

**Интегрированная среда разработки
и отладки программного
обеспечения
Code Composer Studio**

**Создание программ
Основные инструментальные средства
Язык Ассемблера
Поддержка программирования на C/C++**

Code Composer Studio™ IDE

PLATINUM EDITION



Code Composer Studio Development Tools v3.3

Getting Started Guide

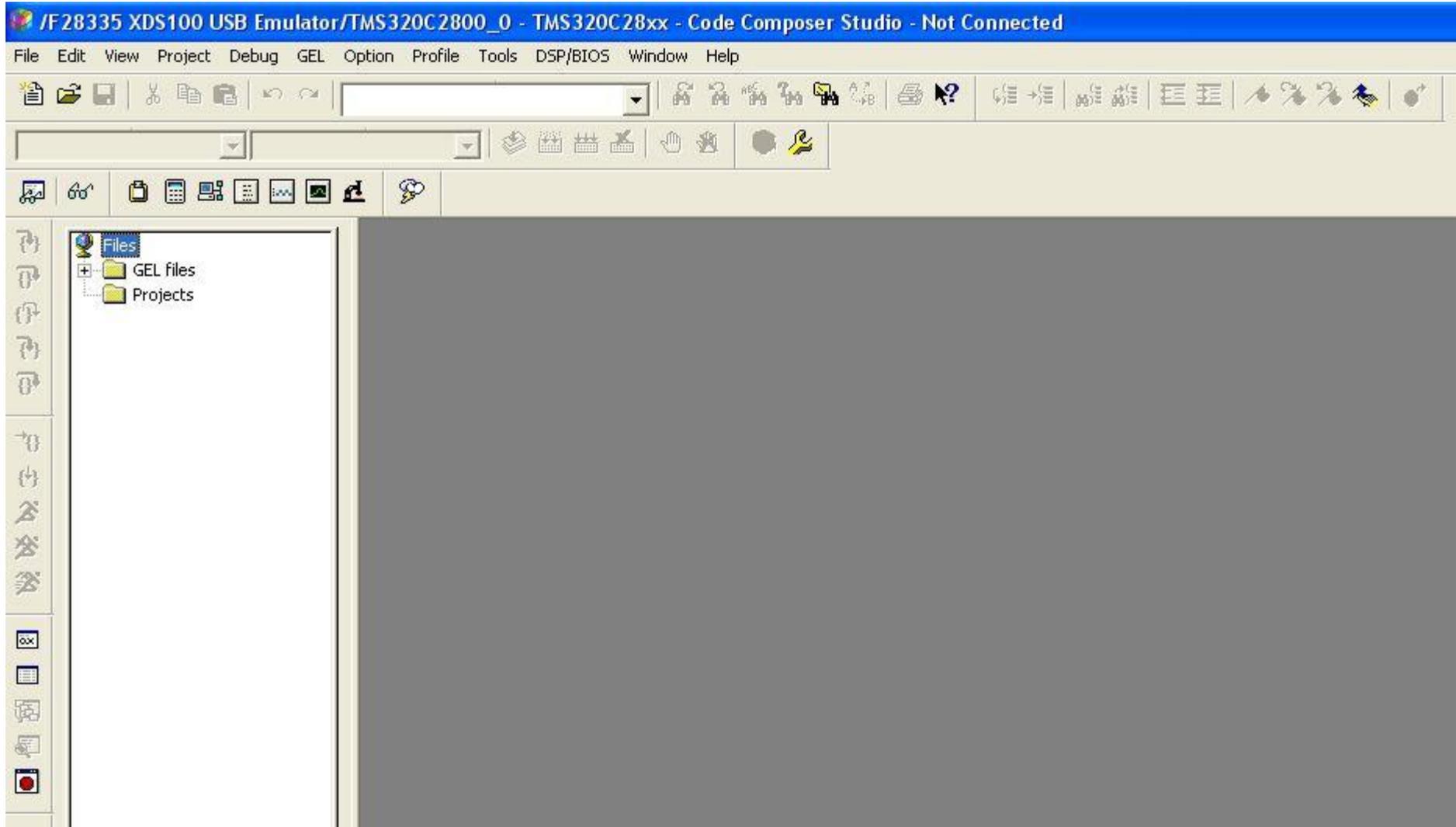
Literature Number: SPRU509H

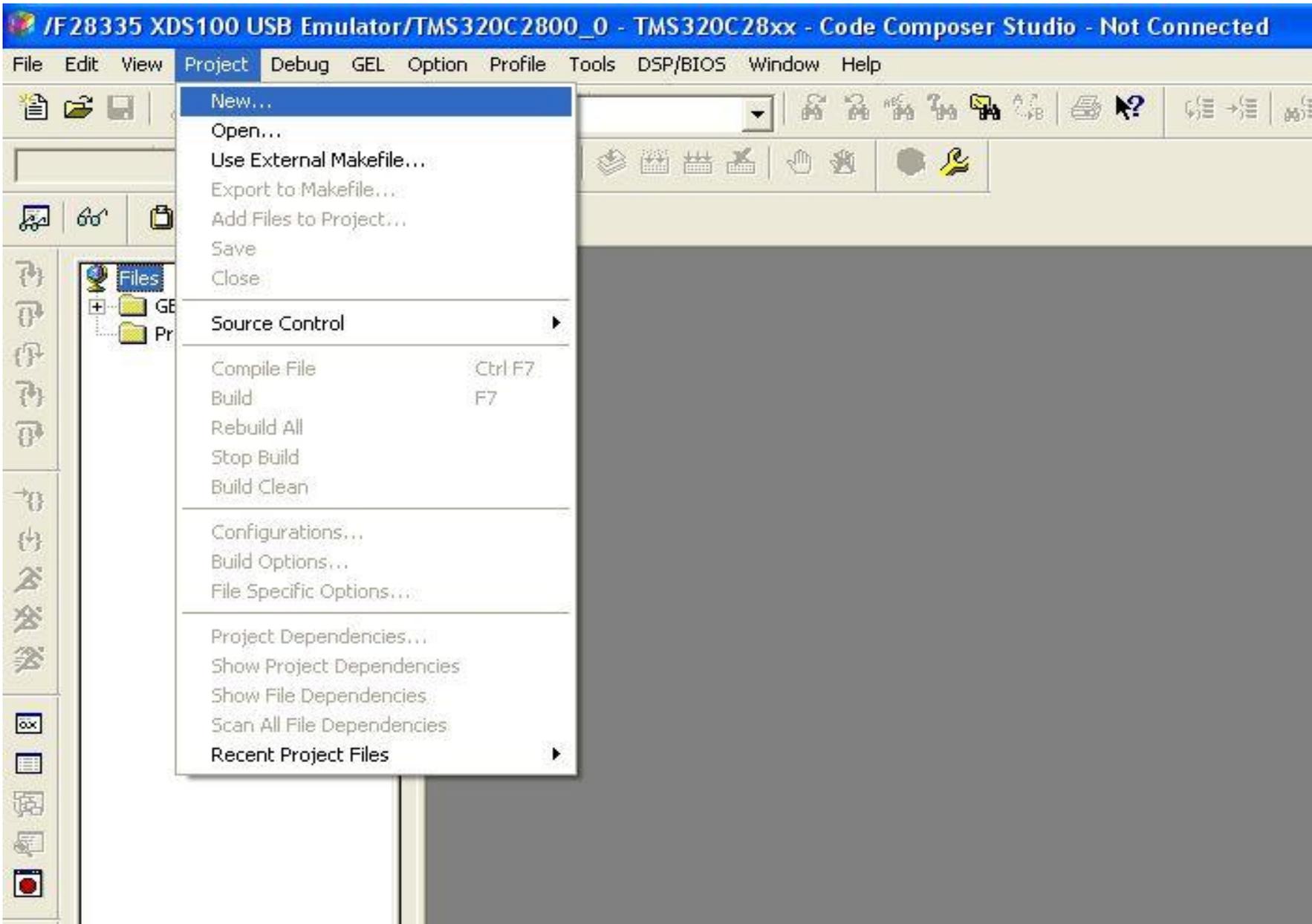
October 2006

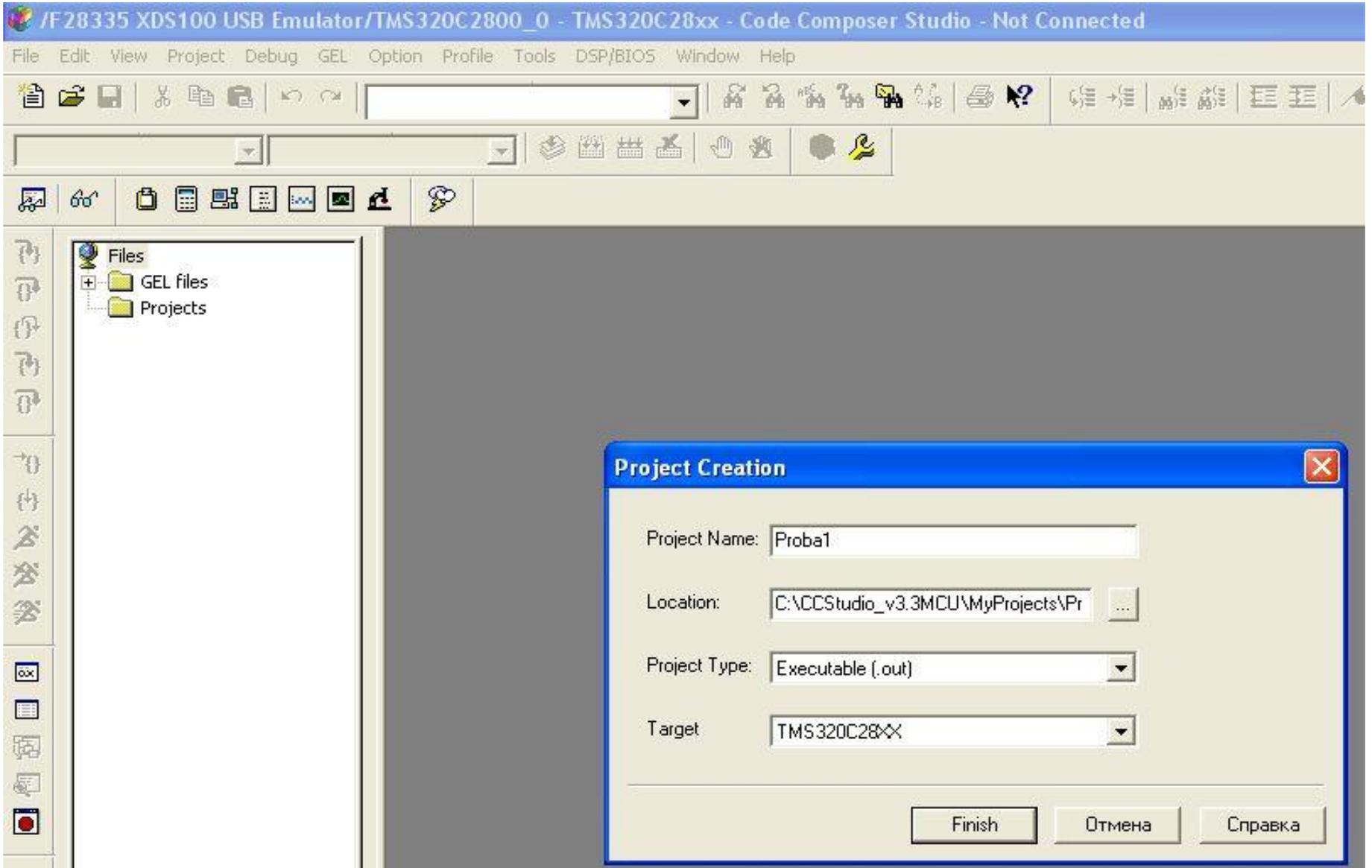
Создание проекта

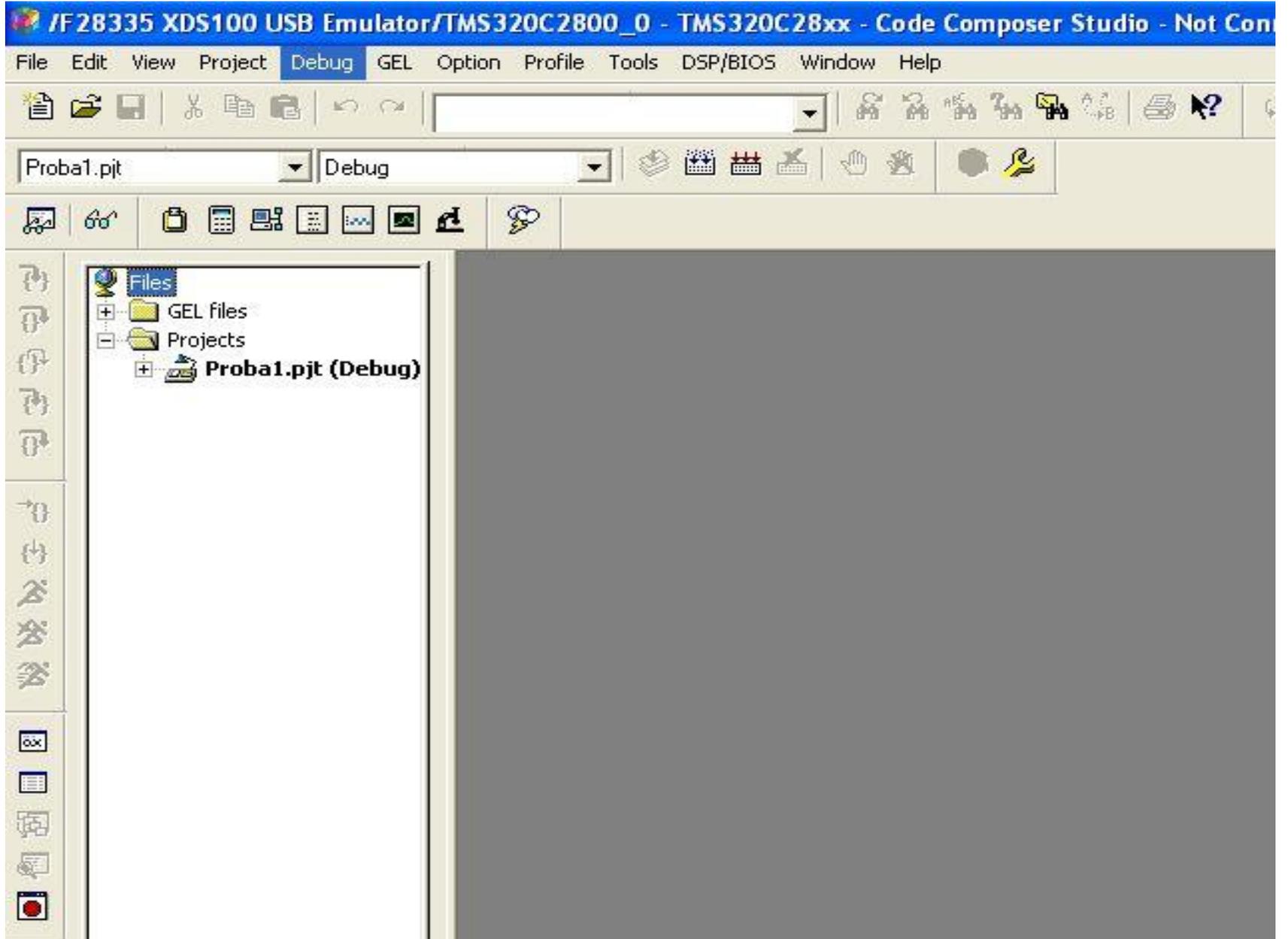
Файл конфигураций проекта содержит:

1. Имена файлов, содержащих исходные тексты программы и файлы объектных библиотек
2. Настройки инструментов генерации исполняемого кода
3. Зависимости вхождения файлов

