TASKING C166/ST10 Tool Chain v8.6r1 patch 3 RELEASE NOTE

This patch updates the C166/ST10 v8.6r1 product.

Overview of changes:

v8.6r1 patch 3:

- Fixed register files (PR35002)
- Added DAvE XC164CM support
- Added new pragmas align32 and noalign32 (see description below)
- The #pragma align now supports D to align at double word (see description of pragma align32 below)
- Assembler EVEN directive now has additional argument (see description below)
- All of patch 2

v8.6r1 patch 2:

- Added support for XC22xx
- Allow more than 4096 registers in SFR file for CrossView Pro
- All of v8.6r1 patch 1

v8.6r1 patch 1:

- Fixed PR34691, PR34686, PR34857, PR34871, PR34603, PR34434, PR34878 (see details below)
- Added new debugger command for interrupt enabled debugging: ied
- Default OCDS interrupt level changed to 16
- Added support for XC164CM series (XC164LM,SM,TM,KM and GM): new register files and EDE support

Patch Date

2006-05-19

Components

- C Compiler (bin/c166.exe) build #759
- Assembler (bin/a166.exe) build #303
- Linker/locator (bin/l166.exe) build #260
- CrossView Pro (bin/xfw166.exe) build #360
- EDE DOIL file (etc/c166.dol) build #424
- CrossView Pro configuration files (etc/*.cfg) build #098
- Register files (etc/reg*.def, include/reg*.h) build #109
- Eval board debug instrument (bin/dieva166.dll) build #136
- Simulator (bin/disim166.dll) build #182
- Peripheral simulation (bin/psm166.dll) build #036

- Miscellaneous header files (include/*.h) build #272

Installation

To install the patch run the setup.exe

New debugger command: ied

The new ied command controls the use of Interrupt Enabled Debugging:

- "1 ied" enables
- "0 ied" disables
- "ied" shows the mode

By default Interrupt Enabled Debugging is disabled. When it is enabled, the debugger will disable interrupts temporarily during a single step to avoid landing in the interrupt routine of pending interrupts.

Default OCDS Interrupt Level

The default value of the CMCTR register is changed from 0x3000 to 0x8000. This means that the default OCDS interrupt level is now set 16. The CMCTR register can be modified from the Project Options dialog -> Application -> Startup -> CMCTR

New C Compiler Pragmas to Control 32-bit Alignment

The C compiler supports four new pragmas to change the alignment of structs, unions and longs. With the -z option these pragmas can be supplied on the command line, for example: -zpragma32. The existing #pragma align now additionally supports an align type D, which effectively does the same as the #pragma align32, but can then be specified per memory type.

#pragma align32
#pragma noalign32

The #pragma align32 changes the alignment as follows:

- align long data types on 32-bit boundaries (not for stack objects)
- align struct/union data types on 32-bit boundaries (not for stack objects)
- the size of a struct/union will always be a multiple of 4 bytes
- The _packed keyword overrules the command line option

struct/union members:

- members of type long will be 32-bit aligned
- members of type bit-field (base: int) will be 32-bit aligned
- packing of bit-field members will still be done according to the base type (int)

Example of bit-field member packing:

```
struct s
{                          /* offset       */
```

```
            int bf1 : 15;   /*  0 bits      */
            int bf2 : 2;    /* 32 bits      */
            int bf3 : 2;    /* 34 bits      */
            int     : 12;
            int bf4 : 5;    /* 64 bits      */
};
```

In words: a bit-field will be aligned to the next 32-bit boundary if the bit-field crosses a 16-boundary, or if the bit-field is located at a 16-bit boundary.
A bit-field of size 0 will align any struct member to the next 32-bit boundary.

The #pragma noaling32 switches back to the default alignment as described in the product manuals

#pragma suspend_align32
#pragma resume_align32

Whith these pragmas the #pragma align32 can be temporarily suspended/resumed.

Optional Argument for EVEN Directive

To support the pragma align32 as described above, the assembler's EVEN directive is extended with an optional argument to specify the alignment in words:

```
EVEN    [num]
```

For example "EVEN 2" aligns at 32 bit. This alignment is relative to the section begin. For correct alignment it is also required to specify the corresponding aligntype on the SECTION directive or to locate the section absolute at an aligned start address. For example to align at 32 bit the section align type must be DWORD.

