56F800EX new additional 32bit multiplication instruction test

Target on MC56F84xxx DSC

Moris Hsu
Field Application Engineer

Feb/25/2013

Requirement

- How to use new 56F800EX additional 32bit multiplication instructions in CodeWarrior IDE?

56F800EX Core

- Core 56F800EX is a version 3 of core 56F800E.
- New Additional 32-Bit DSP56800EX Multiplication Instructions supported

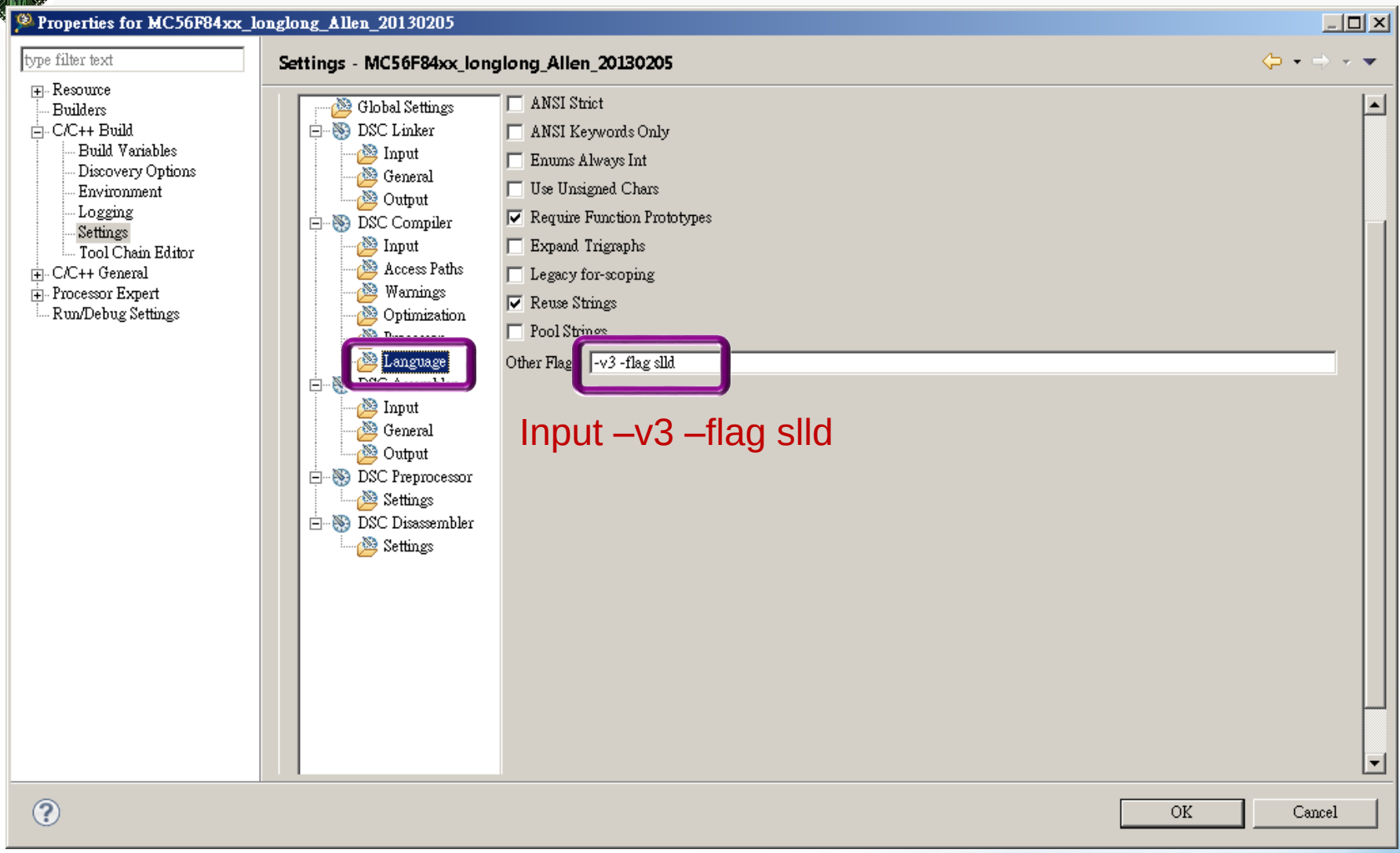
Instruction	Parallel Move?	Description
IMAC32	—	Integer multiply-accumulate 32 bits
IMPY32	—	Integer multiply 32 bits x 32 bits → 32 bits
IMPY64	—	Integer multiply 32 bits x 32 bits → 64 bits
IMPY64UU	—	Unsigned integer multiply 32bits x 32 bits → 64 bits
MAC32	—	Fractional multiply-accumulate 32 bits x 32 bits → 32 bits
MPY32	—	Fractional multiply 32 bits x 32 bits → 32 bits
MPY64	—	Fractional multiply 32 bits x 32 bits → 64 bits

Test Environment

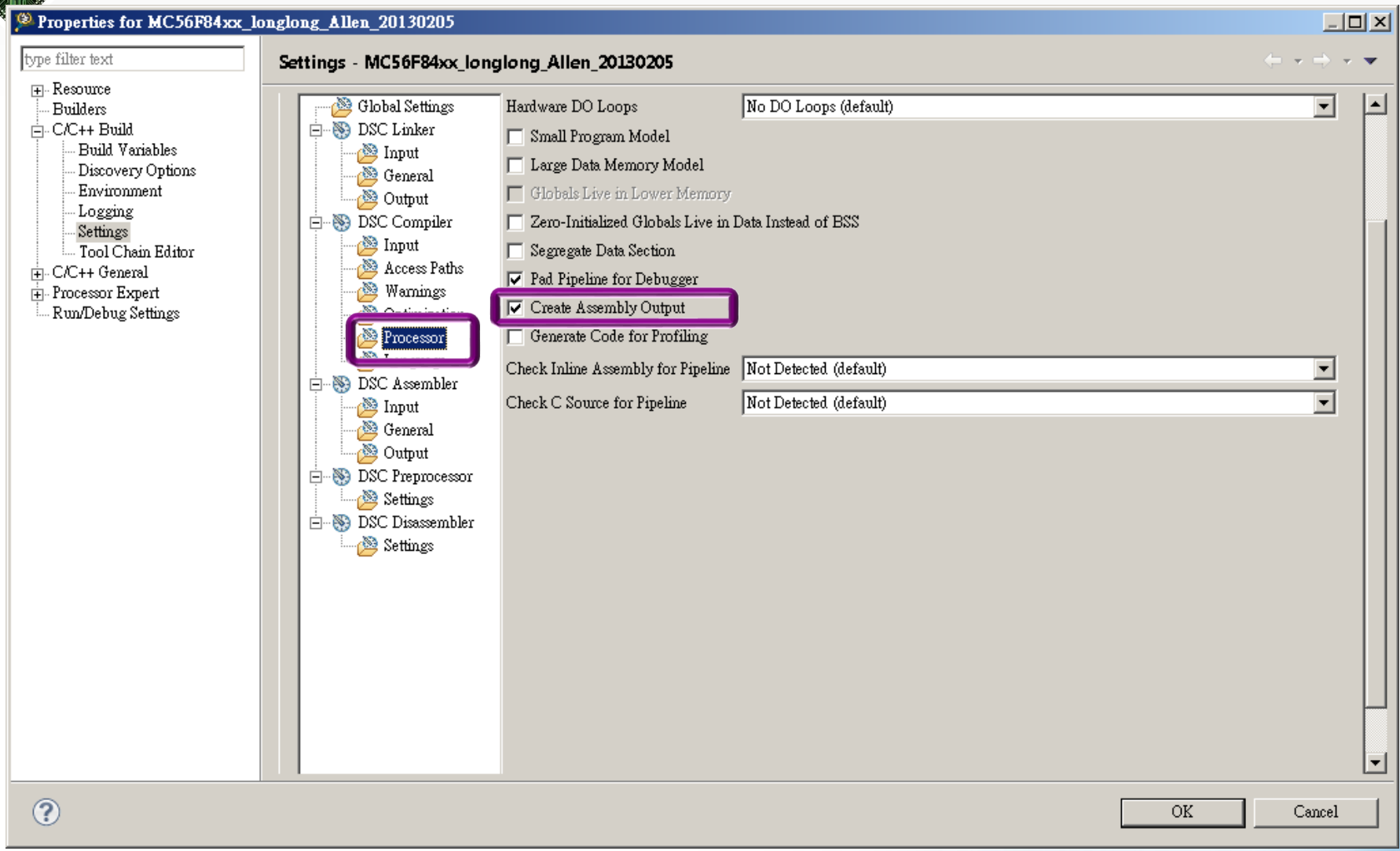
- Hardware:
  - TWR-56F8400 (MC56F84789 on board)
- Software:
  - CW for MCU V10.3
  - CW for MCU V10.5 / V10.6 (DSC compiler supports 56800EX new additional instructions natively)
  - Location of Long Long process source code: “c:\Freescale\CW for MCU V10.3\MCU\M56800E Support\runtime_56800E\ll\”