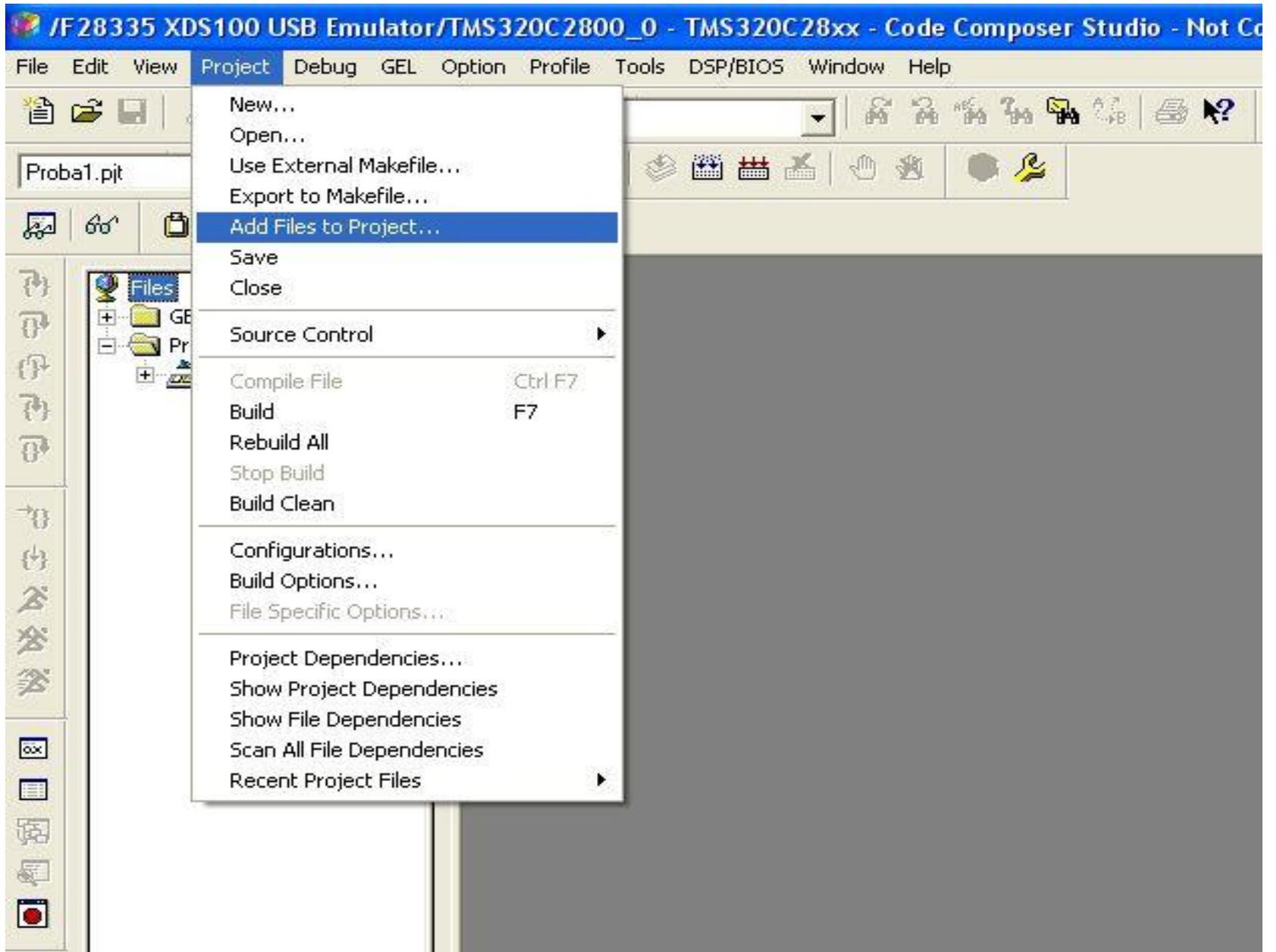


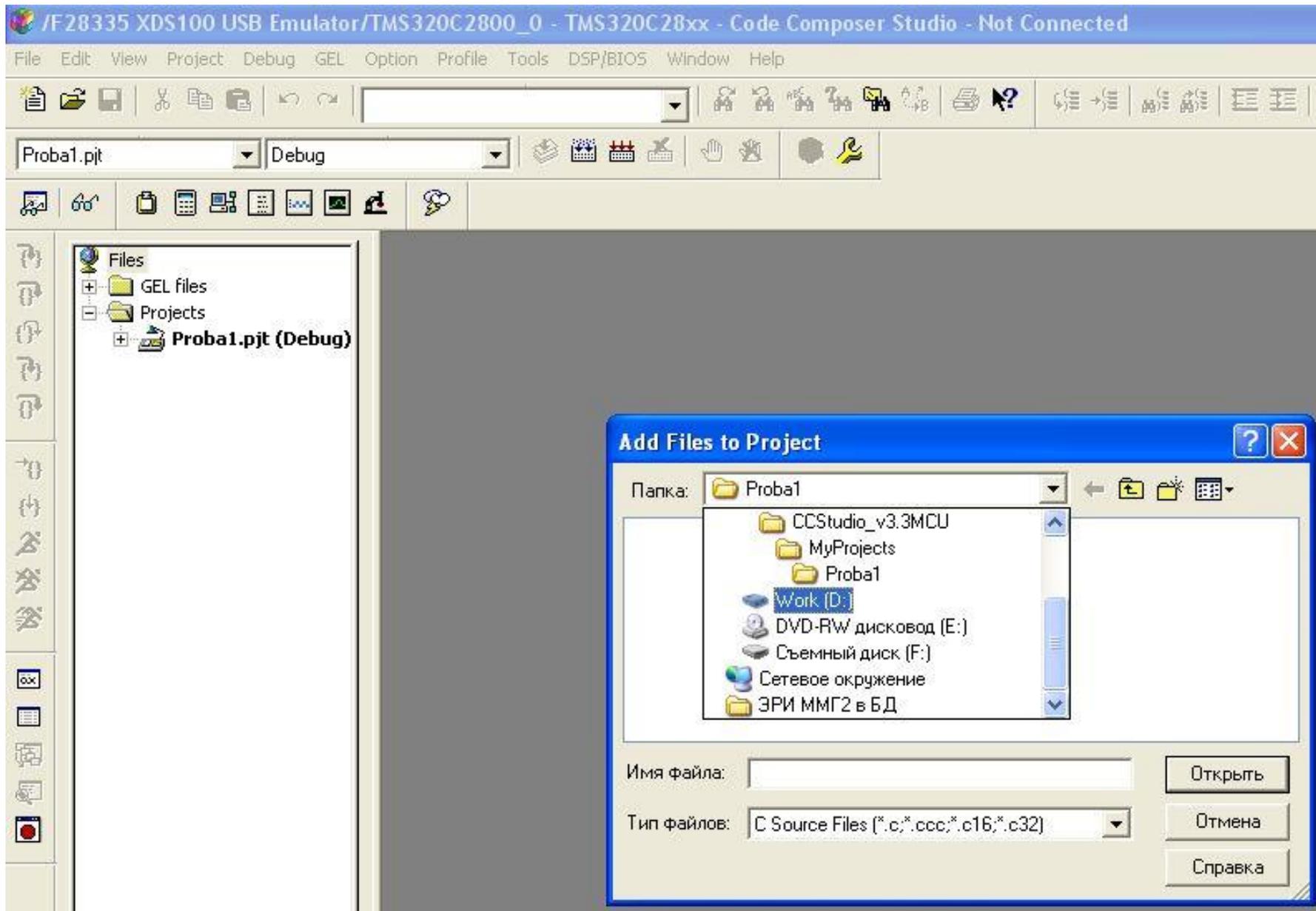


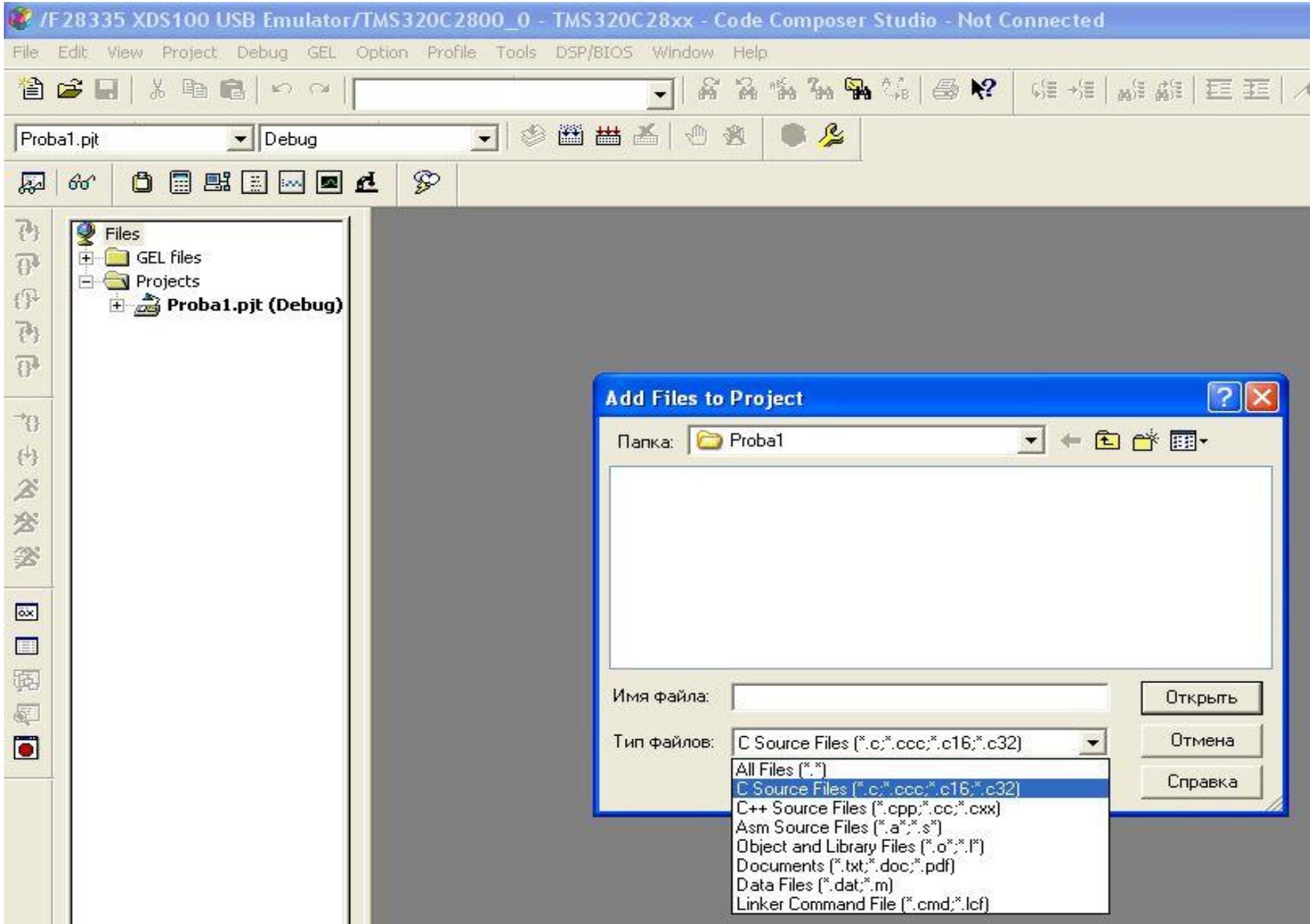




Добавление файлов в проект

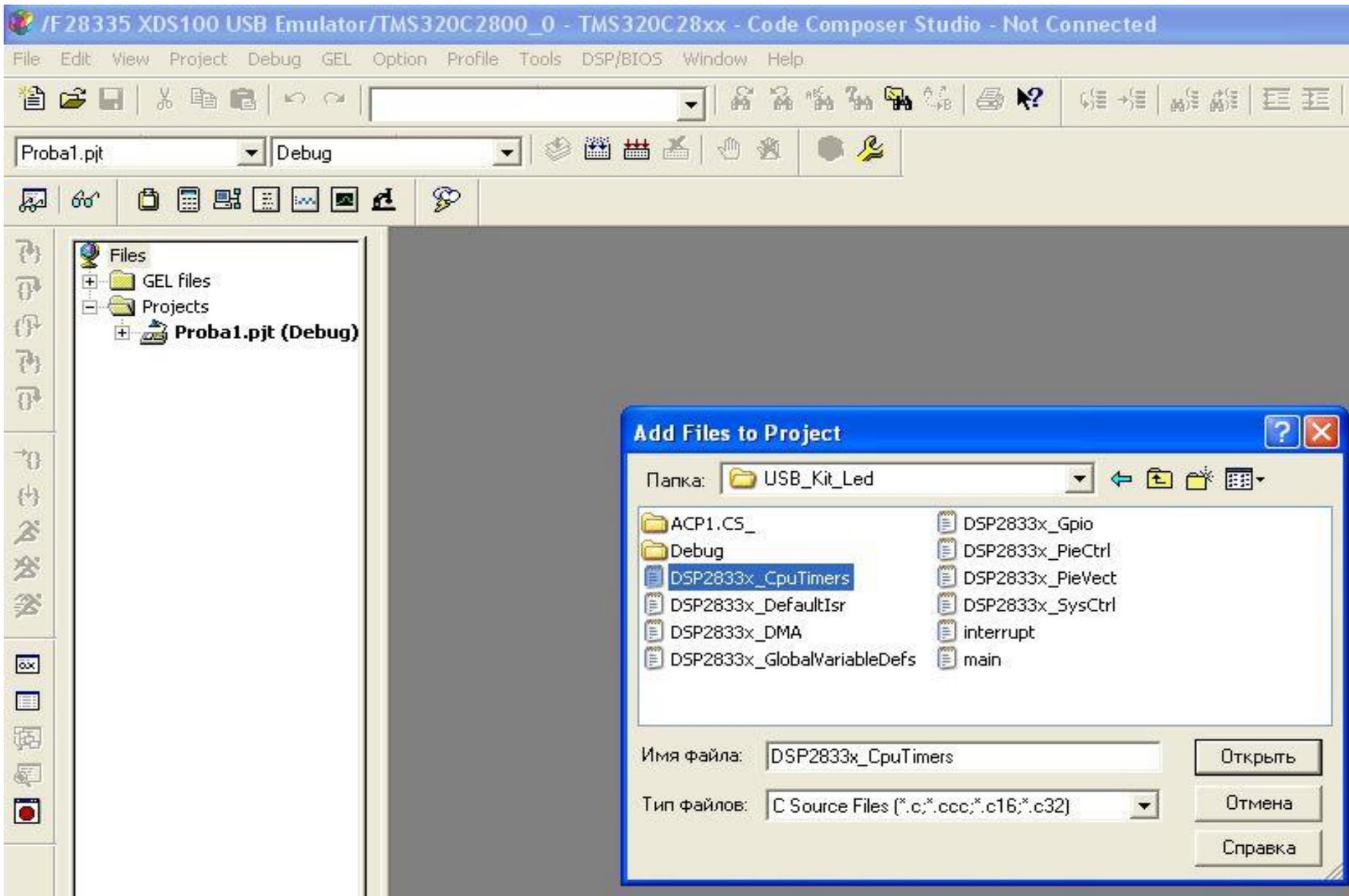


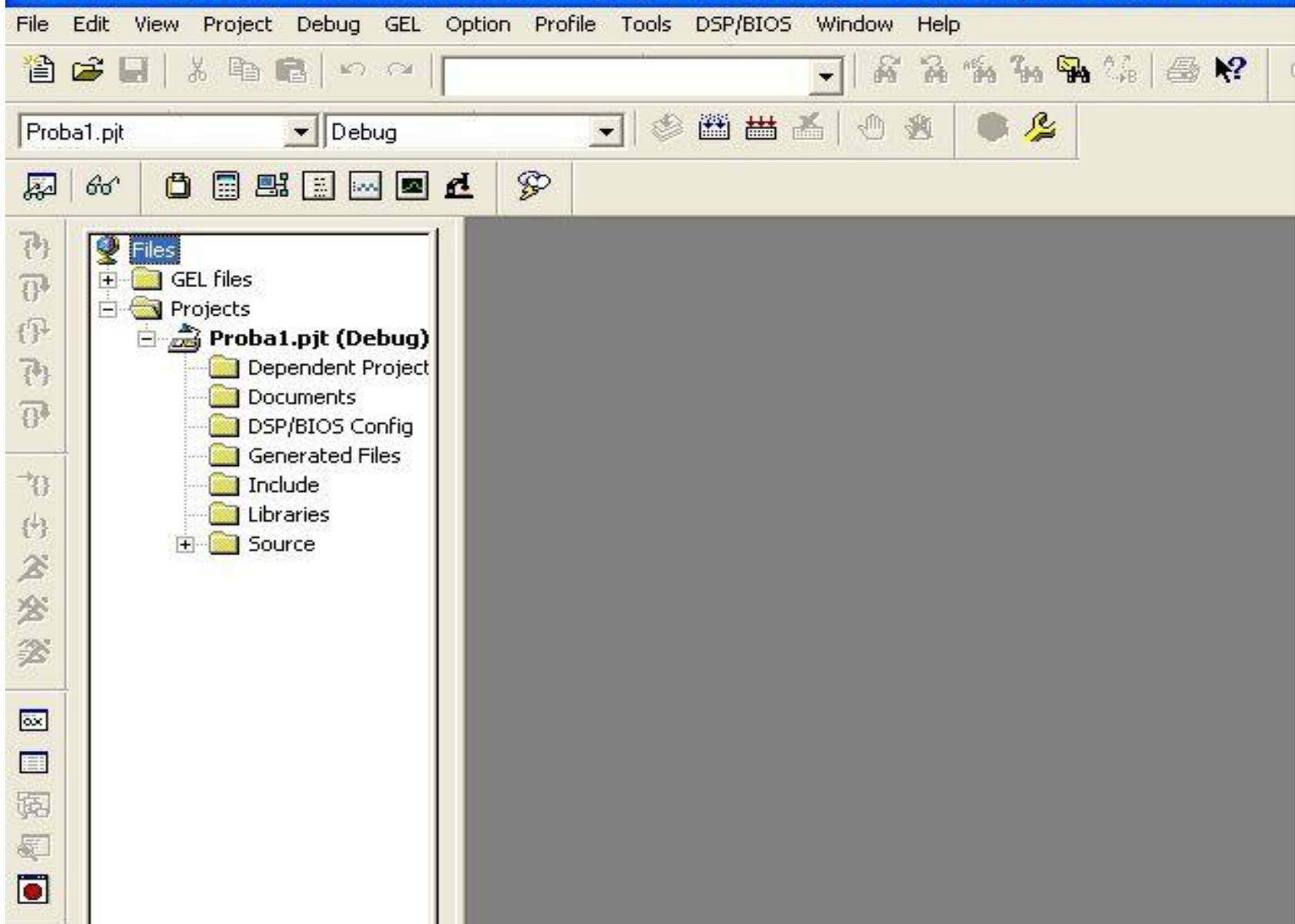


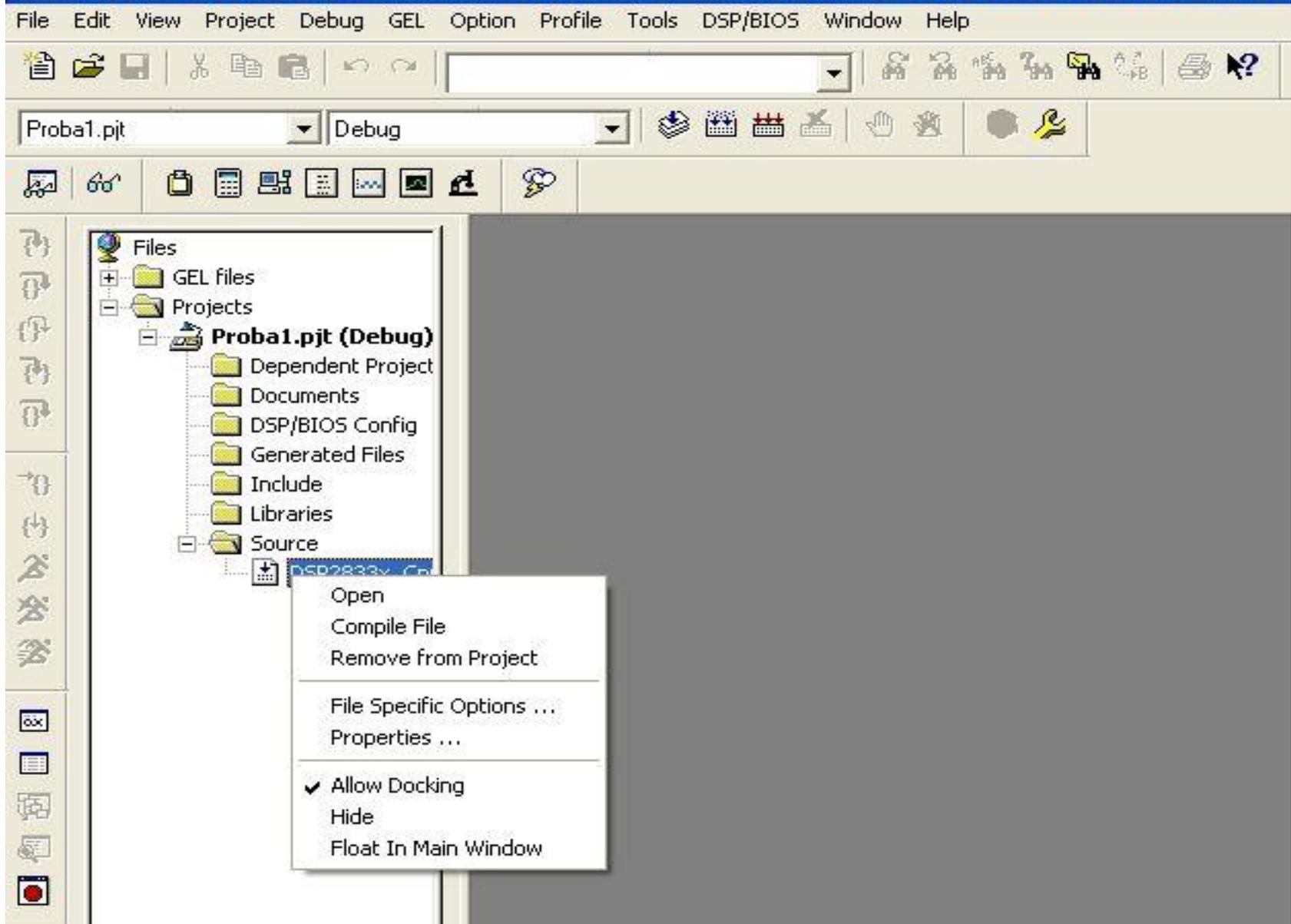


Типы поддерживаемых файлов

- *.c, *.cсс – файлы, использующие правила языка C
- *.cpp, *.cc, *.cхх – файлы, использующие правила языка C++
- *.asm - файлы, использующие правила языка Assembler
- *.lib – файлы объектных библиотек
- *.cmd – файл управления линкером

