] = {1,2};
int a,b;
void main()
{
     a = table2[b];
}
```

How to avoid the problem

Fix the range of the const variable within 64 pages.

7.3 In a few cases, internal error occurs when a function is called within a do/while statement and if/else is used.

Problem Description

In a few cases, internal compiler error will occur: in extract_insn, at recog.c:2154

For eaxmple:

```
unsigned int  lg;
unsigned int  getadr;
void Exel_d(void)
{
   lg=100;
}
void lookfor(void)
{
   unsigned char m2;
   unsigned char n2;
   do
   {
       getadr=m2+n2;
       getadr/=2;
       Exel_d();
```

```
        if(getadr<lg)
        {
            n2=getadr;
        }
        else
        {
            m2=getadr;
        }
    }while(m2+1<n2);
}
```

How to avoid this problem

a. Define the getadr as volatile, or

b. Define the getadr, m2 and n2 with the same data type, unsigned char or unsigned int

7.4 The debug of the structure bit-field displays error information.

Problem Description

When the width of the structure bit-field is 8, the watch window value will display error (without affecting the execution results), for example:

```
struct INA
{
   unsigned char aa:1;
   unsigned char ab:8;
};
volatile struct INA A;
void main()
{
   A.ab = 0x22;
   A.ab += 0x33;
}
```

How to avoid the problem

When the bit-field width is 8, the number can be omitted, so it can be changed to:

```
struct INA
{
   unsigned char aa:1;
   unsigned char ab;
};
```

7.5 When local bit and switch are used at the same time, internal error will be reported

Problem Description

In a few cases, internal compiler error will occur: in expand_movbi, at config/holtek/holtek.c:5501

For example:

```
typedef struct
{
   unsigned char m1 : 1;
   unsigned char m2 : 1;
} BitsMotor_t;
BitsMotor_t IO;
void MotorDriverPulse(unsigned char Step)
{
   bit m1 = 0, m2 = 0;
   switch (Step)
```

```
    {
        case 1:
          m2 = 1;
          break;
        case 2:
          m2 = 1;
          break;
        case 3:
          m1 = 1;
          break;
    }
    IO.m1 = m1;
    IO.m2 = m2;
}
```

How to avoid the problem

a. Change switch to if/else, or

b. Define the m1 and m2 as global variables or

c. Define the m1 and m2 as structure bit-field.

7.6 In a few cases, using global bit may produce incorrect assembly syntax.

Problem Description

In a few cases, incorrect assembly syntax will occur, such as the following program, where the output assembly variable _mode_oper_2[-1] will result syntax error.

For example:

```
unsigned char cnt_time_enable;
bit   way4_IO;
bit   way4_on;
extern volatile unsigned char cnt_state;
extern unsigned char cnt_time_flag;
extern unsigned char djrflag;
unsigned char float_uintcmp()
{
   return 0;
}
void mode_oper(void )
{
   way4_IO=way4_on;
   if(float_uintcmp())
   {
       way4_IO=~(way4_on);
   }
   switch(cnt_state)
   {
       case 0x10:
           cnt_time_flag=0;
           break;
       case 0x11:
           cnt_time_enable=1;
           break;
   }
}
```

```
void djr_oper(void)
{
      if(float_uintcmp())djrflag=1;
}
```

How to avoid the problem

a. Change switch to if/else, or

b. Define the way4_IO and way4_on as structure bit-field.

7.7 The function parameter with multiplication or division operation is performed incorrectly

Problem Description

When the function parameter is more than one and the parameter (not the first parameter from right to left) is with multiplication or division operation, MCUs that do not use the hardware multiplication and division operation will be performed incorrectly.

For example:

a. func(a/b,1);

b. unsinged int temp = a * b; func(temp,1);

c. func(3,a%b,1);

How to avoid this problem

Define a volatile temp variable, calculate the expression first.

For example:

volatile unsigned int temp = a * b;

func(temp,1);