CW V10.3 Configuration

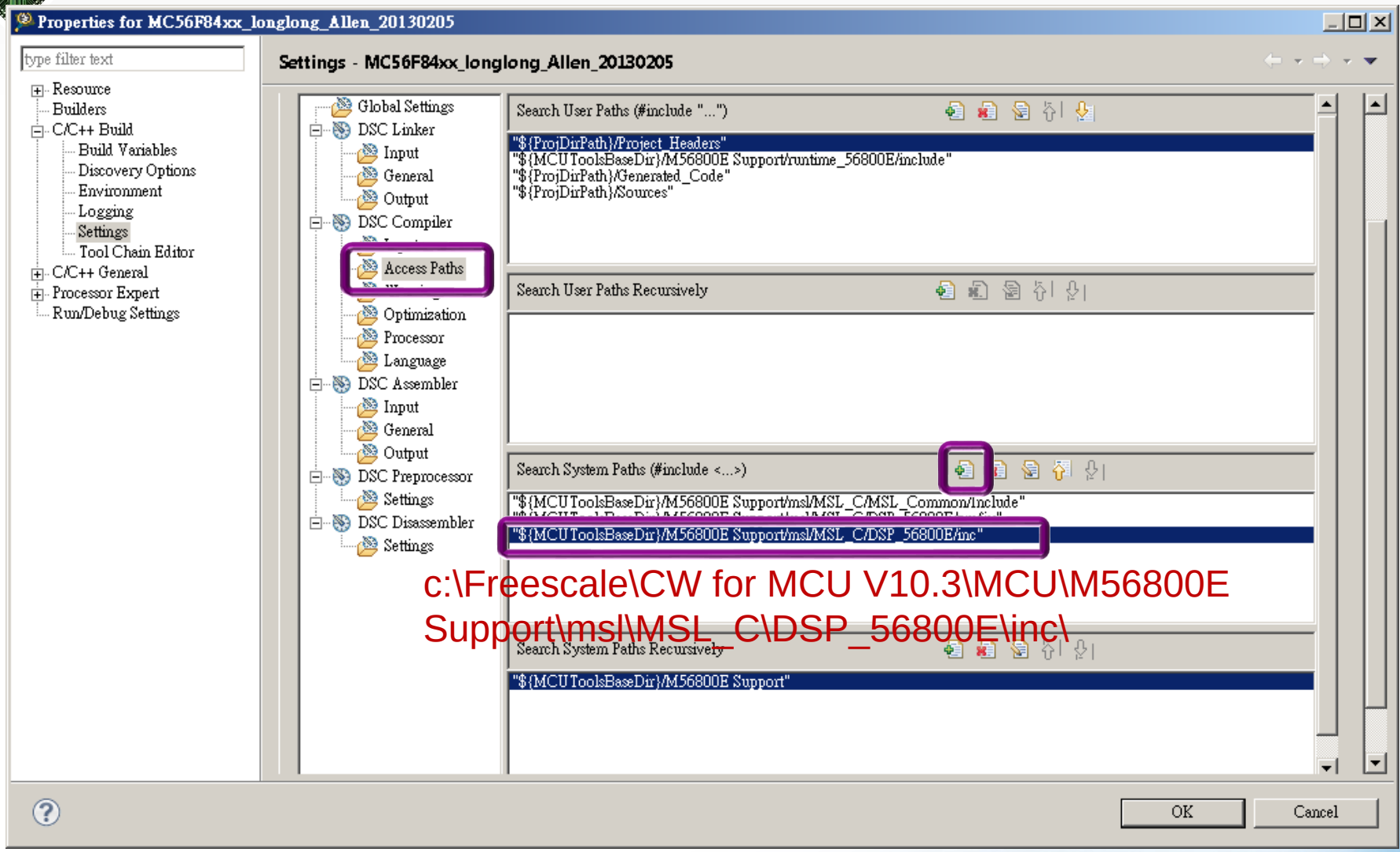
Compiler Setting



Compiler Setting

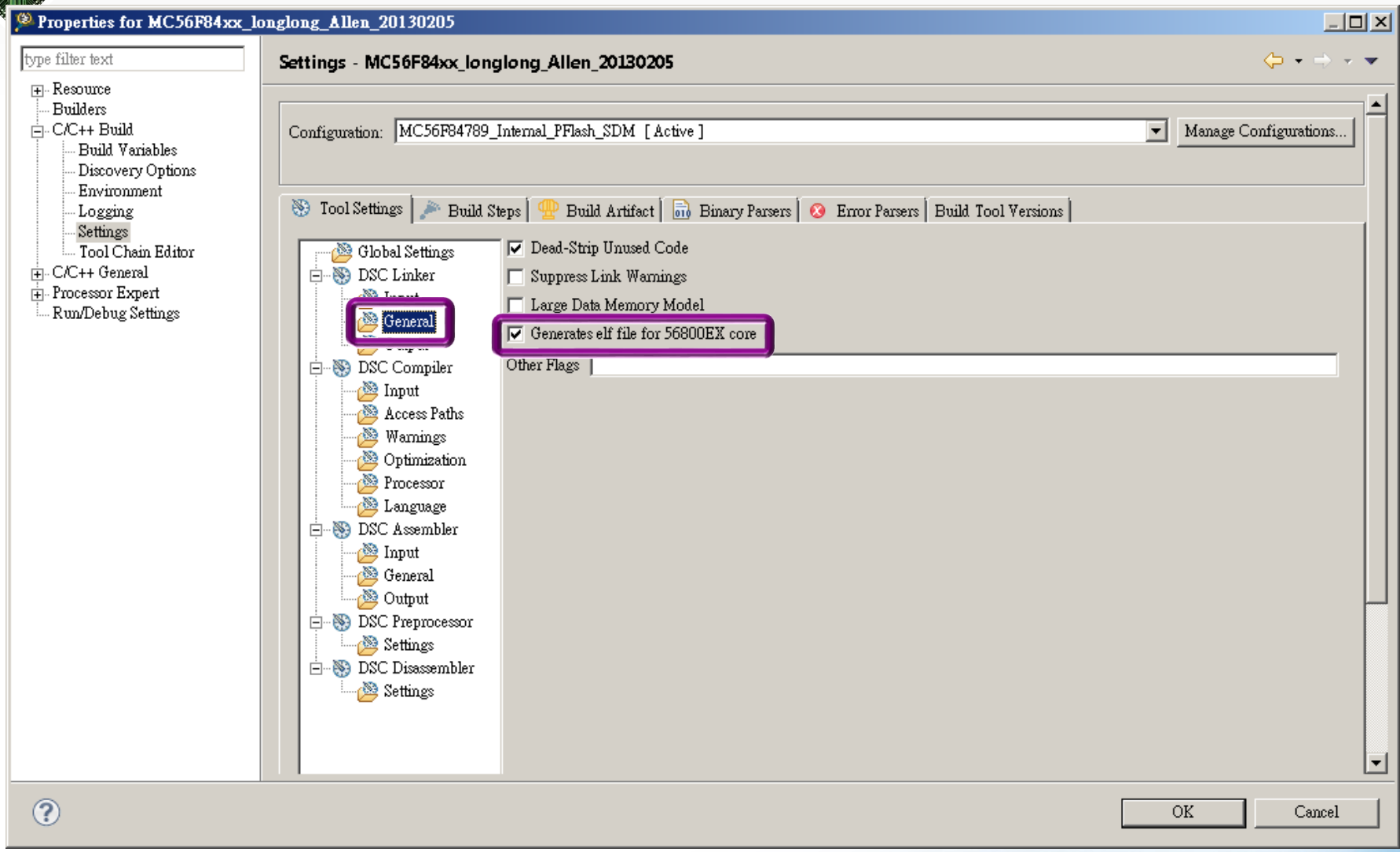


Compiler Setting

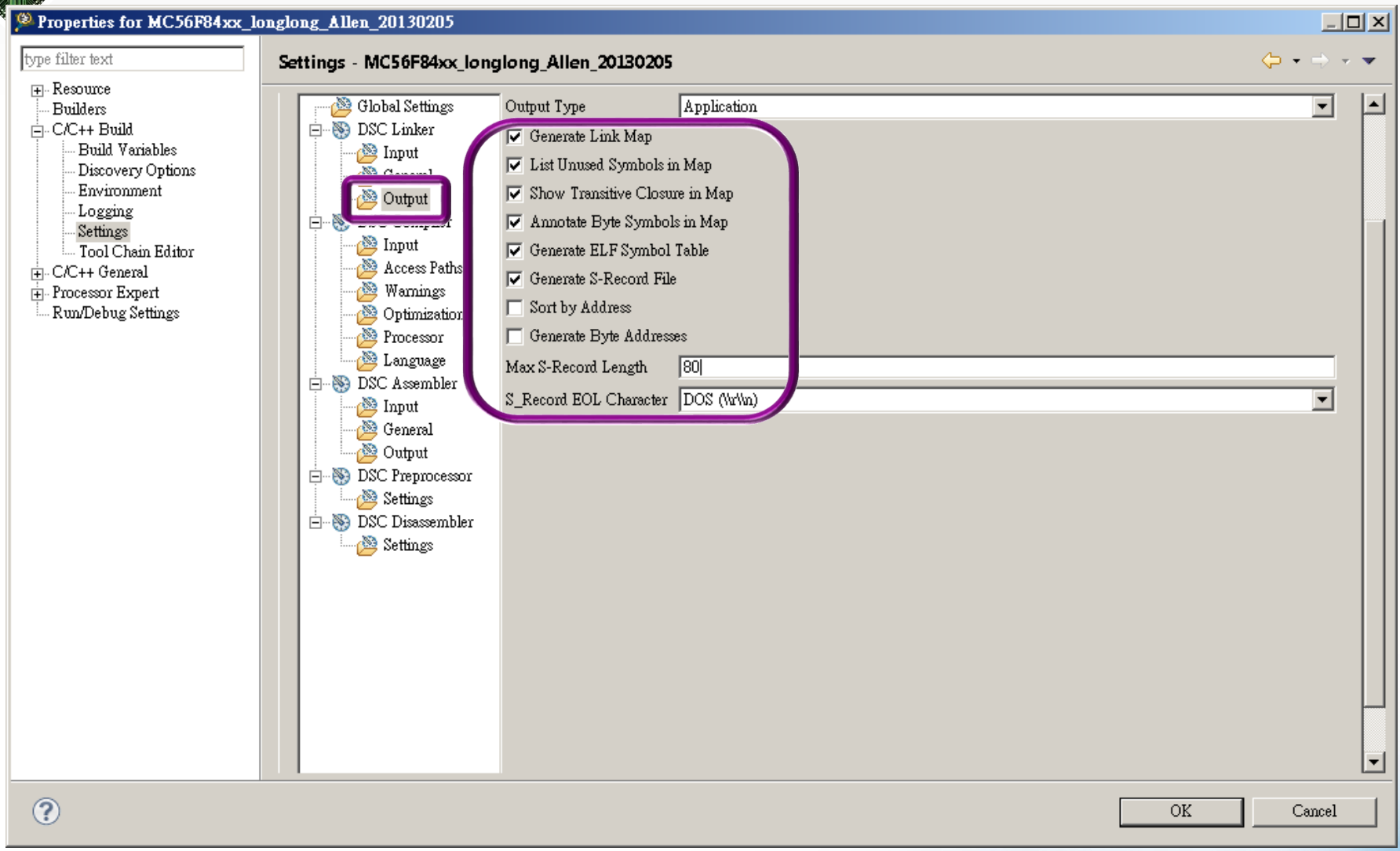


c:\Freescale\CW for MCU V10.3\MCU\M56800E Support\msl\MSL_C\DSP_56800E\inc\

Linker Setting



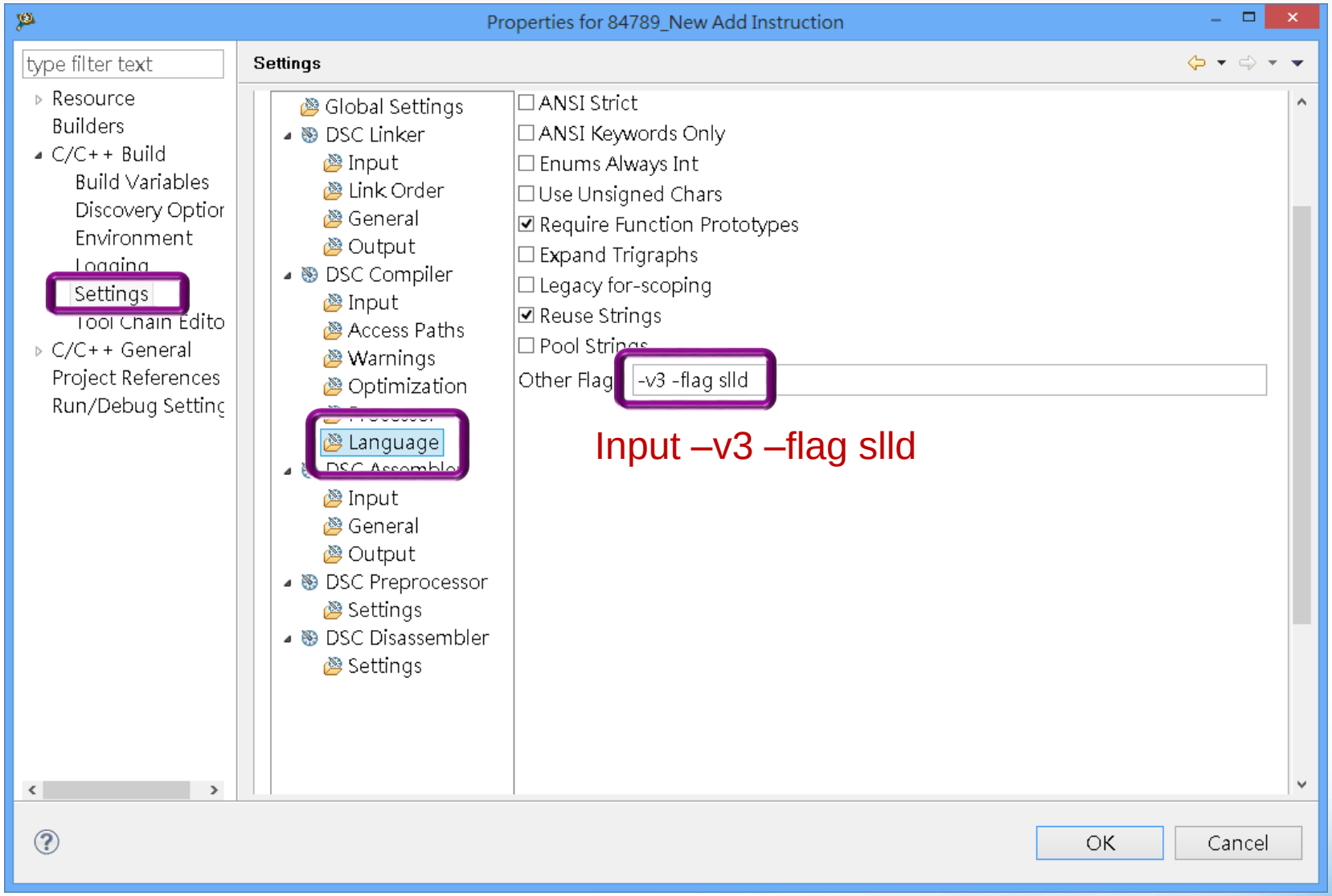
Linker Setting



CW V10.5 / V10.6 Configuration

Updated on Jun/2014