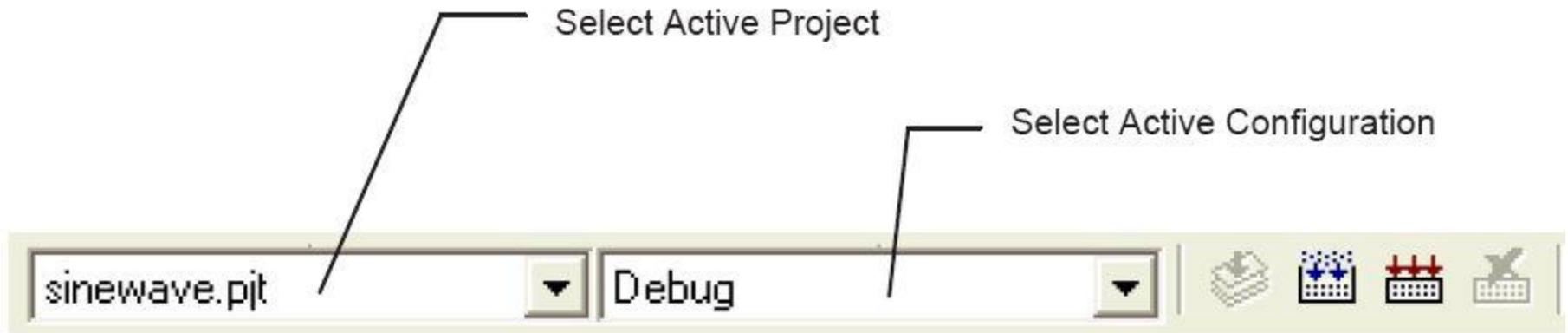


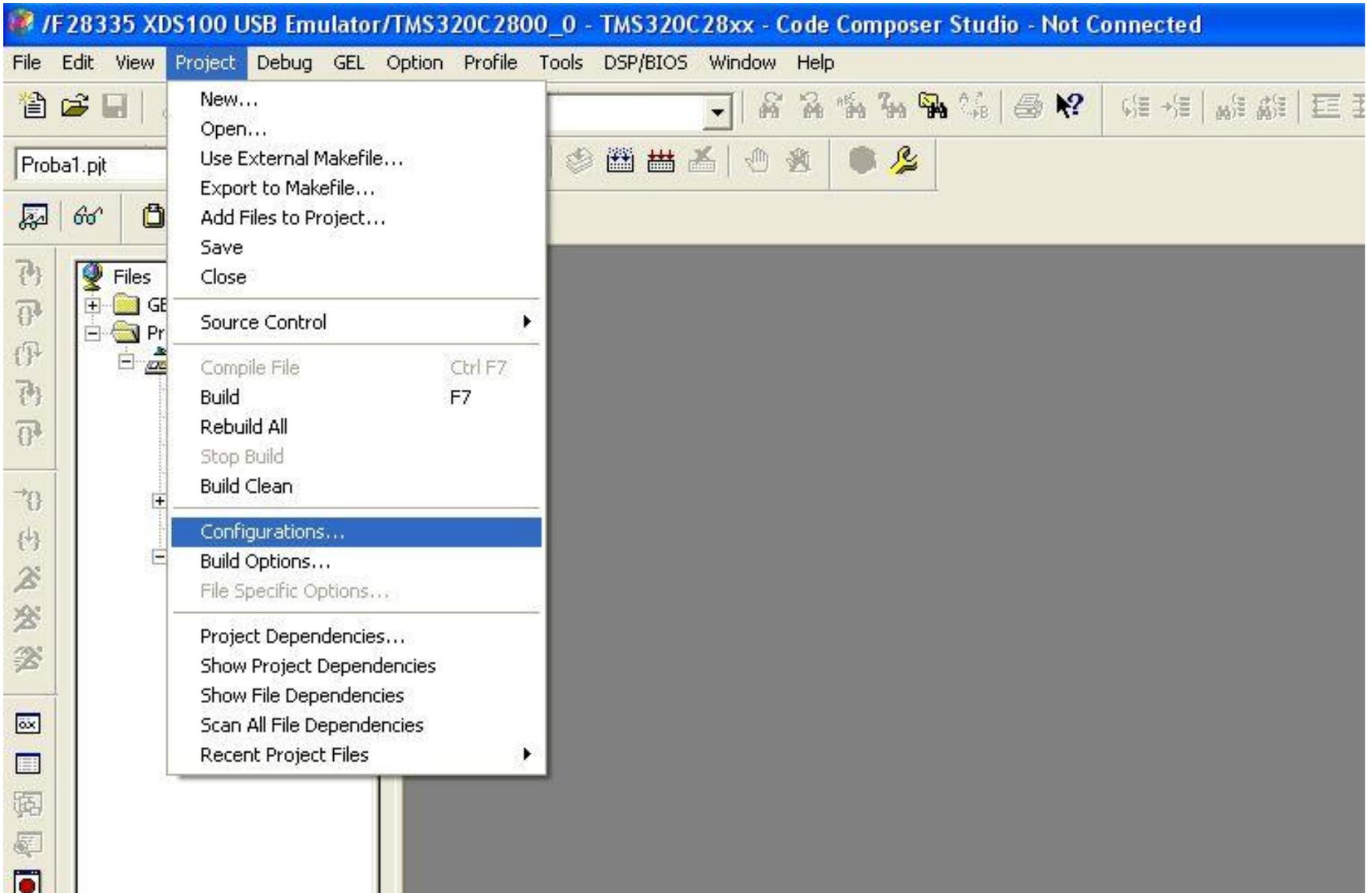


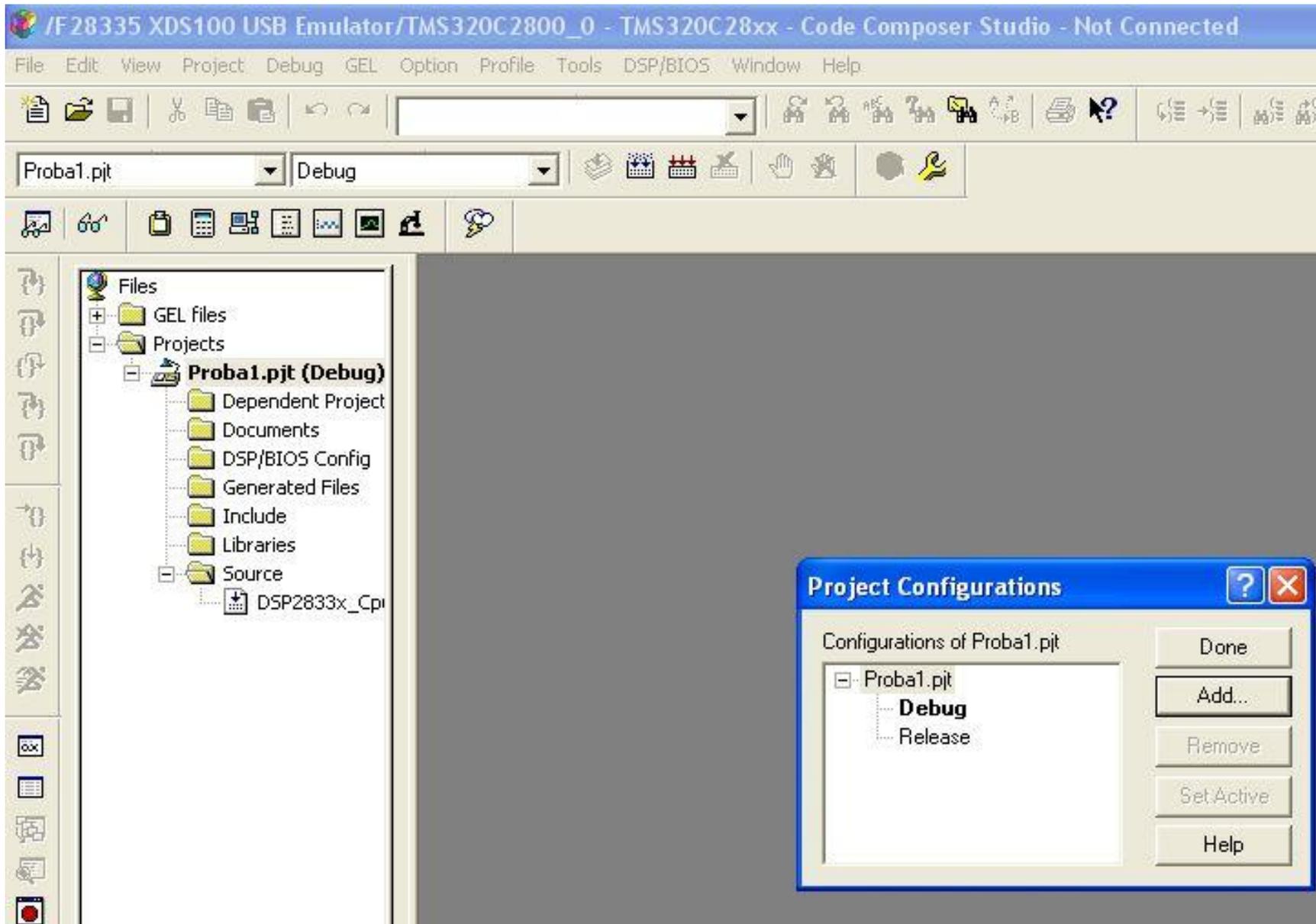


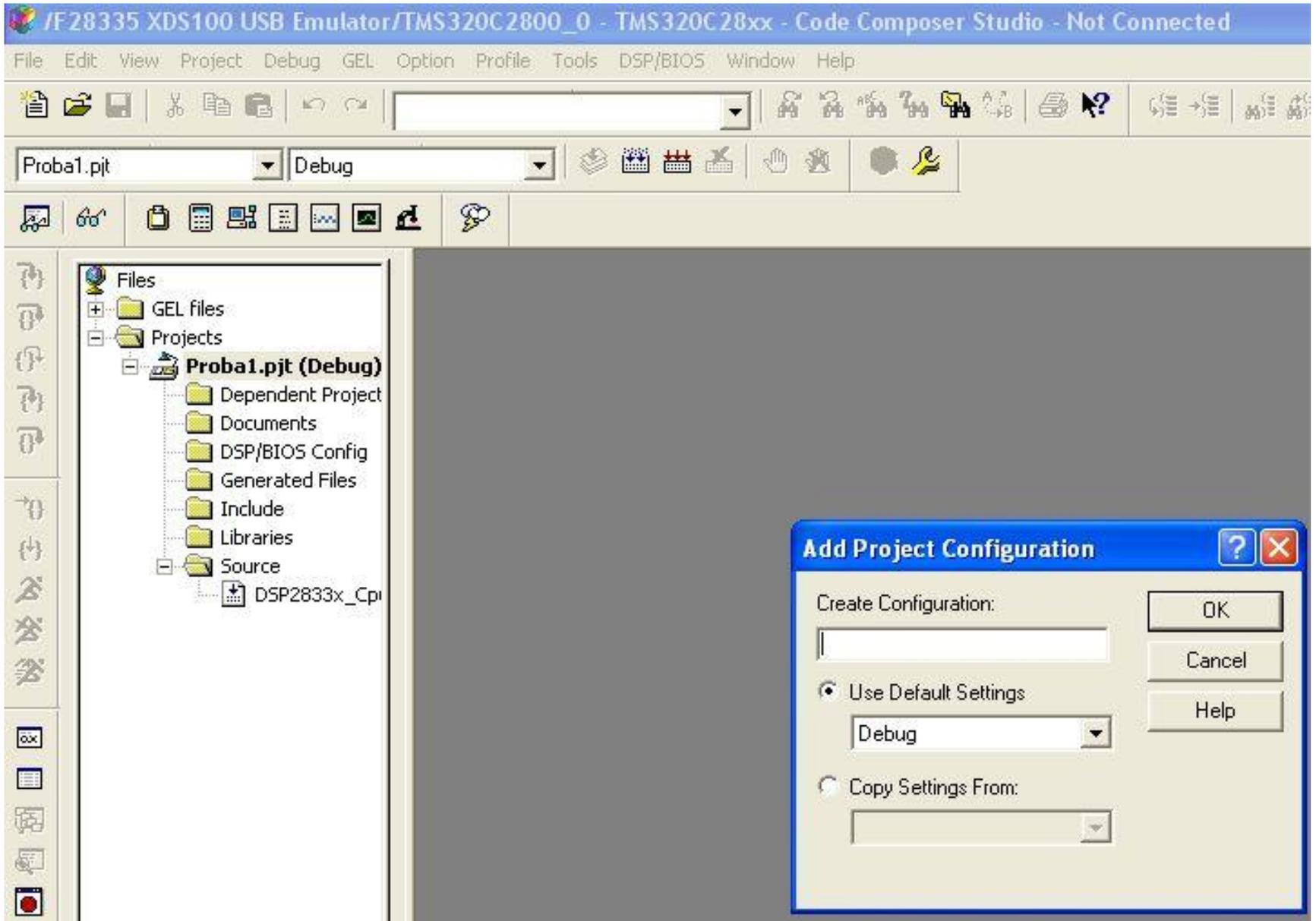
Конфигурация проекта

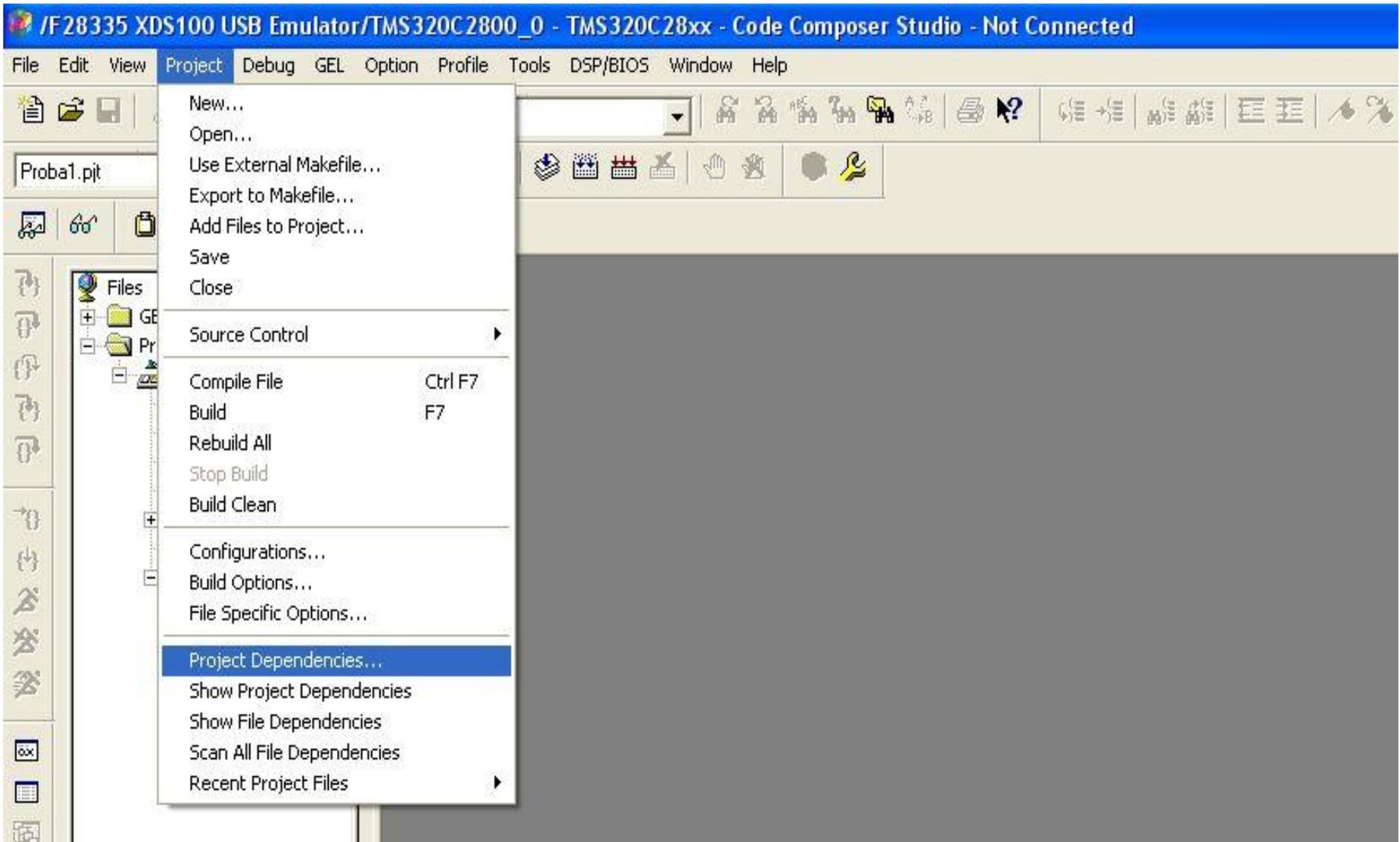
Figure 4-4. Configuration Toolbar











Текстовый редактор

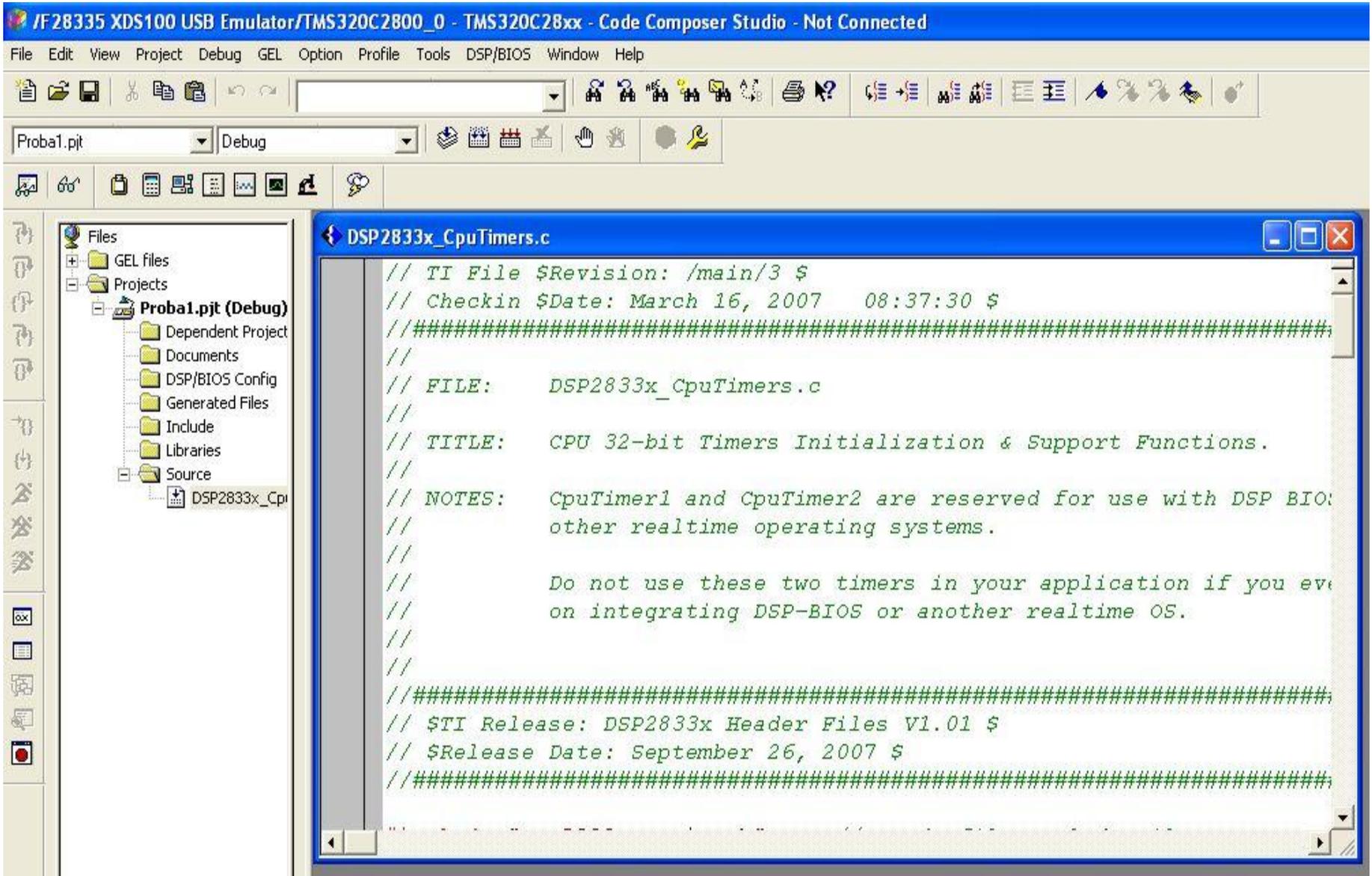
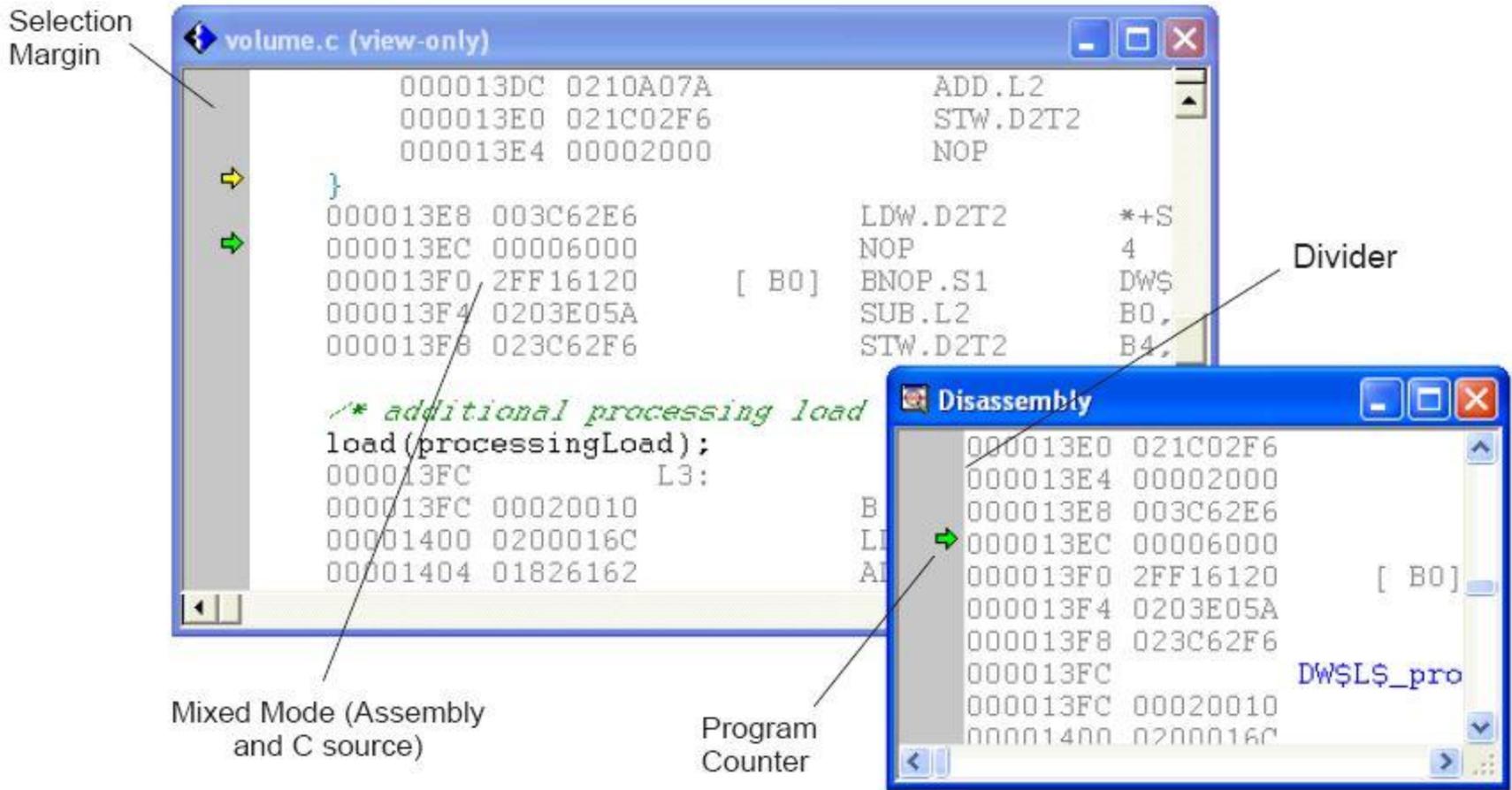
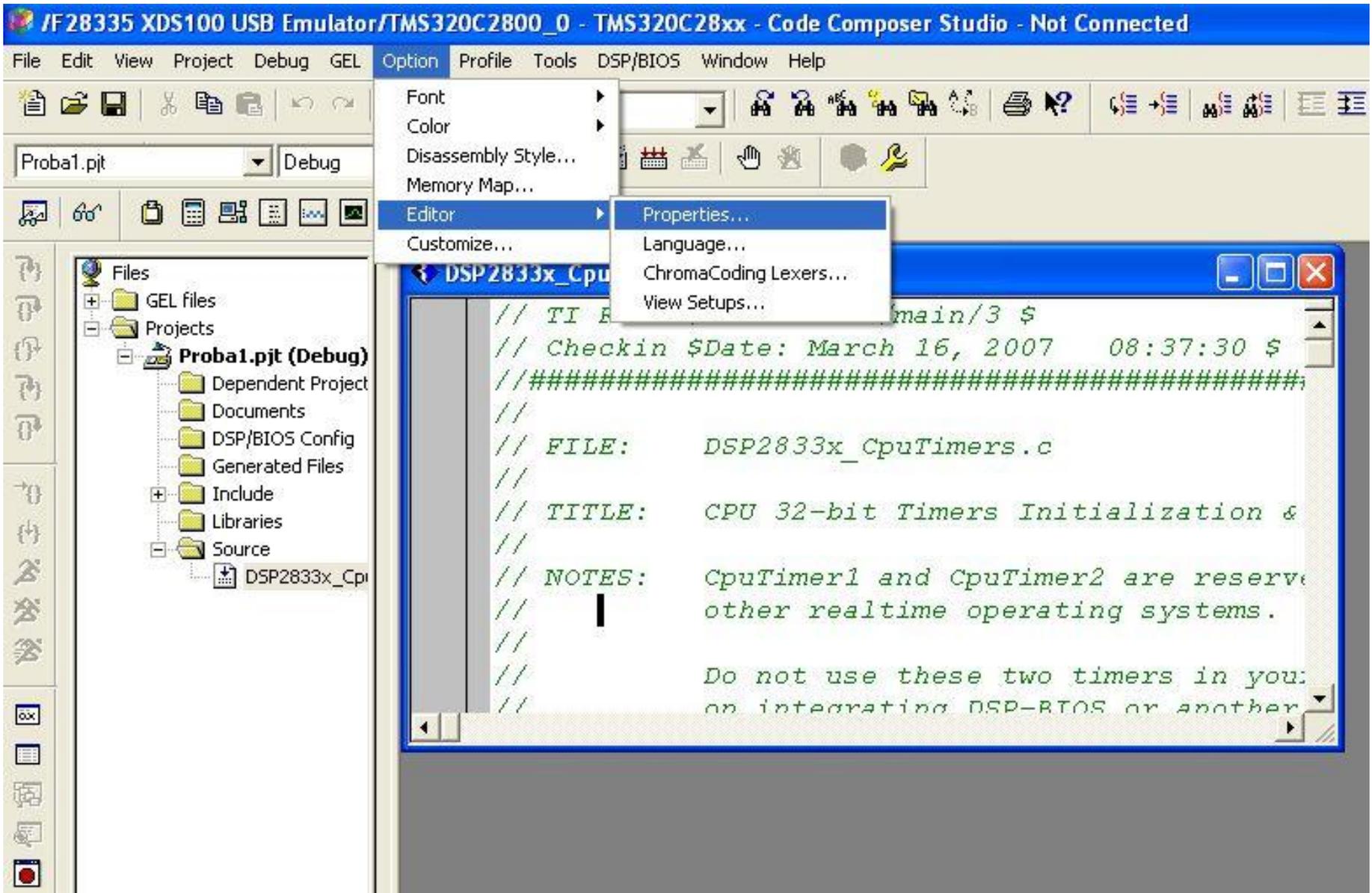
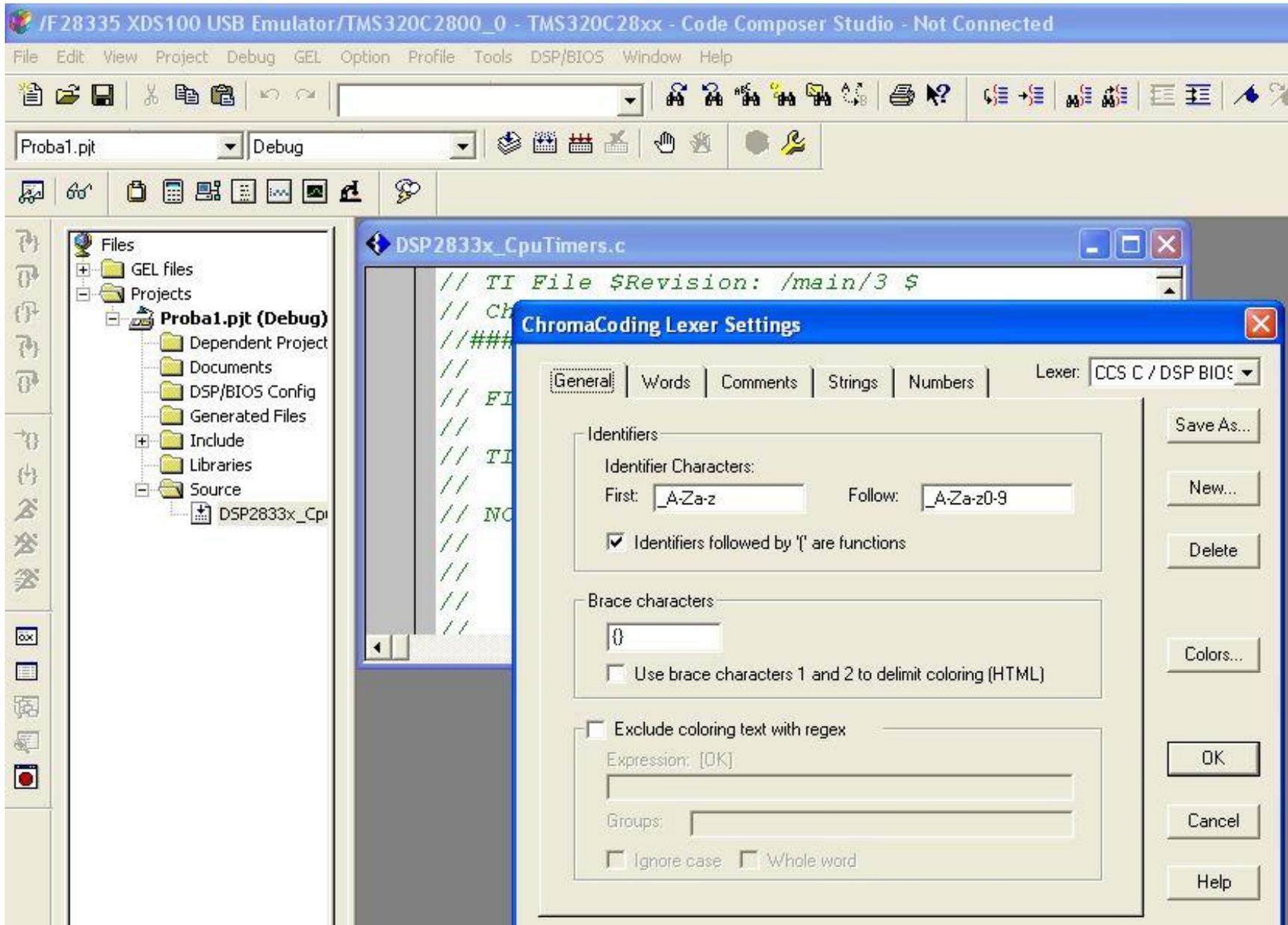
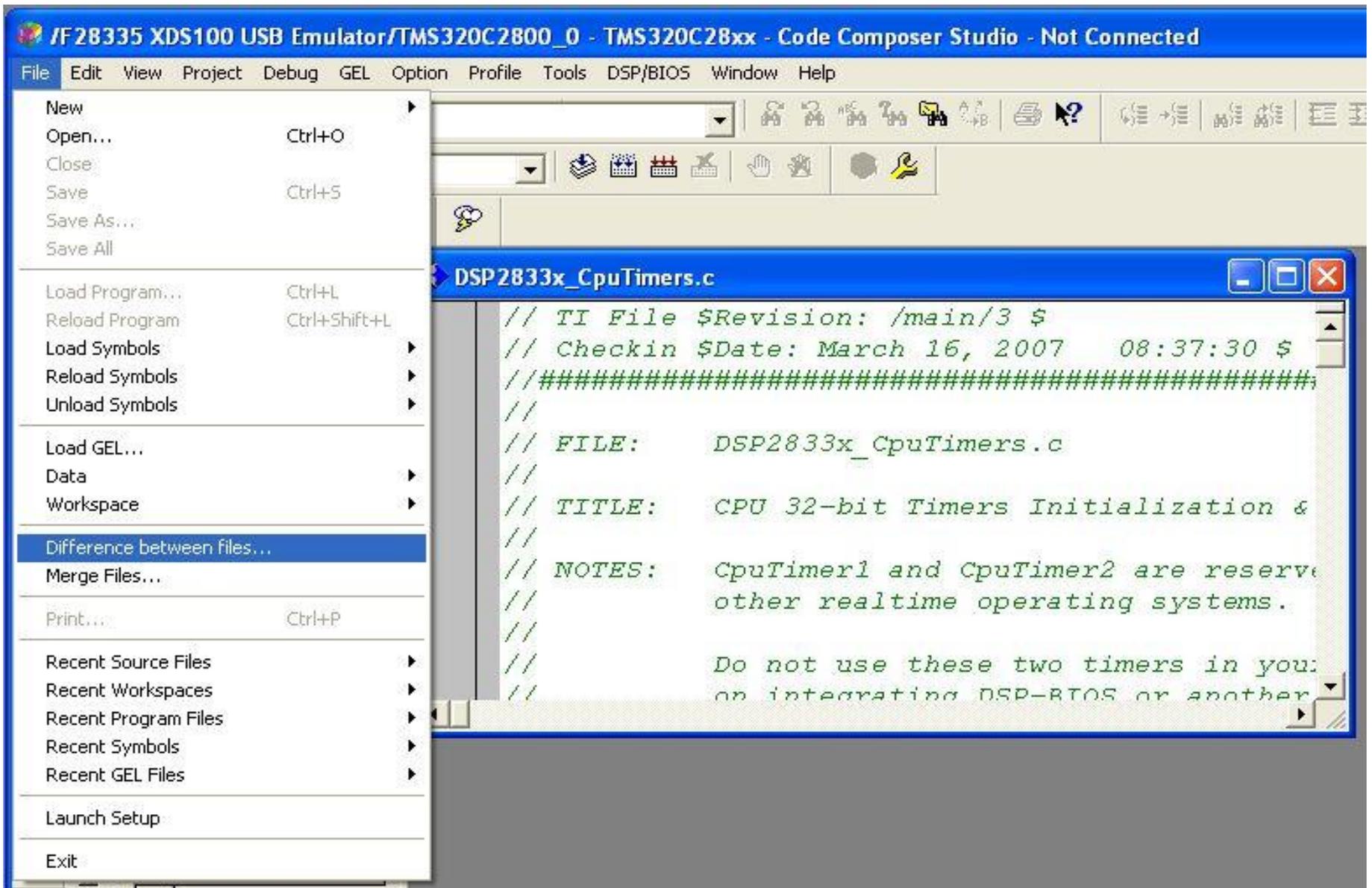