Solved problems

SOLVED PR34691: Smart linking: I 900: internal error l166(../../link2.c,590): symbol table bad

Linking with smart linking enabled results in the error:

```
I 900: internal error l166(../../link2.c,590): symbol table bad
```

PR34686: Application is halted inside an interrupt routine after single step using OCDS

When a single source line step is executed, CrossView first looks how many instructions are involved. If the amount of instructions is below a certain amount, a corresponding amount of single step instructions is executed by CrossView instead of setting a breakpoint at the next source line.

When an interrupt occurs during execution of these single step instructions, the application unexpectedly ends up somewhere inside the interrupt routine.

Solved by adding new command: Interrupt enabled debugging

PR34857: Wrong code for BUSCONx initialization when CPU.21 bypass is enabled

When CPU.21 bypass is enabled wrong code is generated for BUSCONx registers in the startup code. For example, using C167CS:

```
AND BUSCON0, #(~0xDFFF)|0
OR  BUSCON0, #((0x0000&~0)&0xDFFF)
AND BUSCON1, #~0xDFFF
OR  BUSCON1, #(0xDFFF&0x0000)
```

which corresponds to:

```
AND BUSCON0,#0x2000
OR  BUSCON0,#0x0
AND BUSCON1,#0x2000
OR  BUSCON1,#0x0
```

As a result of the AND instruction, BUSCONx is almost completely set to zero and the application will crash.

PR34871: DAS server is not always started

In some cases the DAS server is not always started when connecting the target board.

To solve this EDE generates cfg files with the TerminateServer entry inside the cfg file by changing the value "1" into "0".

PR34603: FILLGAP control overwrites constant value

When building the example, the hex file contains the following information:

```
:011FFE00AA38
:011FFF00558C
:011FFF00449D
```

The third line shows that address 0x01FFF is overwritten with a fill byte 0x44.

Example

```
const unsigned char low_u8 _at(0x1FFE)=  0xaa;
const unsigned char high_u8 _at(0x1FFF)= 0x55;
```

Invocation:

```
cc166 test.c -ihex novt MEMORY(ROM(1FFEH-1FFFH FILLGAPS(0x44)))
```

PR34434: C166_US secsize control ignored when C166_US is truncated to 16384 bytes

When the total C166_US section is exceeding 16Kbyte, it is truncated by the linker/locator to 16384 bytes automatically. However, in this case any C166_US secsize control will be ignored completely.

Example

main.c:

```
extern int foo(void);

int main(void)
{
 char big[1024*16];
 return foo()+big[2];
}
```

foo.c:

```
int foo(void)
{
 char small[1024];
 return small[1];
}
```

Linker/locator secsize control: SECSIZE(C166_US(2000h))

```
W 514: module start.obj (CSTART): userstack section C166_US is truncated to
       16384 bytes
W 514: module foo.obj (FOO_C): userstack section C166_US is truncated to 16384
       bytes
```

Result: The length of C166_US is 4000h bytes instead of 2000h.

PR34878: Wrong segment address when constant is casted to pointer in large memory model

When using a #define to cast a constant to a pointer in the large memory model, the wrong segment is calculated by the compiler for this pointer:

```
 MOV R14,#05678h
 MOV R15,#01234h
 EXTP #PAG 0200300h,#01h
 MOV POF 0200300h,R14
 EXTP #PAG 0800302h,#01h  ;segment 0x80 is being used here instead of 0x200
 MOV POF 0800302h,R15
```

Example code

```
#define u32 unsigned long

struct  my_st
{
  u32  var_long;
};

#define my_pointer ((struct my_st volatile far *) 0x200300)

void main(void)
{
  my_pointer[0].var_long =  (u32)0x12345678;
}
```

PR34648: XC167CI register files are missing bits AN8..AN11 and P5_8..P5_11

XC167CI register files are missing bits AN8 to AN11 and the corresponding port 5 definitions: P5_8, P5_9, P5_10, P5_11 and P5D_8, P5D_9, P5D_10, P5D_11 and T5EUD, T6EUD.

Note: this also applies to various other register files

PR35002:

Several registers in v8.6r1 patch 1 have a mismatch in the register definition in the .def file and the .h file. When compiling C files that use these registers it can result in errors from the assembler like:

```
'E 292: illegal operand combination'
```

The involved registers are:
RSTCON2
RSTCON
SYSSTAT
CC2_ID
GPT12E_ID
IDDMP1
IDDMPM
PSCSTAT
RTC_ID

Note this problem is introduced in patch 1