Compiler Setting



Compiled C Code in CW V10.3

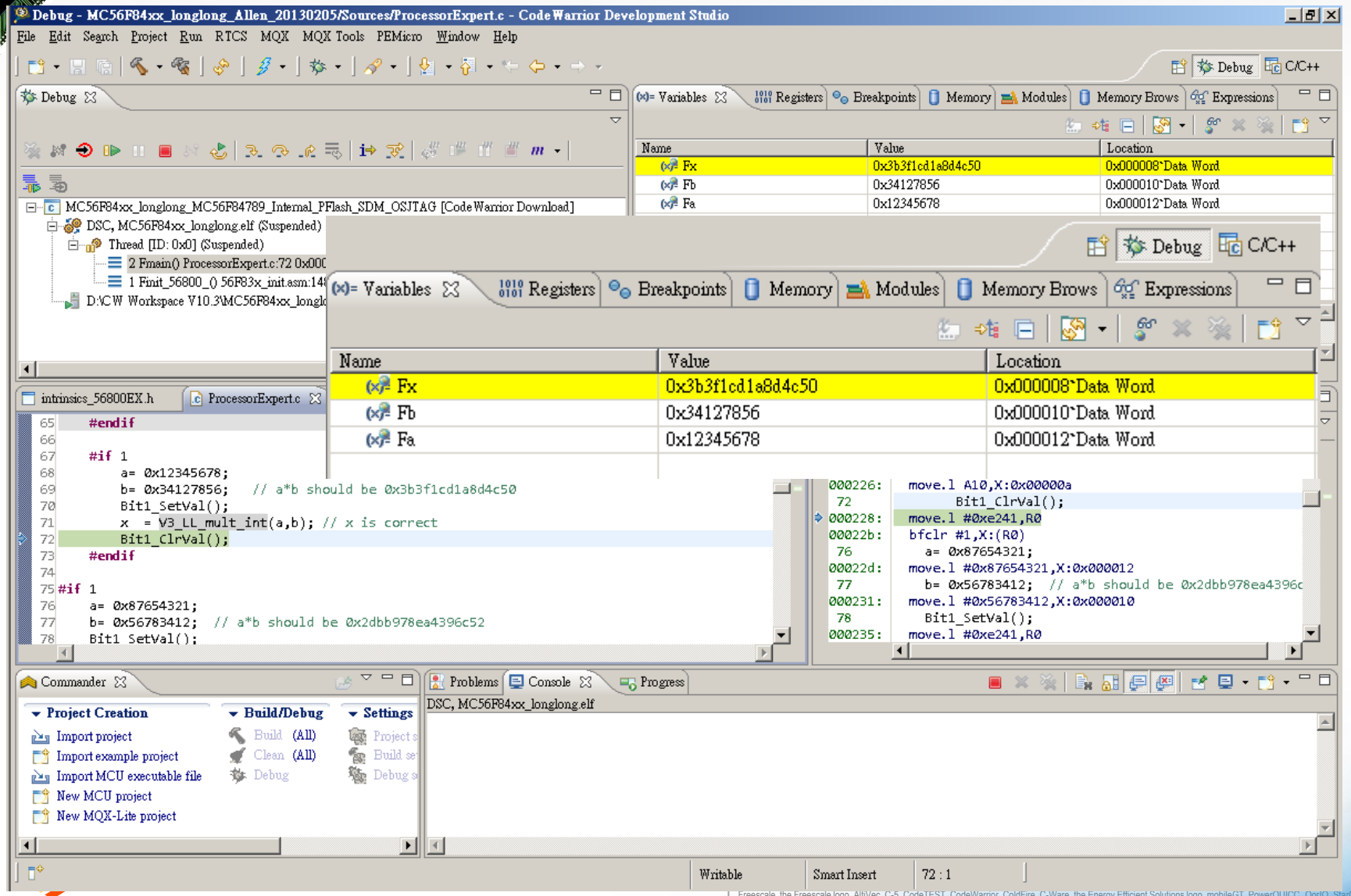
32bit*32bit= 64bit with V3 function call, success

- unsigned long a, b;
- long long x = 0;

- a= 0x12345678;
- b= 0x34127856;
- Bit1_SetVal();
- x = V3_LL_mult_int(a,b);
- Bit1_ClrVal();

```
adda      #<8,SP
move.l    #$12345678,X:>Fa   // a= 0x12345678;
move.l    #$34127856,X:>Fb   // b= 0x34127856;

move.l    #$e241,R0          // Bit1_SetVal();
bfset     #1,X:(R0)

move.l    X:>Fb,B            // x  = V3_LL_mult_int(a,b);
move.l    X:>Fa,A
jsr       >FV3ARTIMPY64
move.l    B10,X:(SP)
move.l    A10,X:(SP-2)
move.l    X:(SP),A
move.l    X:(SP-2),B
move.l    A10,X:(SP-4)
move.l    B10,X:(SP-6)
move.l    X:(SP-4),A
move.l    X:(SP-6),B
move.l    B10,X:>Fx
move.l    A10,X:>Fx+2

move.l    #$e241,R0          // Bit1_ClrVal();
bfclr     #1,X:(R0)
```