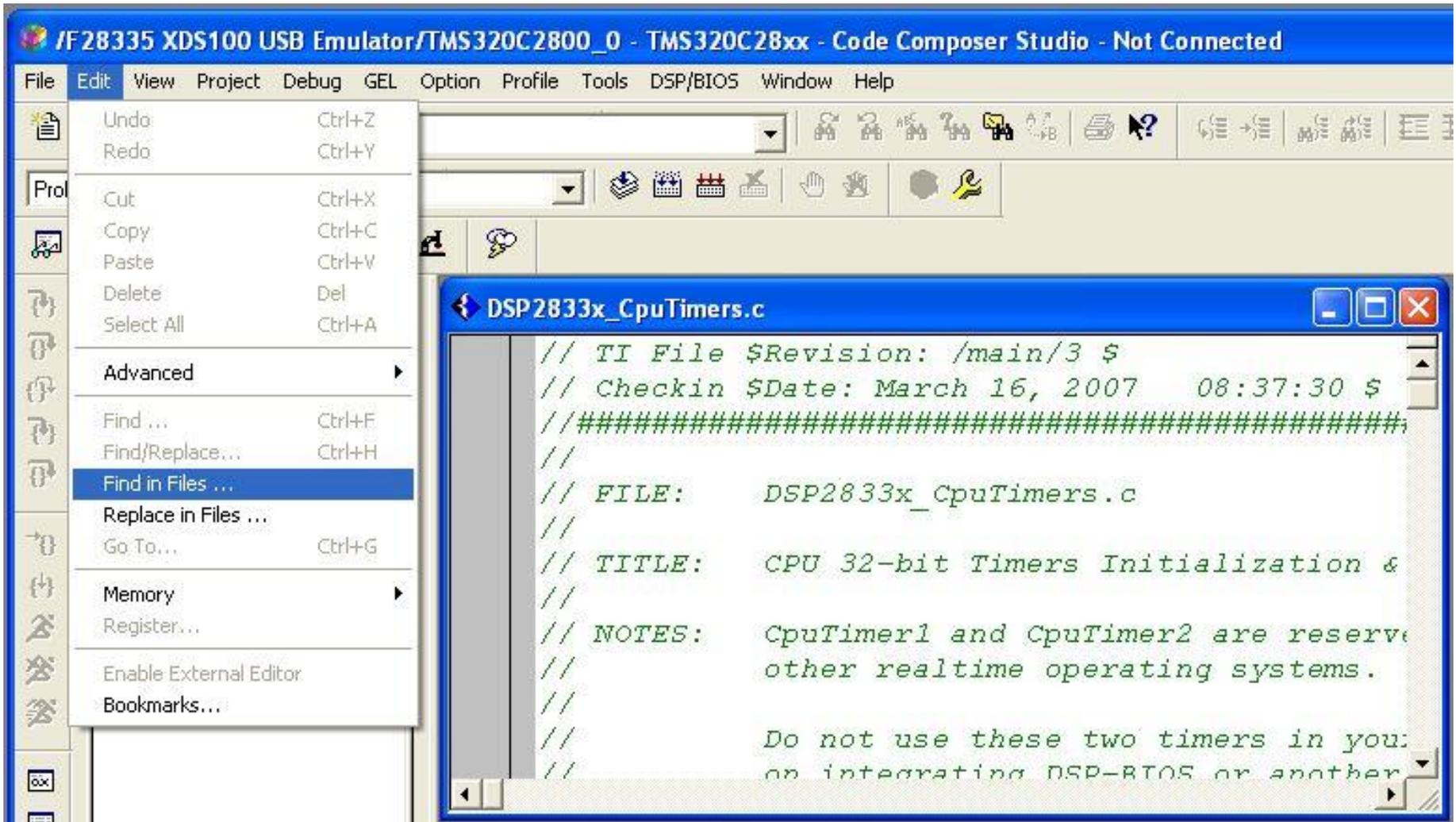
Figure 4-8. Elements in the Source Code Window

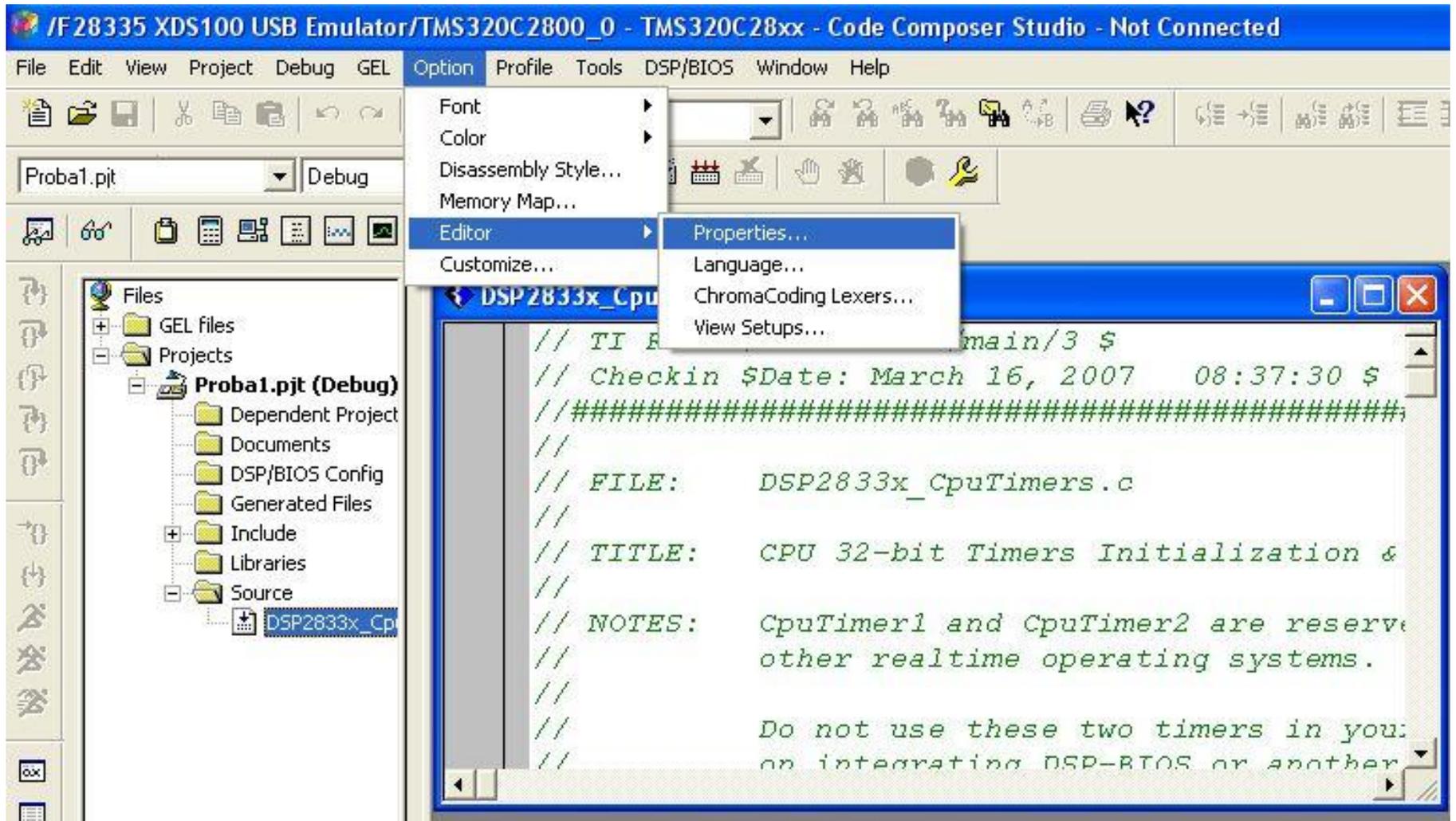


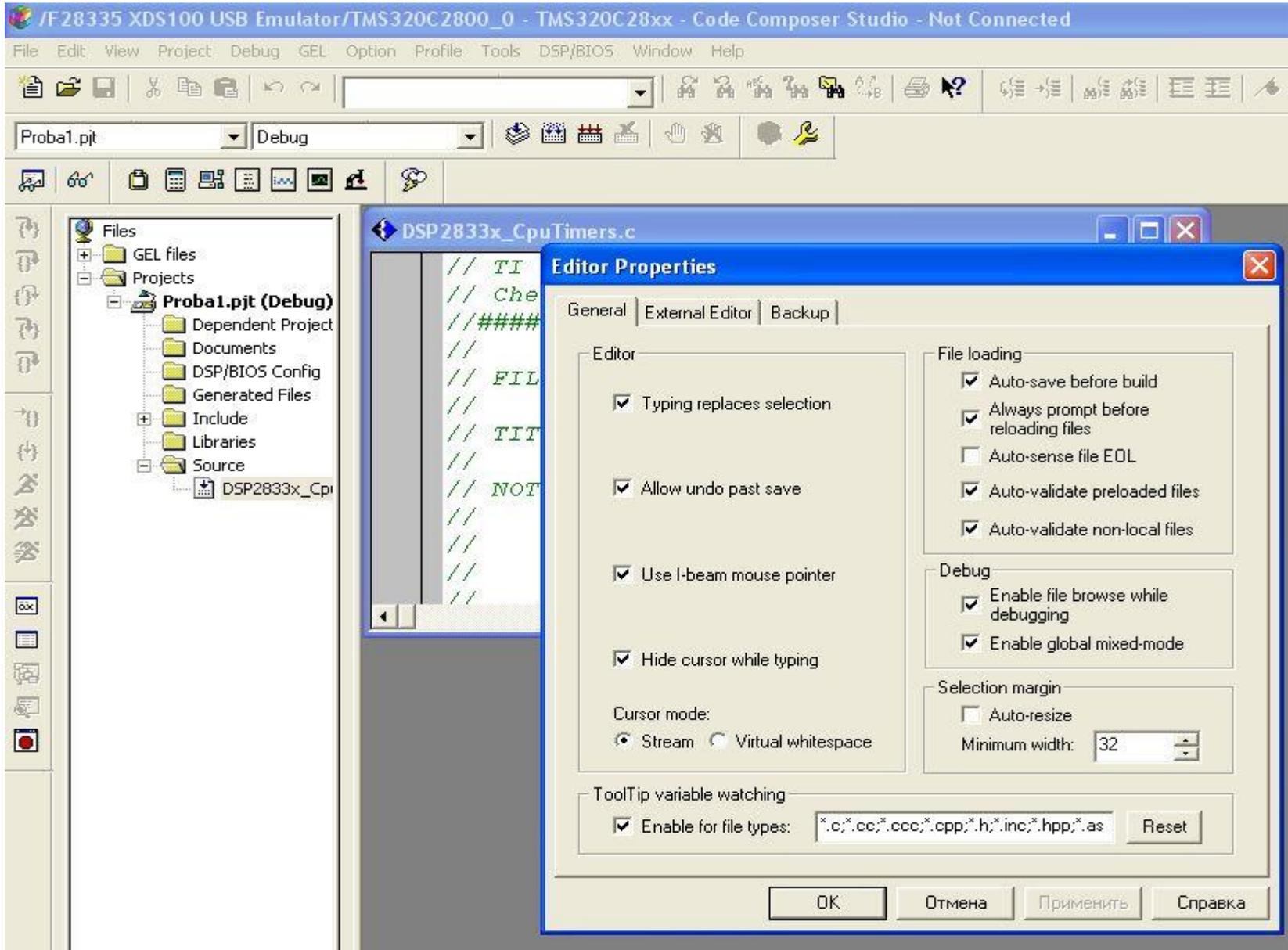


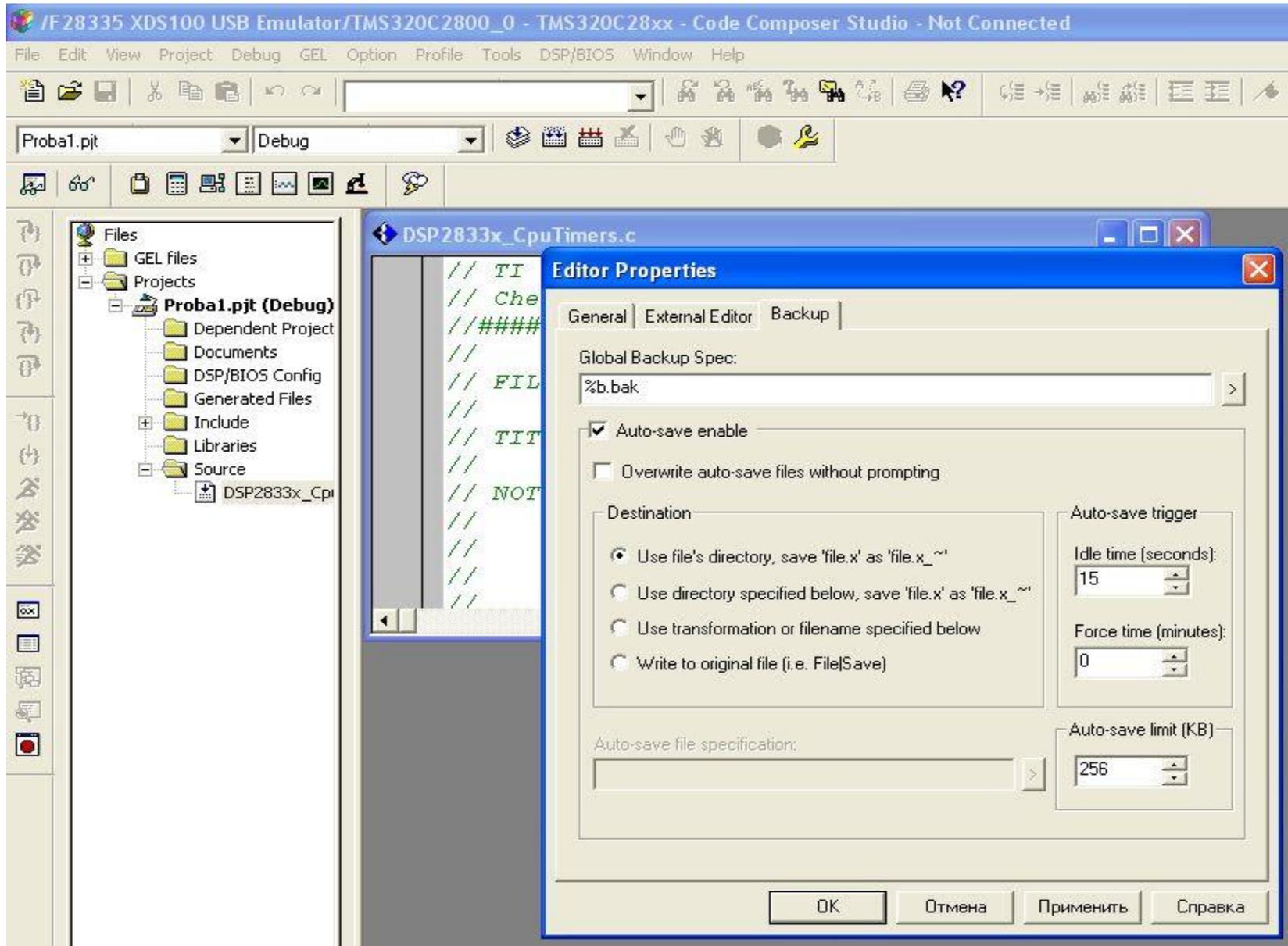


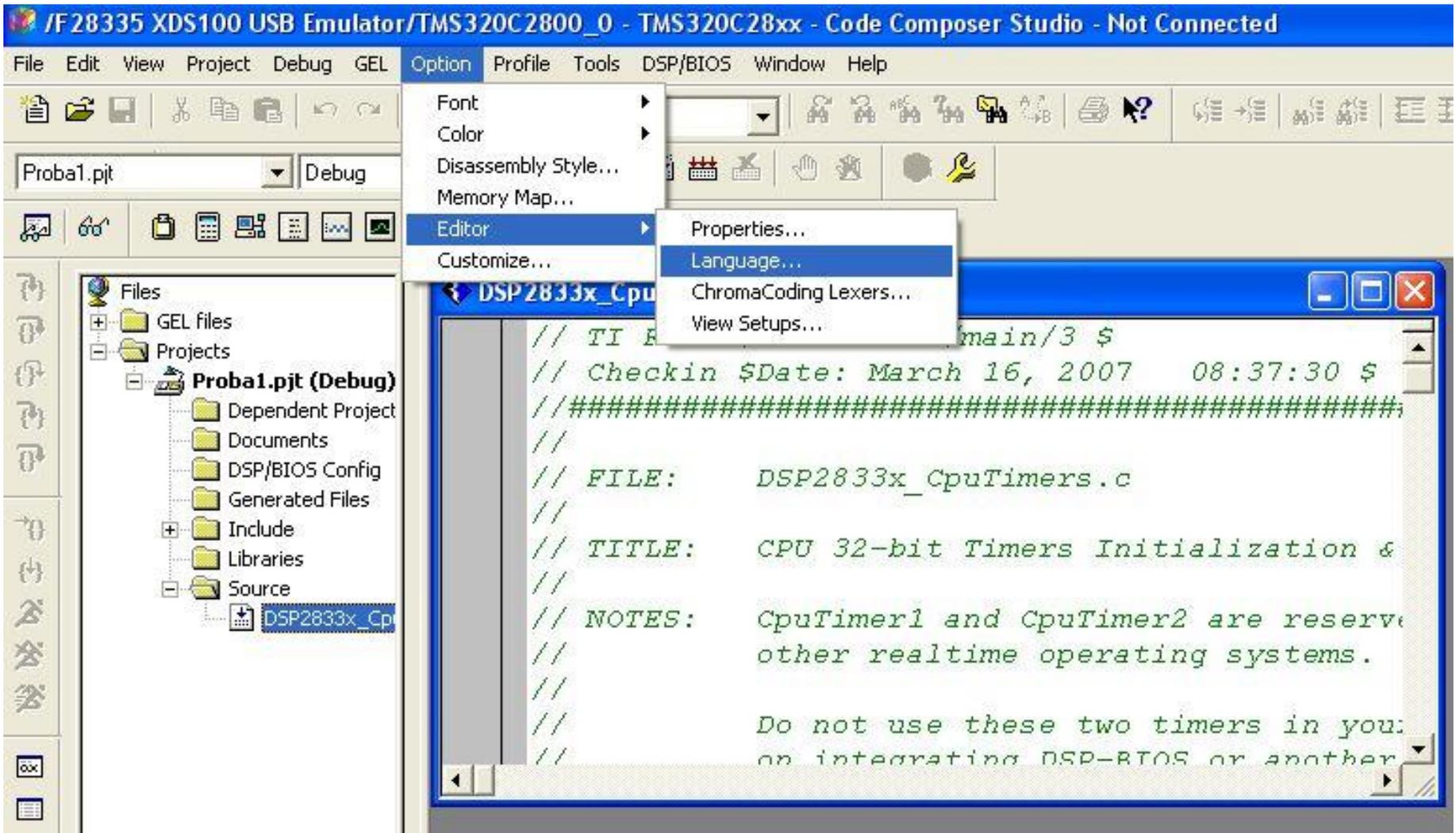


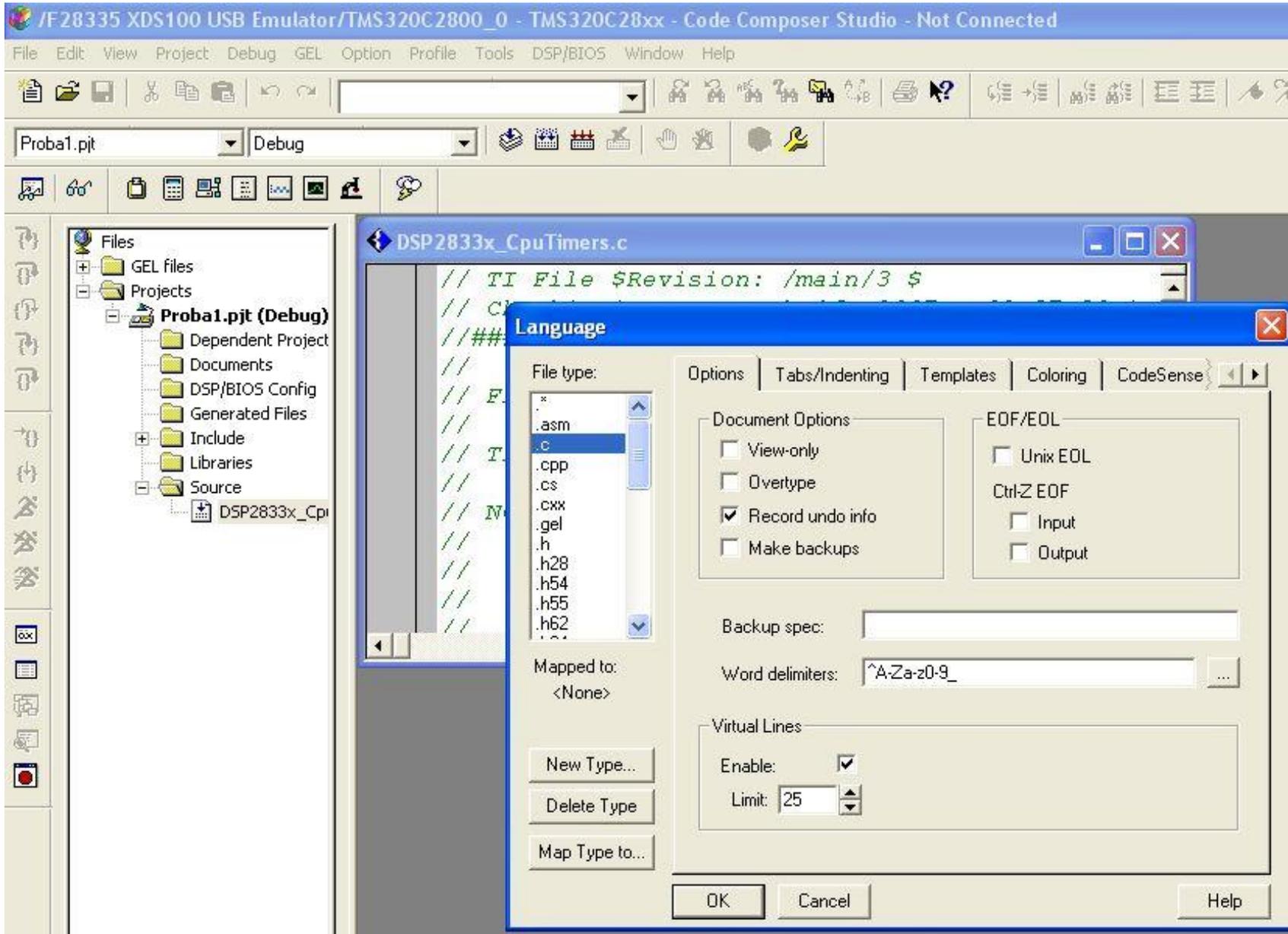












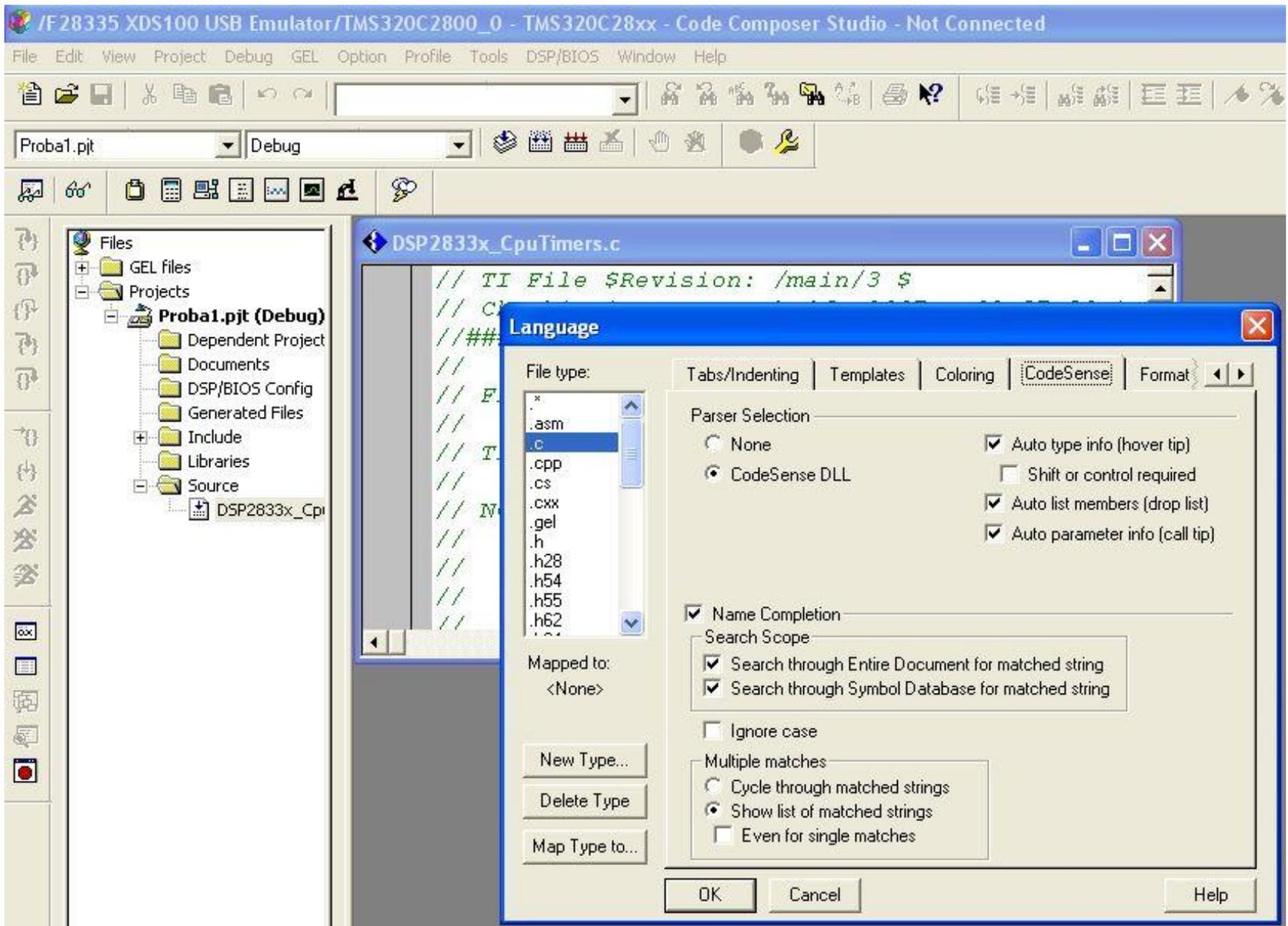
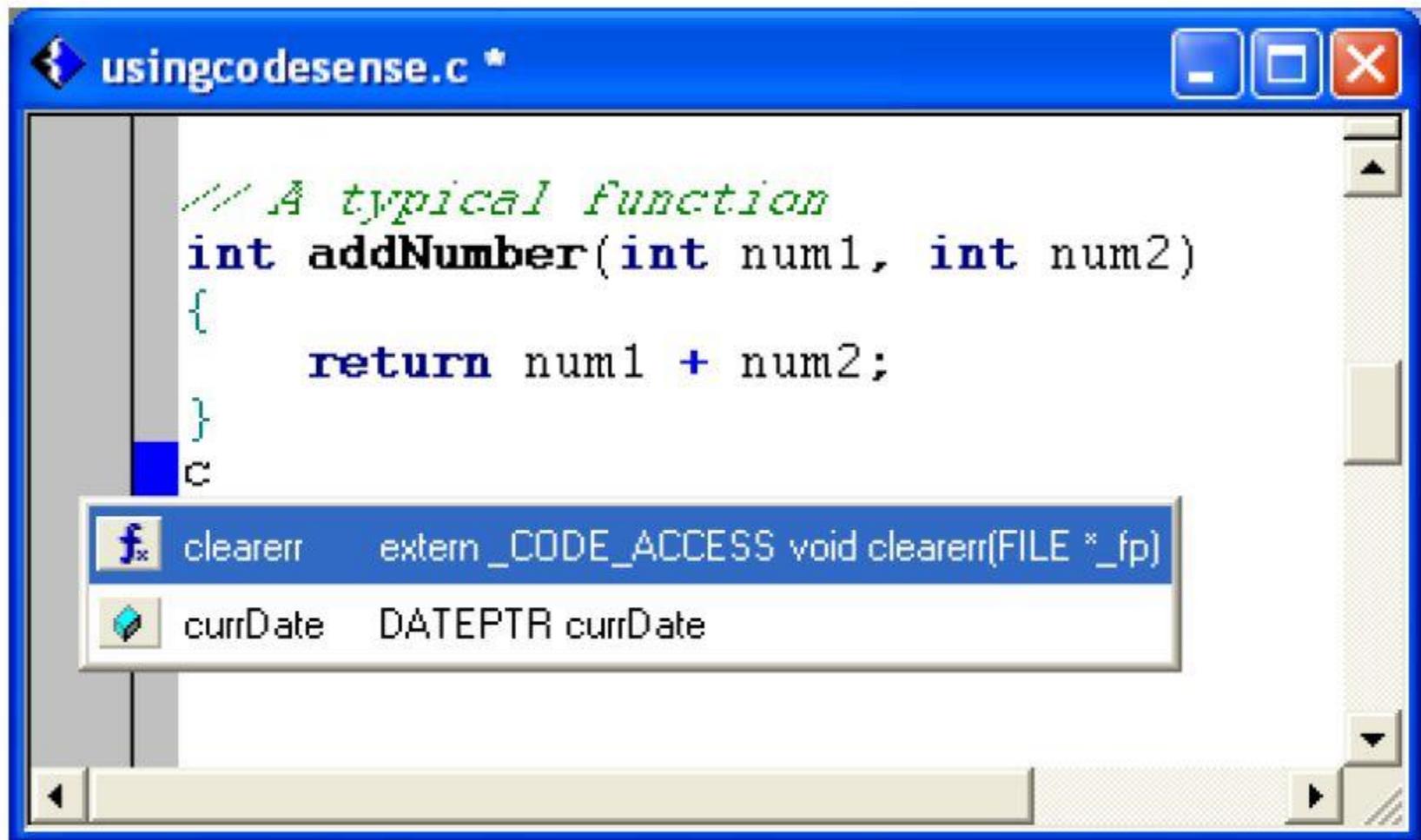


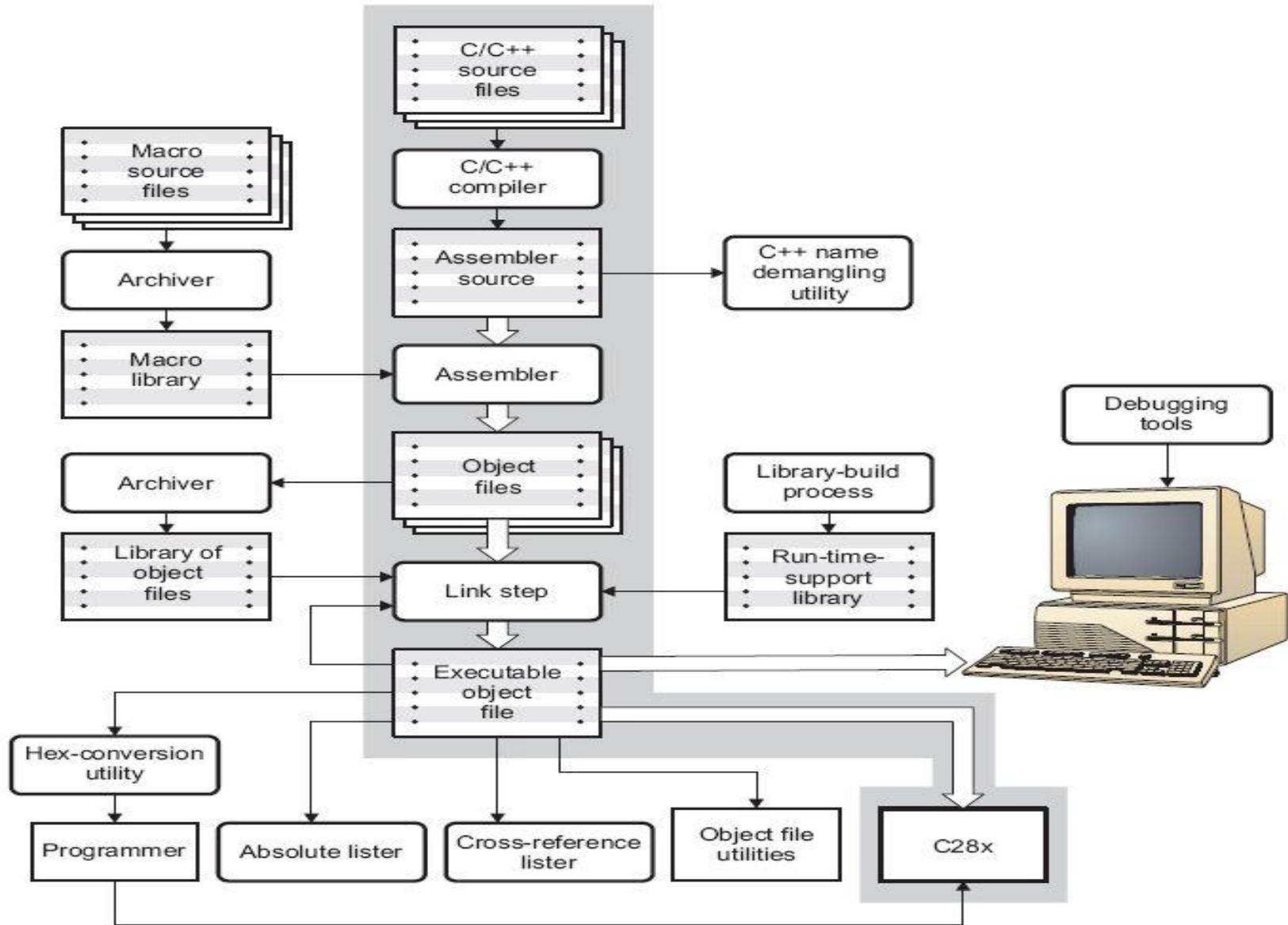