32bit*32bit= 64bit with V3 function call, success



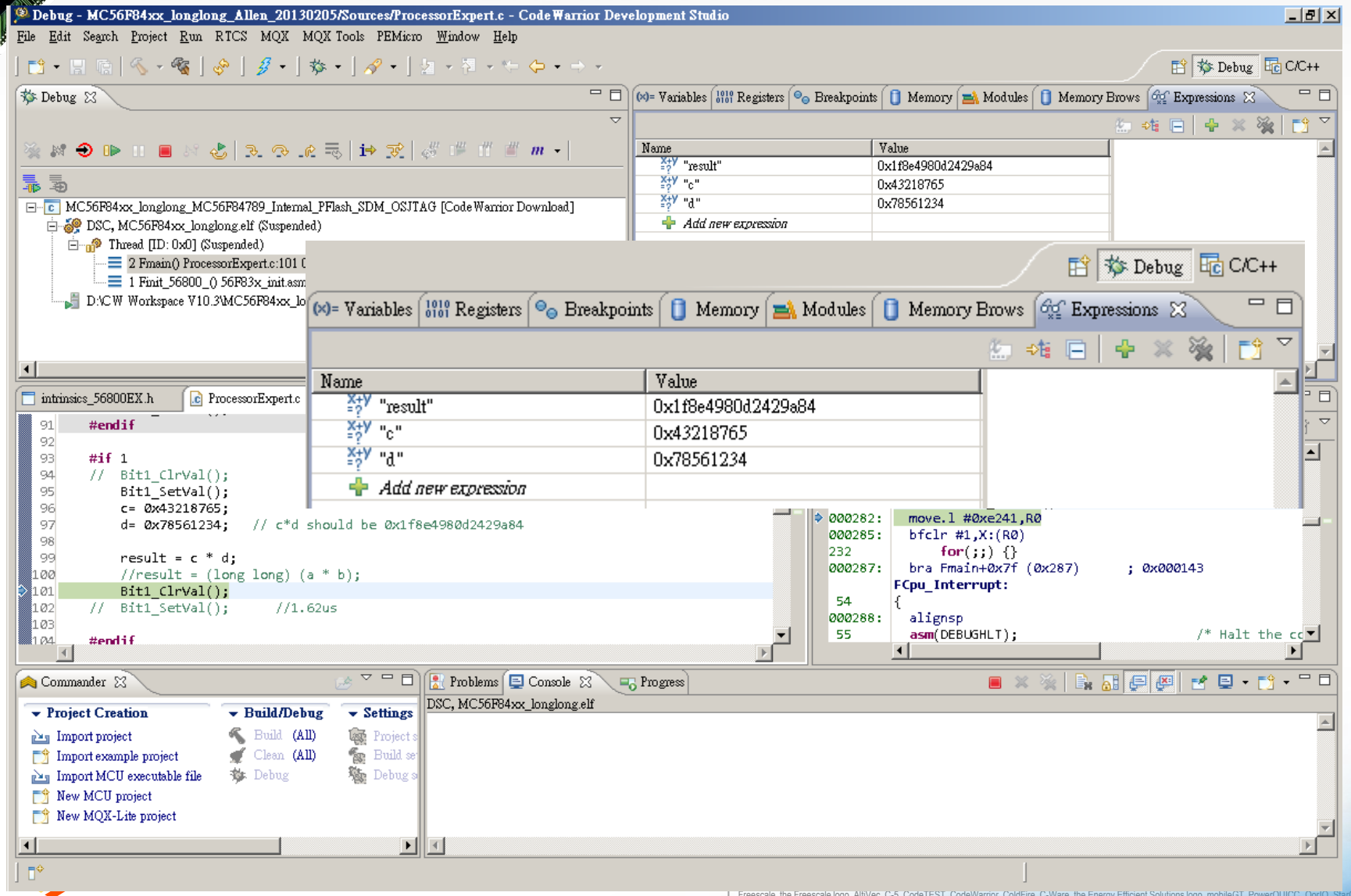
64bit*64bit= 64bit, success

- long long c=0, d= 0, result = 0;

- Bit1_SetVal();
- c= 0x43218765;
- d= 0x78561234;
- result = c * d;
- Bit1_ClrVal();

```
move.l   #$e241,R0        // Bit1_SetVal();
bfset    #1,X:(R0)

move.l   #$43218765,B     // c= 0x43218765;
move.l   #>>0,A           //
move.l   B10,X:>Fc        //
move.l   A10,X:>Fc+2      //
move.l   #$78561234,B     // d= 0x78561234;
move.l   #>>0,A           //
move.l   B10,X:>Fd        //
move.l   A10,X:>Fd+2      //
adda     #<8,SP
move.l   X:>Fc,B
move.l   X:>Fc+2,A
move.l   B10,X:(SP-2)
move.l   A10,X:(SP)
move.l   X:>Fd,B
move.l   X:>Fd+2,A
move.l   B10,X:(SP-6)
move.l   A10,X:(SP-4)
jsr      >ARTMULLL64
suba     #<8,SP
move.l   A10,X:>Fresult   // result = c * d
move.l   B10,X:>Fresult+2 //

move.l   #$e241,R0        // Bit1_ClrVal();
bfclr    #1,X:(R0)
```

64bit*64bit= 64bit, success



32bit*32bit= 64bit, not success

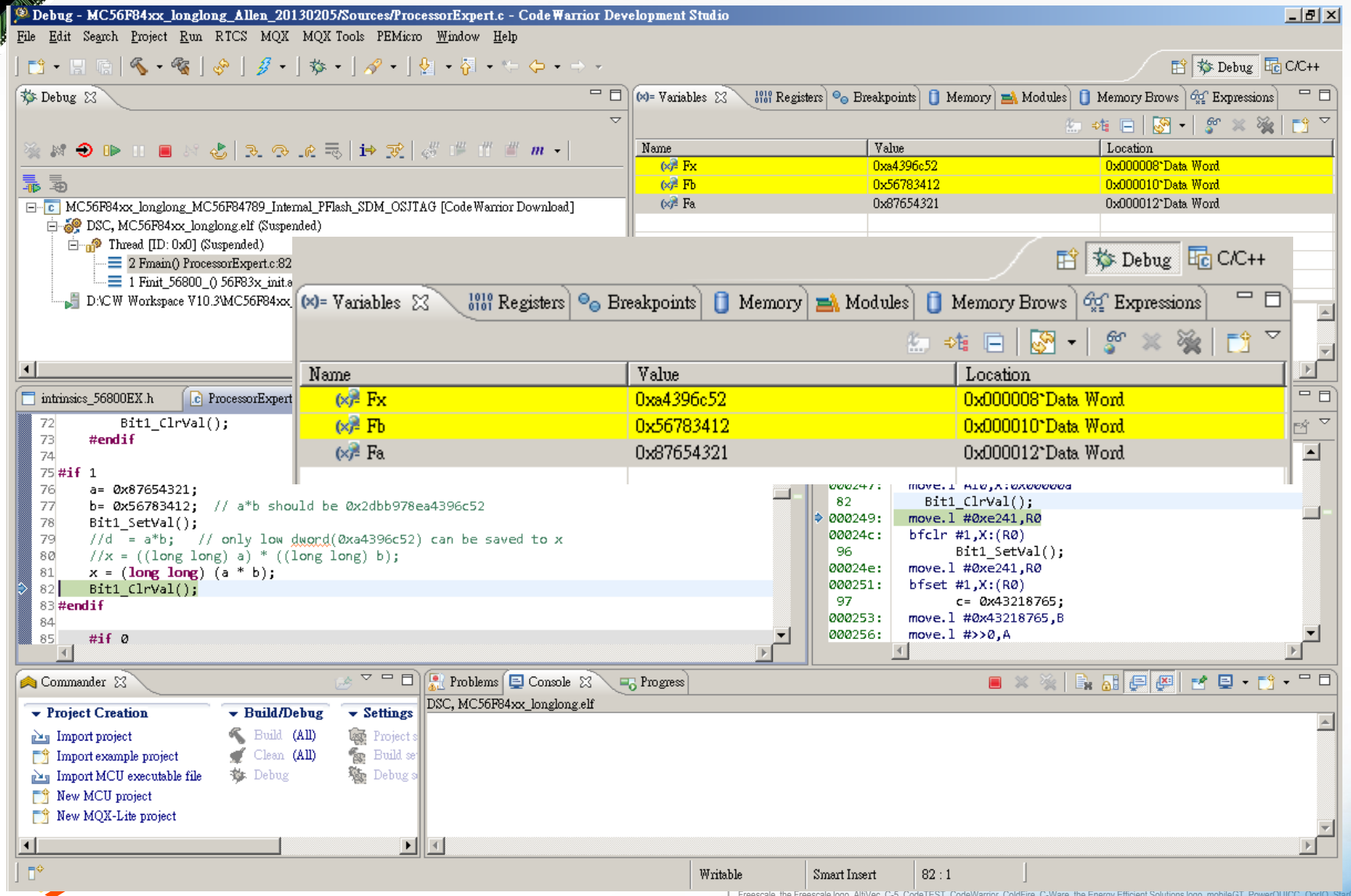
- unsigned long a, b;
- long long x = 0;