Figure 4-11. Code Sense



Инструментальные средства для генерации кода

Установка опций для проекта и файла

Figure 1-1. TMS320C28x Software Development Flow



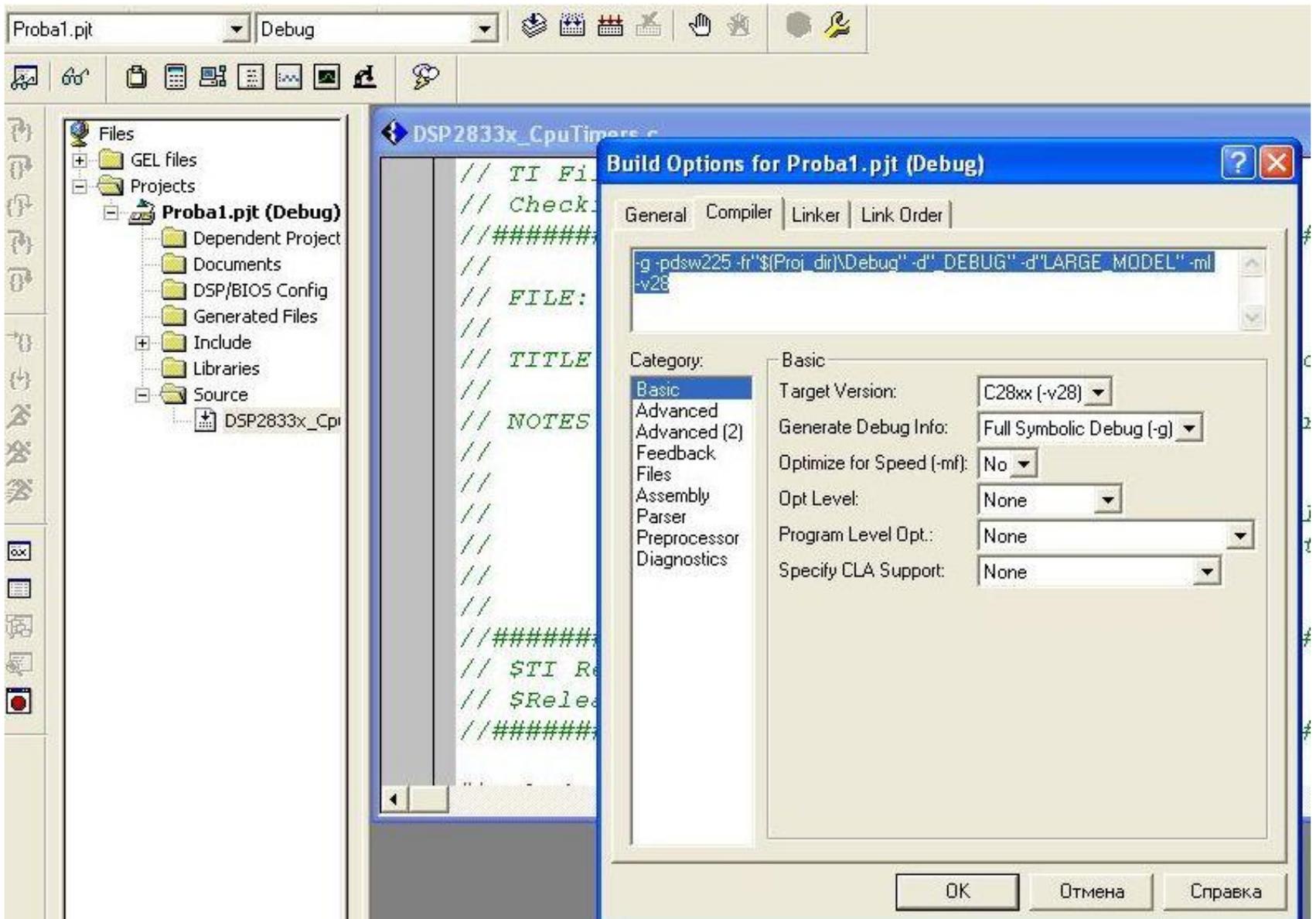
Вызов утилиты компиляции

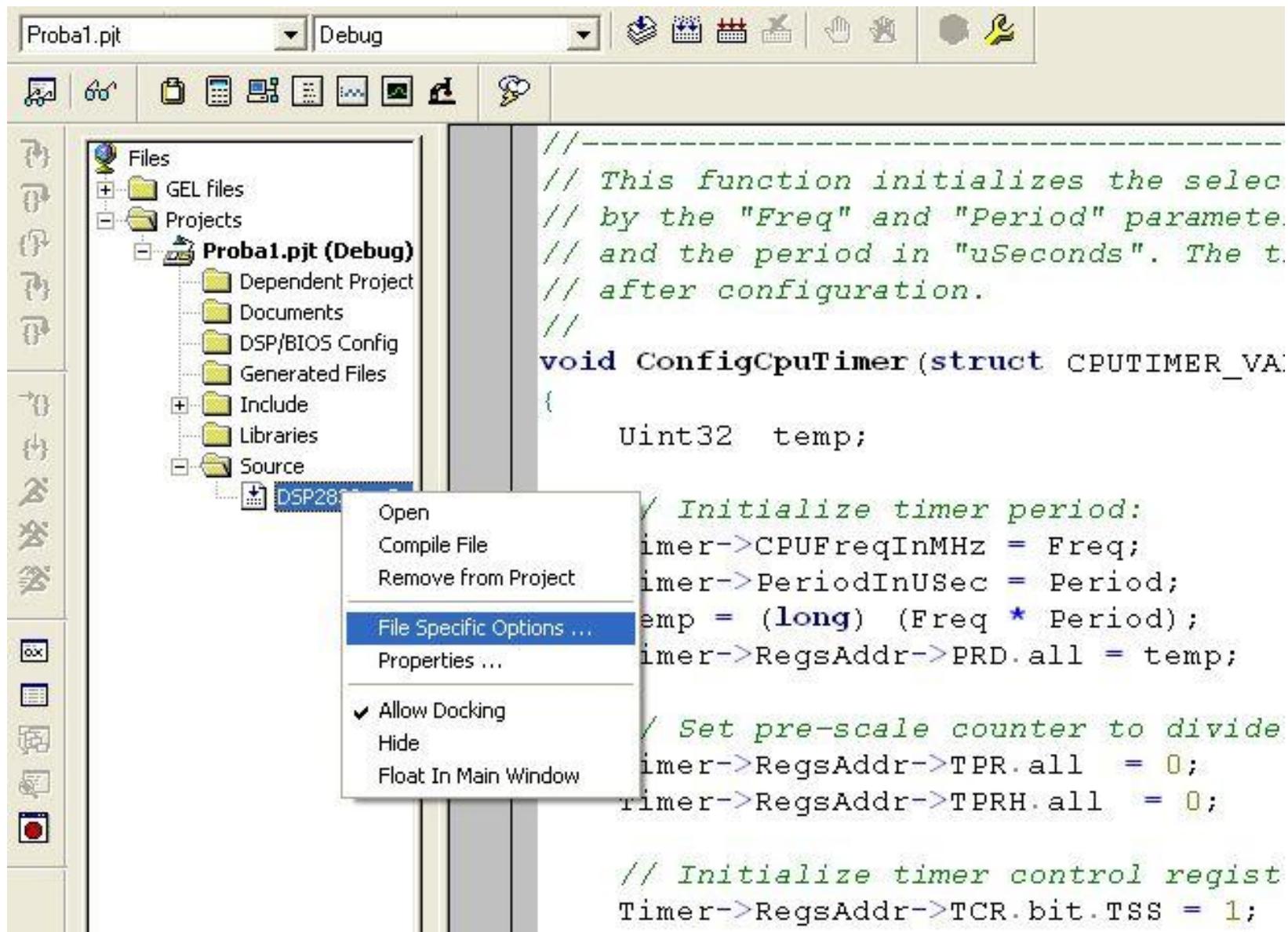
cl2000 -v28 [*options*] [*filenames*] [--run_linker [*link_options*] *object files*]]