- a= 0x87654321;
- b= 0x56783412;
- Bit1_SetVal();
- x = (long long) (a * b);
- Bit1_ClrVal();

Only low 32bit result can be saved!

```
move.l    #$87654321,X:>Fa   // a= 0x87654321;
move.l    #$56783412,X:>Fb   // b= 0x56783412;

move.l    #$e241,R0          // Bit1_SetVal();
bfset     #1,X:(R0)

move.l    X:>Fa,B
move.l    X:>Fb,A
impyuu    A1,B0,Y
imacuu    A0,B1,Y
impyuu    A0,B0,B
add       Y0,B
move.l    #>>0,A
move.l    B10,X:>Fx
move.l    A10,X:>Fx+2

move.l    #$e241,R0          // Bit1_ClrVal();
bfclr     #1,X:(R0)
```

32bit*32bit= 64bit, not success



32bit*32bit= 64bit, success

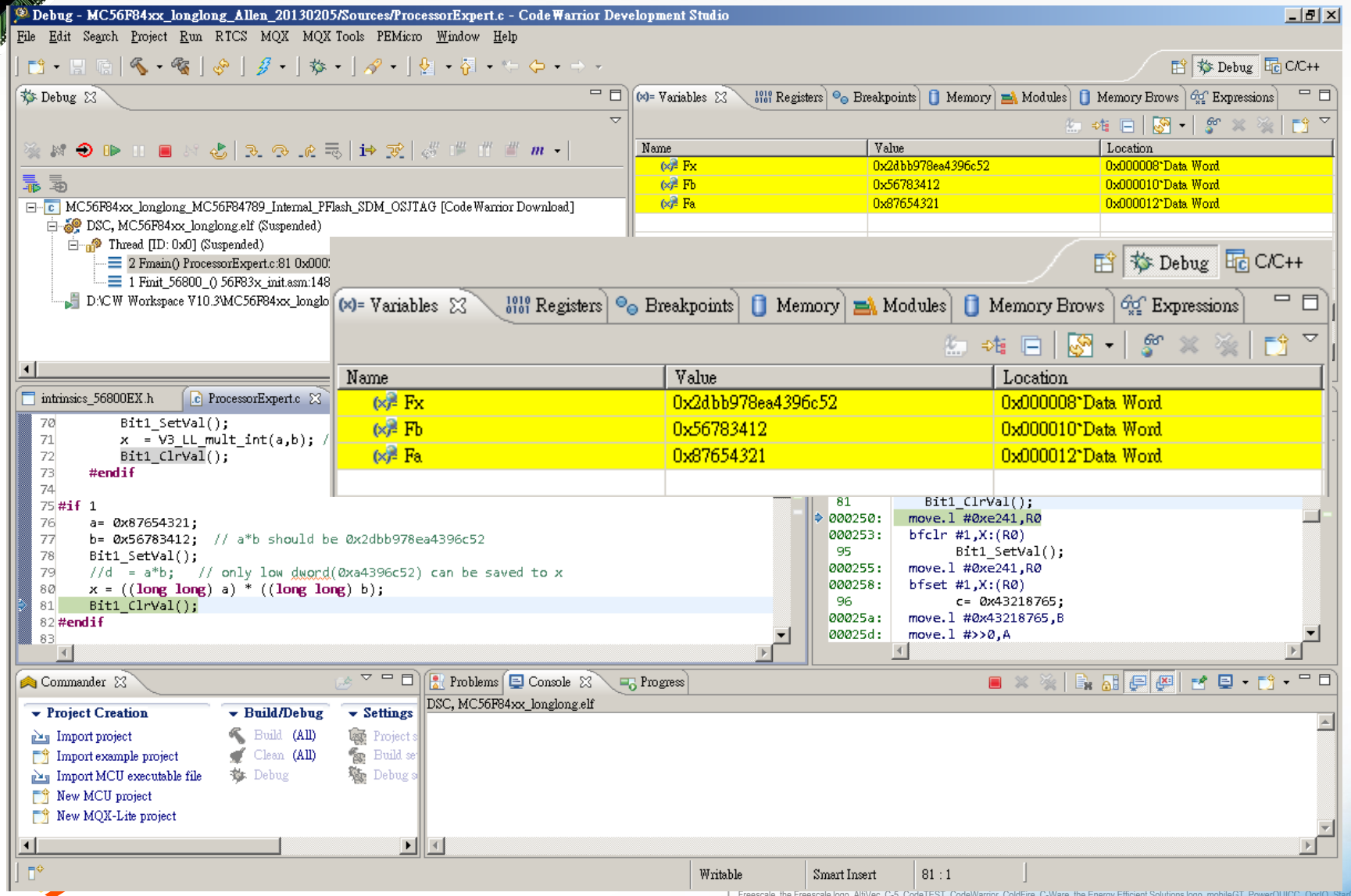
- unsigned long a, b;
- long long x = 0;

- a= 0x87654321;
- b= 0x56783412;
- Bit1_SetVal();
- x = ((long long) a) * ((long long) b);
- Bit1_ClrVal();

Compare with 64bit*64bit=64bit, this will save some instructions, due to there is no variable initiation actions.

```
move.l   #$87654321,X:>Fa   // a= 0x87654321;
move.l   #$56783412,X:>Fb   // b= 0x56783412;

move.l   #$e241,R0          // Bit1_SetVal();
bfset    #1,X:(R0)

adda     #<8,SP
move.l   X:>Fa,B
move.l   #>>0,A
move.l   B10,X:(SP-2)
move.l   A10,X:(SP)
move.l   X:>Fb,B
move.l   #>>0,A
move.l   B10,X:(SP-6)
move.l   A10,X:(SP-4)
jsr      >ARTMULLL64
suba     #<8,SP
move.l   A10,X:>Fx
move.l   B10,X:>Fx+2        // x = ((long long) a) * ((long long) b);

move.l   #$e241,R0          // Bit1_ClrVal();
bfclr    #1,X:(R0)
```

32bit*32bit= 64bit, success



Compiled C Code in CW V10.5 / V10.6

Updated on Jun/2014

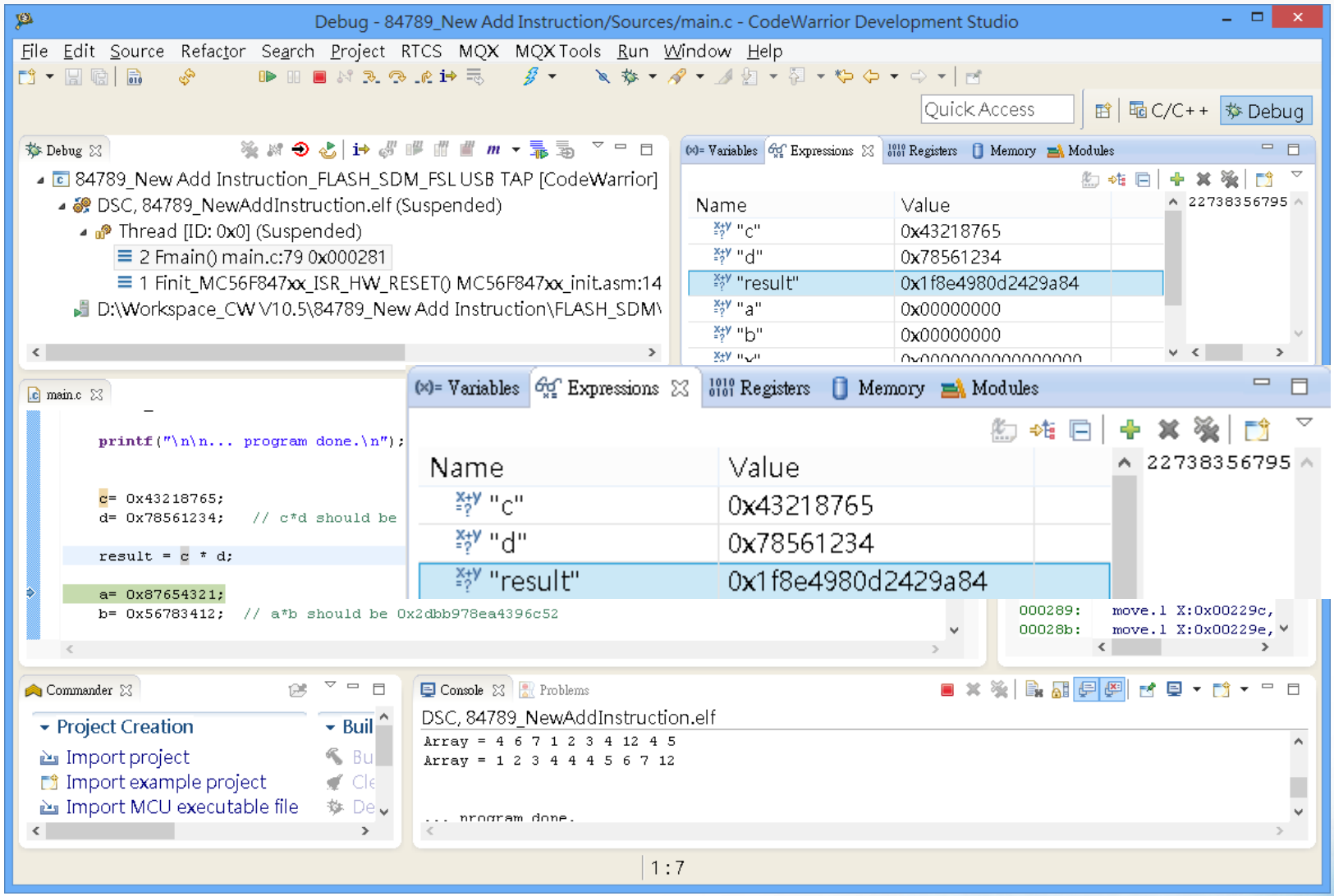
Signed 32bit*32bit= 64bit, success

- **long c=0, d= 0;**
- **long long result= 0;**

- c= 0x43218765;
- d= 0x78561234;
- result = **c * d**;

```
move.l   #0x43218765,X:0x000000
move.l   #0x78561234,X:0x000000
move.l   X:0x000000,B          ; Fc
move.l   X:0x000000,A          ; Fd
impy64   A,B,A
move.l   Y,X:0x000000          ; Fresult
move.l   A10,X:0x000000        ; Fresult+2
```

Signed 32bit*32bit= 64bit, success



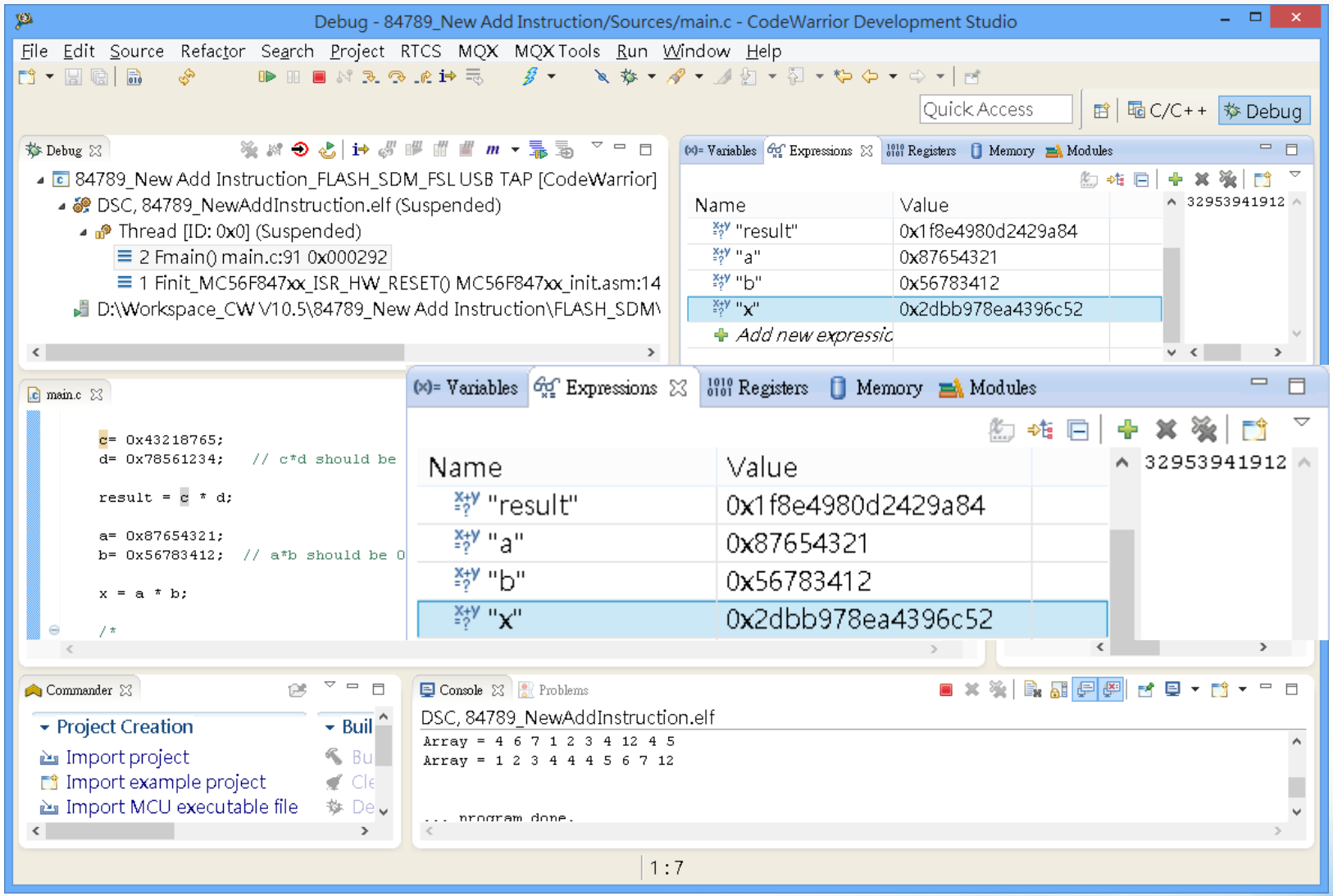
Unsigned 32bit*32bit= 64bit, success

- **unsigned long a=0, b= 0;**
- **long long x= 0;**

- a= 0x87654321;
- b= 0x56783412;
- x = **a * b**;

```
move.l    #0x87654321,X:0x000000
move.l    #0x56783412,X:0x000000
move.l    X:0x000000,B          ; Fa
move.l    X:0x000000,A          ; Fb
impy64uu  A,B,A
move.l    Y,X:0x000000          ; Fx
move.l    A10,X:0x000000        ; Fx+2
```

Unsigned 32bit*32bit= 64bit, success



Long Long(LL) Calculation Function Call

File Location: “c:\Freescale\CW for MCU V10.3\MCU\M56800E Support\runtime_56800E\ll\ARTMULLL64.asm”

V3ARTIMPY64 source code – New Multiplication

```
;=============================================================================
; FUNCTION: V3ARTIMPY64
; DESCRIPTION:          56800EX long long multiplication.(56800EX instructions intrinsic functions support)
;                       Multiply two 32-bit integer values generating a signed 64-bit integer result.
;                       56800EX instruction:        IMPY64 FF1,FF1,FF:Y - Integer Multiply 32x32 = 64 bit; save higher (in Acc), lower (in Y)
;
V3ARTIMPY64:
FV3ARTIMPY64:
            adda #<2,SP
            move.l C10,X:(SP)

            impy64 A,B,C

            adda #<4,SP
            move.l C10,X:(SP)
            move.l Y,X:(SP-2)
            move.l X:(SP),B
            move.l X:(SP-2),A
            suba #4,SP

            move.l X:(SP)-,C
            rts
end_FV3ARTIMPY64:
```

ARTMULLL64 source code – Old Multiplication

```
;=====================================
;=====================================
; FUNCTION: ARTMULLL64
; DESCRIPTION: Long long multiplication.
;
; INPUT: a = x, b = y
; OUTPUT: a = result = x * y
;
ARTMULLL64:
FARTMULLL64:
;ARTMULULL64:
;FARTMULULL64:

    adda    #<2,SP
    move.l  C10,X:(SP)+
    move.l  D10,X:(SP)

    adda    #<14,SP
    move.l  X:(SP-22),A     ; 1
    ;move.l  X:(SP-20),C    ; 0
    move.l  X:(SP-26),B     ; 3
    ;move.l  X:(SP-24),D    ; 2
    impyuu A0,B0,D
    impyuu A1,B0,Y
    move.w B1,B0
    impyuu A0,B0,C
    add C,Y
    move.w Y1,B0
    move.w #$0,B1
    jcc L1
    move.w #$1,B1
L1:
    move.w #$1,X:(SP-2)     ; to keep carry flag - clear up part of D
    tfr Y0,Y
    add D,Y
    tfr Y,D
    jcs L2
    move.w #$0,X:(SP-2)     ; to keep carry flag - set up part of D  - used in the end test
```

ARTMULLL64 source code – Old Multiplication(cont'd)

```
L2:
   move.l D10,X:(SP-4)     ; d5
   move.l B10,X:(SP-8)     ; d14
   move.l X:(SP-22),A      ; 1
   move.l X:(SP-24),B      ; 2
   impyuu A0,B0,C  ; d10
   move.l X:(SP-20),A      ; 0
   move.l X:(SP-26),B      ; 3
   impyuu B0,A0,D  ; d15
   add C,D ; d15
   tfr D,Y
   move.l X:(SP-22),A      ; 1
   move.l X:(SP-26),B      ; 3
   move.w A1,A0
   imacuu A0,B1,Y
   move.l X:(SP-8),C
   add C,Y ; d15
   move.l Y,X:(SP-8)
   move.l X:(SP-22),B          ; 1
   move.l X:(SP-24),A          ; 2
   impysu A1,B0,Y
   imacuu A0,B1,Y
   move.l X:(SP-20),B      ; 0
   move.l X:(SP-26),A      ; 3
   tfr A,C
   imacuu B0,C1,Y
   move.l X:(SP-26),A      ; 3
   imacus A0,B1,Y
   asll.l #$10,Y
   move.l X:(SP-8),D
   bfclr #$01,SR
   add D,Y ; Y = d4
   move.l X:(SP-4),A
   move.w X:(SP-2),B0
```

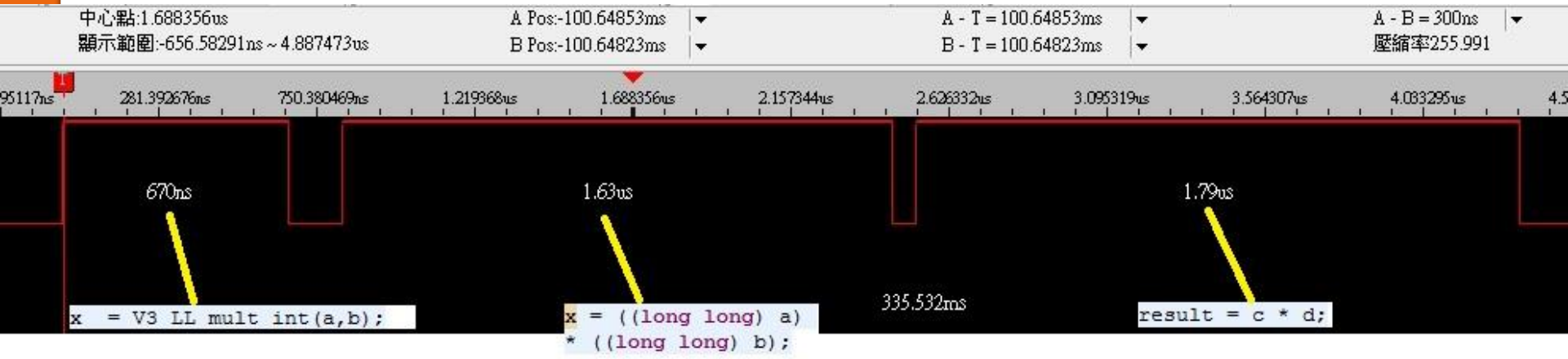
ARTMULLL64 source code – Old Multiplication(cont'd)

```
    bftsth #$0001,B0
    jcc L3
    move.l #$1,C
    add C,Y
L3:
    tfr Y,B
    sxt.l B
    suba    #<14,SP

    move.l  X:(SP)-,D
    move.l  X:(SP)-,C

    rts

end_FARTMULLL64:
;end_FARTMULULL64:
```

Execution time in CW V10.3

Execution time

- It takes 670ns to run “FV3ARTIMPY64” for 32bit*32bit= 64bit.
- It takes 1.63us to run “ARTMULLL64” for 32bit*32bit= 64bit.
- It takes 1.79us to run “ARTMULLL64” for 64bit*64bit= 64bit.



Conclusion

Conclusion

- To use new version of LL calculation function call, refer to “C:\Freescale\CW for MCU V10.3\MCU\M56800E Support\msl\MSL_C\DSP_56800E\inc\intrinsics_56800EX.h”
  - The multiplication calculation which uses new additional instruction does more simple than the old one
- To achieve 64bit result calculation, the multiplicand and multiplier must be 64bit if no V3 LL calculation function call be used.
  - Compiler & Linker options must be configured first.
- If data type of multiplicand and multiplier are not 64bit, data type conversion is required.
  - Note: found this C code can save some instructions comparing with 64bit*64bit= 64bit, due to there is no variable initiation.
- The DSC compiler in CW V10.5 and V10.6 does support 56800EX new additional instructions natively. (Updated on Jun/2014)


freescale™