- **cl2000 -v28** – команда запуска компилятора и ассемблера.
- *options* – настройки, влияющие на процесс компиляции входных файлов.
- *filenames* – один или несколько файлов созданных на языке C/C++, ассемблере или объектных файлов.
- **--run_linker** настройки с которыми запускается линковщик
- *link_options* – настройки управления процессом линкования
- *object files* – имя дополнительного объектного файла для процесса линкования.

The screenshot shows the Code Composer Studio interface. The 'Project' menu is open, displaying options such as 'New...', 'Open...', 'Use External Makefile...', 'Export to Makefile...', 'Add Files to Project...', 'Save', 'Close', 'Source Control', 'Compile File' (with keyboard shortcuts Ctrl F7 and F7), 'Build', 'Rebuild All', 'Stop Build', 'Build Clean', 'Configurations...', 'Build Options...' (highlighted), 'File Specific Options...', 'Project Dependencies...', 'Show Project Dependencies', 'Show File Dependencies', 'Scan All File Dependencies', and 'Recent Project Files'. The background editor shows the source file 'x_CpuTimers.c' with the following content:

```
TI File $Revision: /main/3 $  
Checkin $Date: March 16, 2007 08:3  
#####  
  
FILE: DSP2833x_CpuTimers.c  
  
TITLE: CPU 32-bit Timers Initializ  
  
NOTES: CpuTimer1 and CpuTimer2 are  
other realtime operating sy  
  
Do not use these two timers  
on integrating DSP-BIOS or  
  
//  
//#####  
// $TI Release: DSP2833x Header Files V  
// $Release Date: September 26, 2007 $  
//#####
```





The image shows a development environment window titled 'Proba1.pjt' in 'Debug' mode. The left sidebar displays a file tree with 'DSP2833x_Cpi' selected. The main editor shows C code for a timer interrupt service routine. A 'Build Options for DSP2833x_CpuTimers.c in Proba1.pjt (...)' dialog box is open, showing the 'Compiler' tab with the following settings:

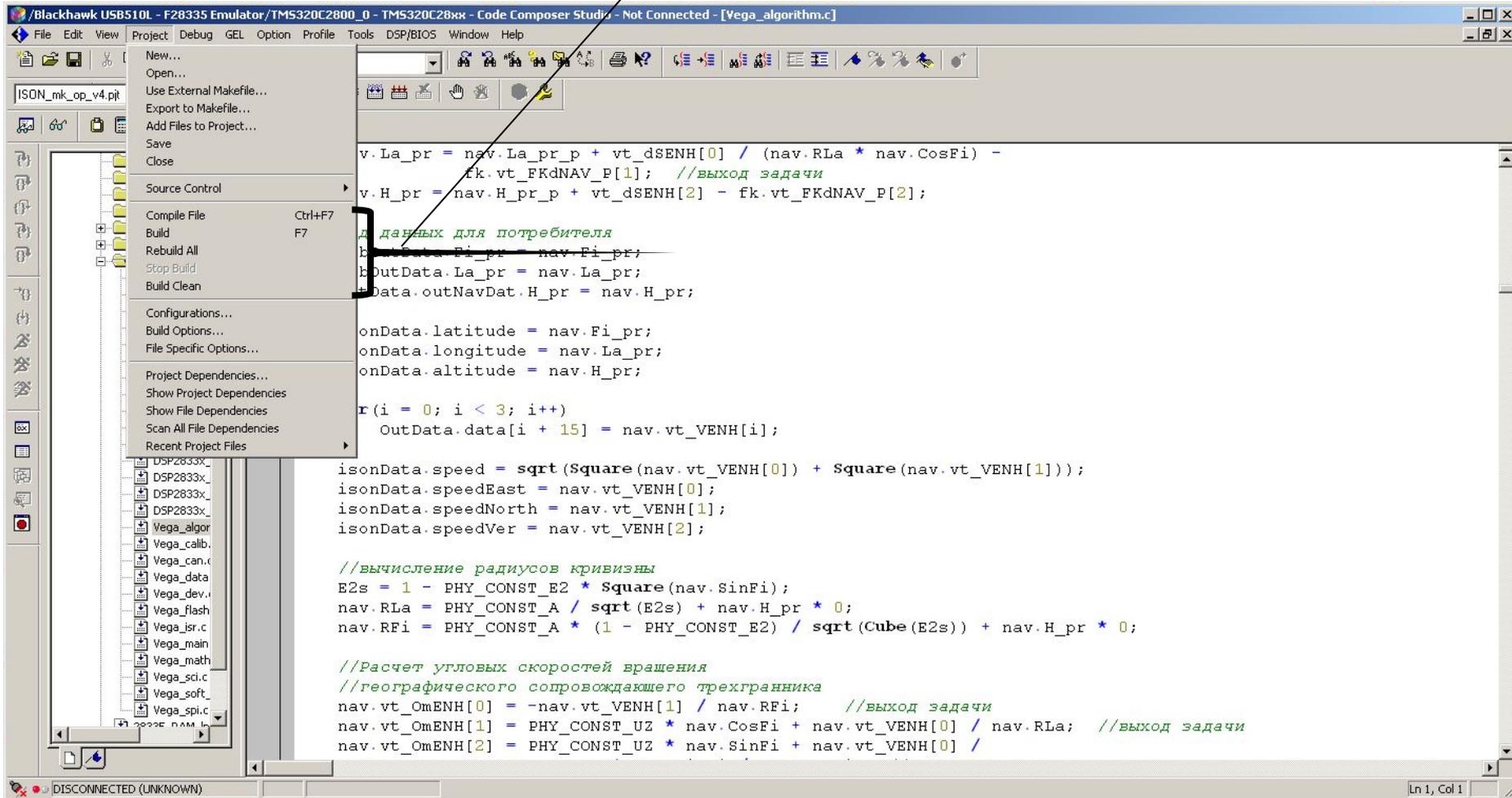
- Project level options: [Reset] [Display full options]
- Category: Basic
- Target Version: C28xx (-v28)
- Generate Debug Info: Full Symbolic Debug (-g)
- Optimize for Speed (-mf): No
- Opt Level: None
- Program Level Opt: None
- Specify CLA Support: None

The code in the background includes:

```
// This file is generated by the...  
// by the...  
// and the...  
// after...  
//  
void Conf...  
{  
    Uint32...  
  
    // In...  
    Timer...  
    Timer...  
    temp...  
    Timer...  
  
    // Se...  
    Timer...  
    Timer...  
  
    // In...  
    Timer...  
    Timer...  
    Timer...  
    Timer...  
    Timer...  
    Timer...  
  
    // Reset interrupt counter:  
    Timer->InterruptCount = 0;  
}
```

Построение проекта в CCS

инструменты компиляции проекта



```
//-----  
// This function initializes the selected tim  
// by the "Freq" and "Period" parameters. The  
// and the period in "uSeconds". The timer is  
// after configuration.  
//  
void ConfigCpuTimer(struct CPUTIMER_VARS *Tim  
{  
    Uint32 temp;  
  
    // Initialize timer period:  
    Timer->CPUFreqInMHz = Freq;  
    Timer->PeriodInUsec = Period;  
    temp = (long) (Freq * Period);  
    Timer->RegsAddr->PRD.all = temp;  
  
    // Set pre-scale counter to divide by 1 (  
    -----
```

File Edit View Project Debug GEL Option Profile Tools DSP/BIOS Window Help

Prob1.pjt Debug

Files

- GEL files
- Projects
 - Prob1.pjt (Deb)
 - Dependent Pro
 - Documents
 - DSP/BIOS Conf
 - Generated File:
 - Include
 - Libraries
 - Source

```

{
    Uint32 temp;

    // Initialize timer period:
    Timer->CPUFreqInMHz = Freq;
    Timer->PeriodInUsec = Period;
    temp = (long) (Freq * Period);
    Timer->RegsAddr->PRD.all = temp;

    // Set pre-scale counter to divide by 1 (SYSCLKOUT):
    Timer->RegsAddr->TPR.all = 0;
    
```

```

----- Prob1.pjt - Debug -----
[DSP2833x_CpuTimers.c] "C:\CCStudio_v3.3MCU\C2000\cgtools\bin\cl2000" -g -pds225 -fr"C:/CC
Warning: The project has no cmd file while the Text Linker is selected
[Linking...] "C:\CCStudio_v3.3MCU\C2000\cgtools\bin\cl2000" -@"Debug.lkf"
<Linking>
warning: creating output section ".ebss" without a SECTIONS specification
warning: creating output section ".reset" without a SECTIONS specification
warning: creating ".stack" section with default size of 0x400; use the -stack
option to change the default size

undefined      first referenced
symbol          in file
-----
_CpuTimer0Regs C:\\CCStudio_v3.3MCU\\MyProjects\\Prob1\\Debug\\DSP2833x_CpuTimers.obj
_CpuTimer1Regs C:\\CCStudio_v3.3MCU\\MyProjects\\Prob1\\Debug\\DSP2833x_CpuTimers.obj
_CpuTimer2Regs C:\\CCStudio_v3.3MCU\\MyProjects\\Prob1\\Debug\\DSP2833x_CpuTimers.obj
    
```

Code Composer Studio interface showing the File menu and a C code snippet for timer initialization.

File Menu:

- New
- Open... (Ctrl+O)
- Close
- Save (Ctrl+S)
- Save As...
- Save All
- Load Program... (Ctrl+L)**
- Reload Program (Ctrl+Shift+L)
- Load Symbols
- Reload Symbols
- Unload Symbols
- Load GEL...
- Data
- Workspace
- Difference between files...
- Merge Files...
- Print... (Ctrl+P)
- Recent Source Files
- Recent Workspaces
- Recent Program Files
- Recent Symbols
- Recent GEL Files
- Launch Setup
- Exit

Code Snippet:

```

{
    Uint32 temp;

    // Initialize timer period:
    Timer->CPUFreqInMHz = Freq;
    Timer->PeriodInUsec = Period;
    temp = (long) (Freq * Period)
    Timer->RegsAddr->PRD.all = temp;

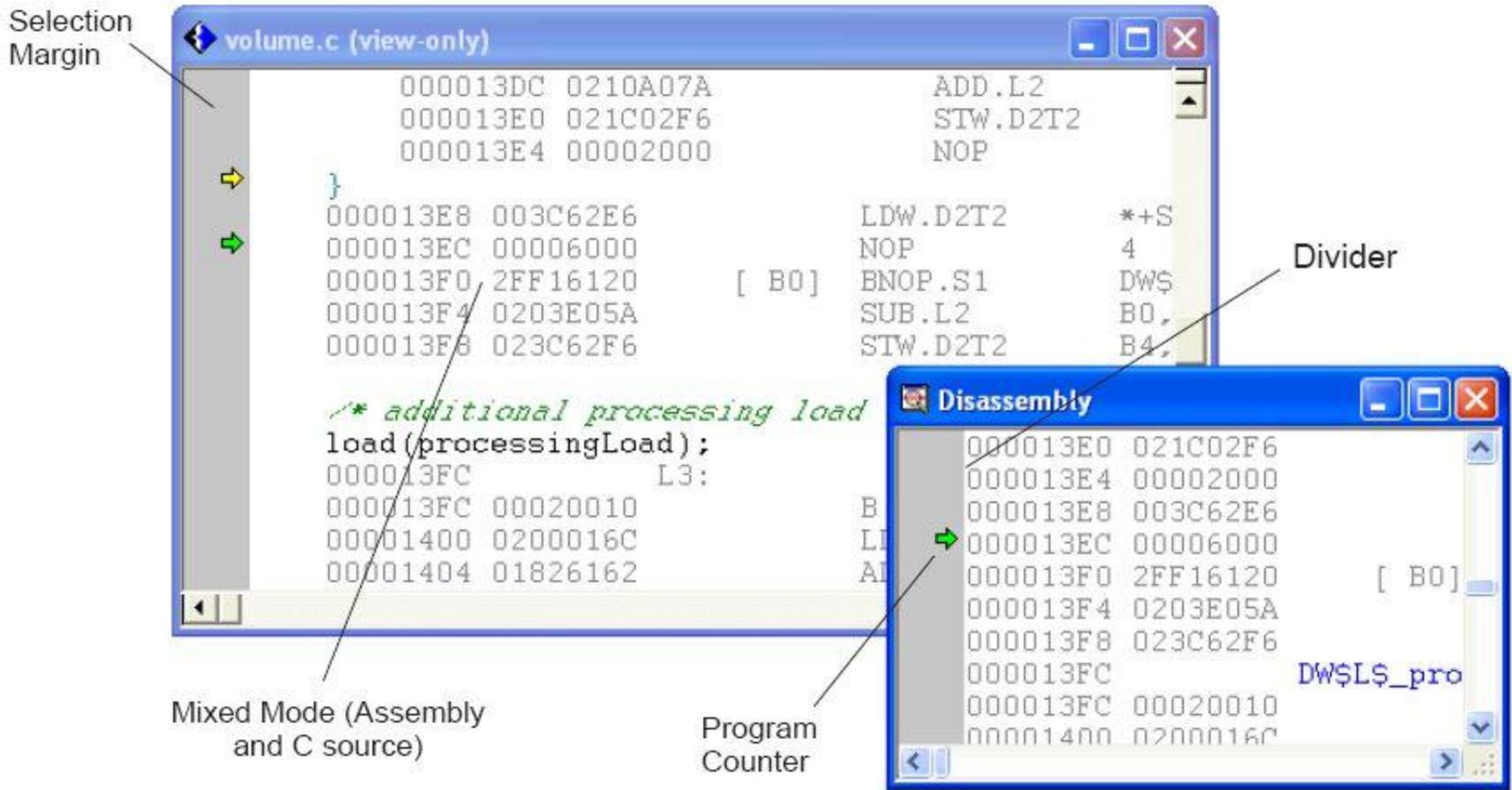
    // Set pre-scale counter to divide by 1000:
    Timer->RegsAddr->TPR.all = 0;
    Timer->RegsAddr->TPRH.all = 1000;

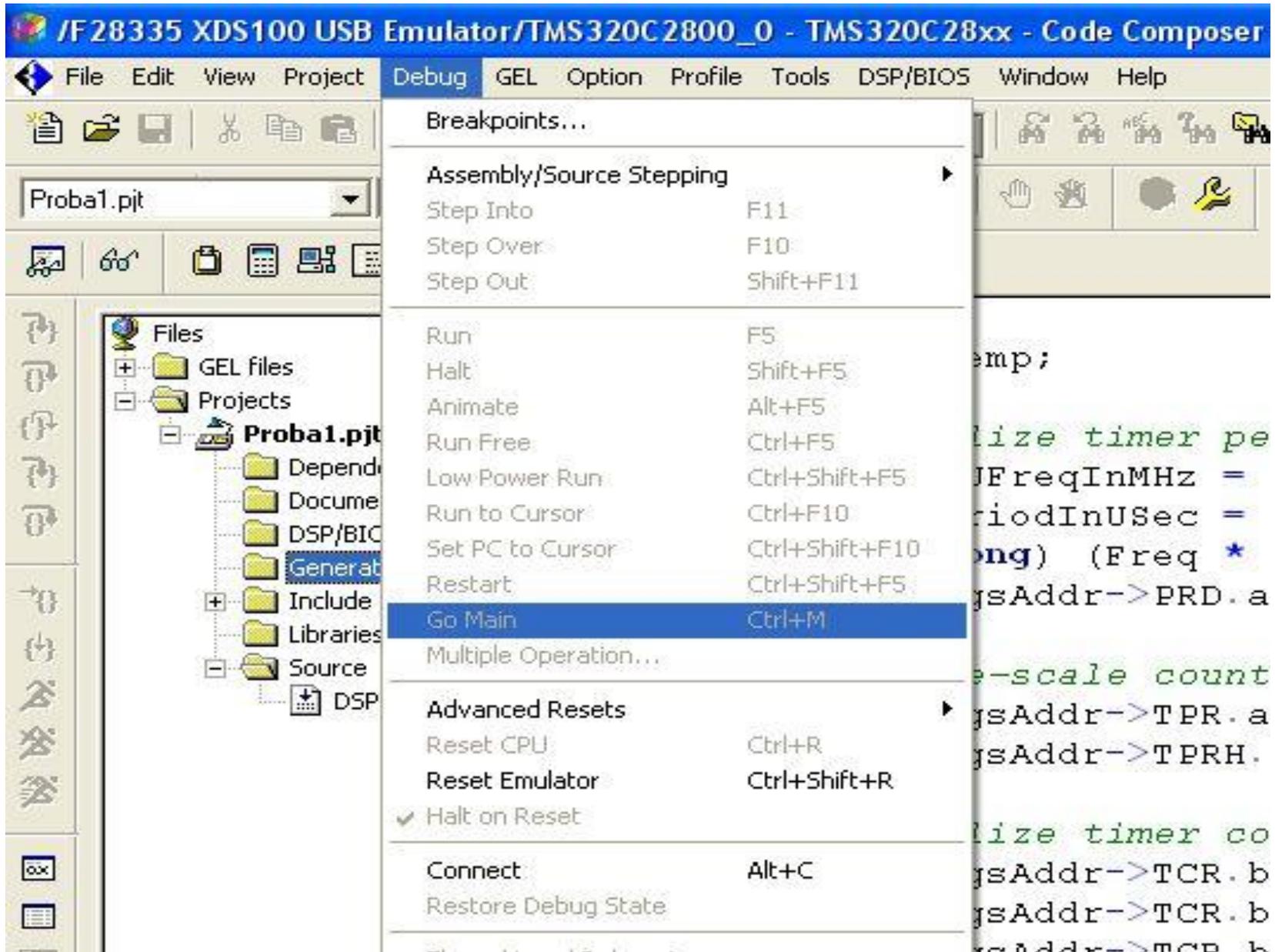
    // Initialize timer control register:
    Timer->RegsAddr->TCR.bit.TSS = 0;
    Timer->RegsAddr->TCR.bit.TRB = 0;
    Timer->RegsAddr->TCR.bit.SOFT = 0;
    Timer->RegsAddr->TCR.bit.FREE = 0;
    Timer->RegsAddr->TCR.bit.TIE = 0;

    // Reset interrupt counter:

```

Figure 4-8. Elements in the Source Code Window





TMS320C28x
CPU and Instruction Set
Reference Guide

Literature Number: SPRU430E
August 2001 – Revised January 2009

- 1 Обзор архитектуры**
- 2 Описание CPU**
- 3 Прерывание и перезапуск**
- 4 Конвейер**
- 5 Режимы адресации**
- 6 Команды ассемблера**
- 7 Особенности эмуляции**
- 8 Приложения**
- 9 Словарь**

Режимы адресации

1. Режим прямой адресации
2. Режим стековой адресации
3. Режим косвенной адресации
4. Режим регистровой адресации

Режим прямой адресации

AMODE	loc16/loc32 Syntax	Description	
0	@6bit	$32\text{bitDataAddr}(31:22) = 0$ $32\text{bitDataAddr}(21:6) = \text{DP}(15:0)$ $32\text{bitDataAddr}(5:0) = 6\text{bit}$	6-ти битное значение смещения объединяется со значением в регистре DP. Адресуется до 63 слов относительно базового адреса
1	@@7bit	$32\text{bitDataAddr}(31:22) = 0$ $32\text{bitDataAddr}(21:7) = \text{DP}(15:1)$ $32\text{bitDataAddr}(6:0) = 7\text{bit}$	Адресуется до 127 слов относительно базового адреса

Пример

MOVW DP,#VarA ; загружаем регистр DP значением страницы из переменной VarA

ADD AL,@VarA ; складываем значение из области памяти со смещением VarA,
;содержащимся в относительно DP с значением в регистре AL

MOV @VarB,AL ; Сохраняем результат в область памяти со смещением VarB

Режим стековой адресации

AMODE	loc16/loc32 Syntax	Description	
X	*SP++	32bitDataAddr(31:16) = 0x0000 32bitDataAddr(15:0) = SP if(loc16), SP = SP + 1 if(loc32), SP = SP + 2	
X	*- -SP	if(loc16), SP = SP - 1 if(loc32), SP = SP - 2 32bitDataAddr(31:16) = 0x0000 32bitDataAddr(15:0) = SP	

Пример

MOV *SP++,AL ; сохраняем содержимое 16-ти битного регистра AL в
; верхней части стека Push

MOVL *SP++,P ; сохраняем содержимое 32-ти битного регистра P в
; верхней части стека

Режим косвенной адресации

AMODE	loc16/loc32 Syntax	Description	
X	*XARn++	ARP = n 32bitDataAddr(31:0) = XARn if(loc16), XARn = XARn + 1 if(loc32), XARn = XARn + 2	
X	*--XARn	if(loc16), XARn = XARn - 1 if(loc32), XARn = XARn - 2 32bitDataAddr(31:0) = XARn	

Пример

```

MOVL XAR2,#Array1 ; Load XAR2 with start address of Array1
MOVL XAR3,#Array2 ; Load XAR3 with start address of Array2
MOV @AR0,#N-1     ; Load AR0 with loop count N
    
```

Loop:

```

MOVL ACC,*XAR2++ ; Load ACC with location pointed to by XAR2,
                  ; post-increment XAR2
MOVL *XAR3++,ACC ; Store ACC into location pointed to by XAR3,
                  ; post-increment XAR3
BANZ Loop,AR0--  ; Loop until AR0 == 0, post-decrement AR0
    
```

Режим регистровой адресации

AMODE	loc32 Syntax	Description
X	@ACC	Доступ к содержимому регистра ACC

Пример

```
MOVL XAR6,@ACC      ; Load XAR6 with contents of ACC
MOVL @ACC,XT        ; Load ACC with contents of XT register
ADDL ACC,@ACC       ; ACC = ACC + ACC
```

Типы ассемблерных команд

1. Арифметические операции:
 - сложение и вычитание – ADDB, ADD, SUBB, SBRK, ADDF32, SUBF32
 - умножение – MPYB, MPYU, SQRA, IMPYL, MPYF32
 - логические операции – AND, OR, XOR, NOT
 - инкремент и декремент – DEC, INC
 - операции сдвига – LSL, LSR, ASR, SFR
2. Операции работы с памятью
 - операции сохранения – MOV, MOVB, MOVW, MOVZ
 - операции работы со стеком – PUSH, POP
 - чтение и запись – PREAD, PWRITE
3. Операции вызова – LC, FFC
4. Операции цикла – LOOPZ, RPT
5. Операции условного и безусловного перехода – LB, B, BF
6. Операции с регистрами – EINT, DINT,
7. Операции преобразования типов данных – UI32TOF32, I16TOF32, F32TOUI32

ABS ACC

Absolute Value of Accumulator

SYNTAX OPTIONS	OPCODE	OBJMODE	RPT	CYC
ABS ACC	1111 1111 0101 0110	X	-	1

Operands **ACC** Accumulator register

Description The content of the ACC register is replaced with its absolute value:

```
if (ACC = 0x8000 0000)
  V = 1;
  If (OVM = 1)
    ACC = 0x7FFF FFFF;
  else
    ACC = 0x8000 0000;
else
  if (ACC < 0)
    ACC = -ACC;
```

Flags and Modes	N	After the operation, the N flag is set if bit 31 of the ACC is 1, else N is cleared.
	Z	After the operation, the Z flag is set if the ACC is zero, else Z is cleared.
	C	C is cleared by this operation.
	V	If (ACC = 0x8000 0000) at the start of the operation, this is considered an overflow value and V is set. Otherwise, V is not affected.
	OVM	If (ACC = 0x8000 0000) at the start of the operation, this is considered an overflow value, and the ACC value after the operation depends on the state of OVM: If OVM is cleared, ACC will be filled with 0x8000 0000. If OVM is set ACC will be saturated to 0x7FFF FFFF.

Repeat This instruction is not repeatable. If this instruction follows the RPT instruction, it resets the repeat counter (RPTC) and executes only once.

Example

```

; Take absolute value of VarA, make sure value is saturated:
MOVL  ACC,@VarA          ; Load ACC with contents of VarA
SETC  OVM                ; Turn overflow mode on
ABS   ACC                ; Absolute of ACC and saturate
MOVL  @VarA,ACC          ; Store result into VarA

```


TMS320C28x Assembly Language Tools v5.0.0

User's Guide

Literature Number: SPRU513C
October 2007

- 1 Введение в объектные модули**
- 2 Описание Ассемблера**
- 3 Директивы Ассемблера**
- 4 Макросы**
- 5 Линкование**
- 6 Листеры**
- 7 Преобразование в Нех код**
- 8 Ассемблер в Заголовочных
файлах**
- 9 Словарь**

Инструменты ассемблирования

- **Ассемблер** – транслирует файлы, написанные на языке ассемблера, в объектные файлы на машинном языке;
- **Архиватор** – собирает из группы файлов единый архивный файл, называемый библиотекой;
- **Линкер** – объединяет объектные файлы в один исполняемый объектный модуль (COFF-файл);
 - Формирователь листингов с абсолютными адресами;
 - Формирователь таблицы перекрестных ссылок;
 - Преобразователь COFF формата в ASCII-hex и другие форматы.

Формат объектного файла

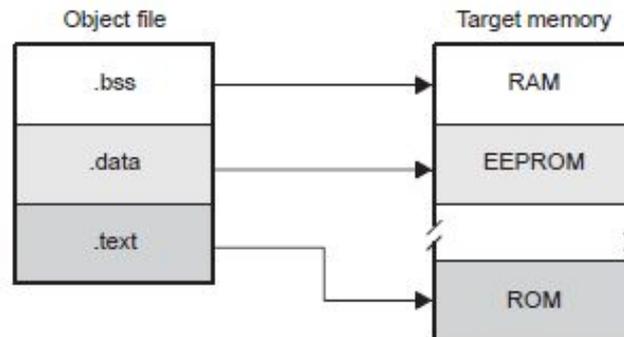
Секция – это блок кода или данных который занимает пространство в памяти контроллера и является наименьшей единицей объектного файла.

Основные секции:

- .text section – содержит исполняемый код
- .data section – содержит инициализированные данные
- .bss section – зарезервированное пространство под неинициализируемые переменные

Основные типы секций:

1. Инициализируемые - .text, .data и т.д.
2. Неинициализируемые - .bss, .ebss и.т.д



Директивы определения секции

Основные директивы создания секций

Неинициализируемые секции

- `.bss`
- `.usect`

Инициализируемые секции

- `.text`
- `.data`
- `.sect`

Пример создания секции

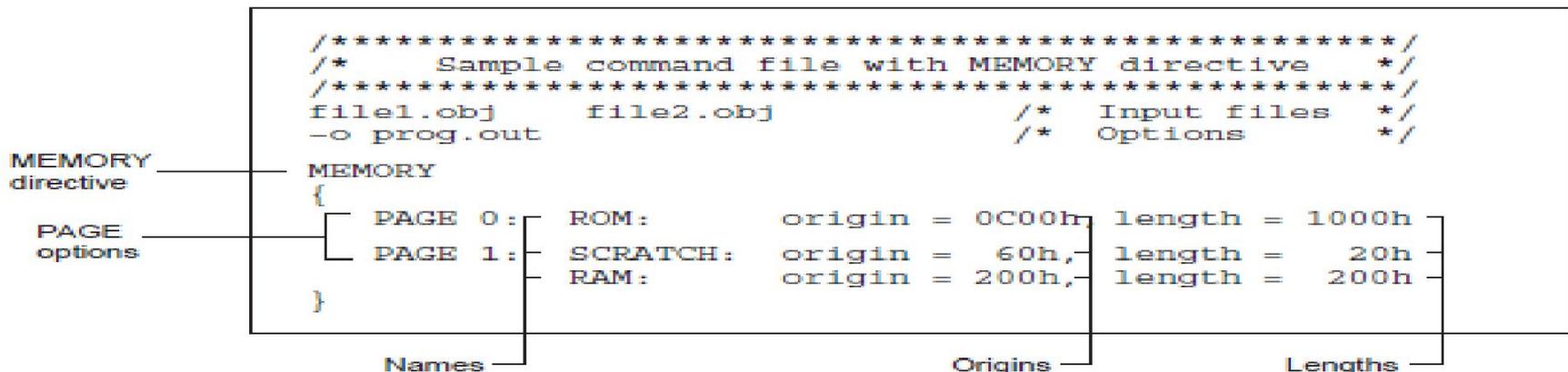
`.bss symbol, size in words [, blocking flag] [, alignment flag]`
`symbol .usect "section name", size in words [, blocking flag] [, alignment flag]`

Командный файл линковщика

Директива MEMORY

```
MEMORY
{
  [PAGE 0 : ] name [(attr) : origin = constant, length = constant[, fill = constant];
  [PAGE 1 : ] name [(attr) : origin = constant, length = constant[, fill = constant];
  .
  .
  .
  [PAGE n : ] name [(attr) : origin = constant, length = constant[, fill = constant];
}
```

Пример



Командный файл линковщика

Директива SECTION

```
SECTIONS
{
  name : [property [, property] [, property] . . . ]
  name : [property [, property] [, property] . . . ]
  name : [property [, property] [, property] . . . ]
}
```

Свойства:

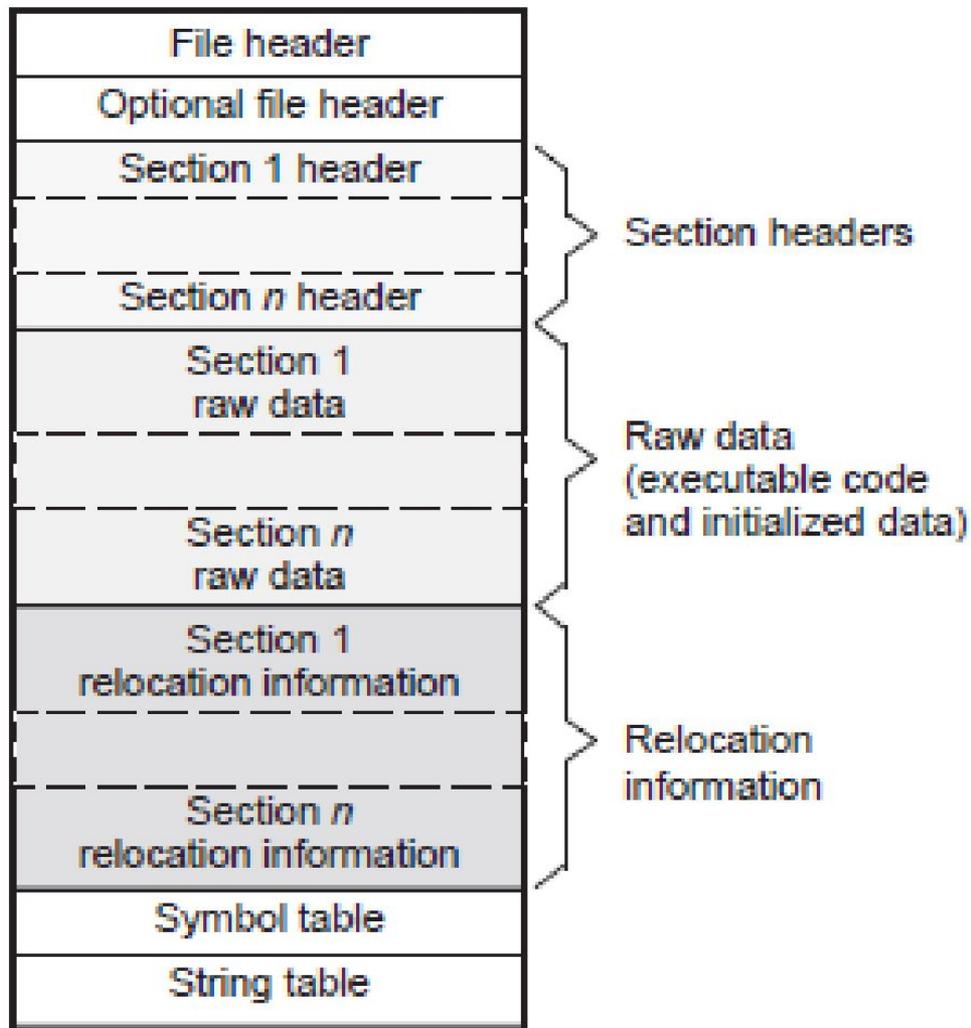
1. Область размещения секции:
load =(>) ОБЛАСТЬ ПАМЯТИ
> ОБЛАСТЬ ПАМЯТИ
2. Область запуска секции:
run =(>) ОБЛАСТЬ ПАМЯТИ
3. Входные секции
{Входные секции}

Пример

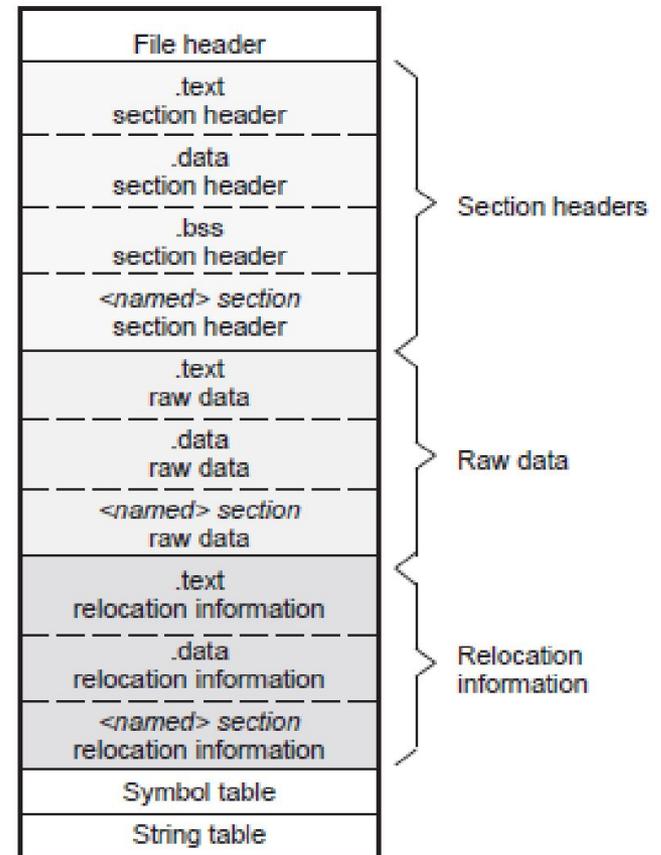
```
SECTION directive — SECTIONS
{
  .text:      load = ROM, run = 800h
  .const:     load = ROM
  .bss:       load = RAM
  .vectors:   load = 0FF80h
  {
    t1.obj(.intvec1)
    t2.obj(.intvec2)
    endvec = .;
  }
  .data:     align = 16
}
```

Section specifications —

Структура COFF - файла



Пример



Структура заголовка COFF - файла

Byte Number	Type	
0-1	Unsigned short	– идентификатор версии файла
2-3	Unsigned short	– количество секций в файле
4-7	Long	– дата и время создания файла
8-11	Long	– указатель на начальный адрес таблицы символов
12-15	Long	– количество таблиц символов
16-17	Unsigned short	– размер в байтах заголовка с настройками
18-19	Unsigned short	– служебные флаги
20-21	Unsigned short	– идентификатор устройства

Структура дополнительного заголовка COFF - файла

Byte Number	Type	
0-1	Short	– слово начала заголовка (0x0108)
2-3	Short	– версия штампа
4-7	Long	– размер в байтах исполняемого кода
8-11	Long	– размер в байтах инициализированных данных
12-15	Long	– размер в байтах неинициализированных данных
16-19	Long	– точка входа
20-23	Long	– стартовый адрес исполняемого кода
24-27	Long	– стартовый адрес инициализированных данных

Структура секционного заголовка COFF - файла

Byte Number	Type	
0-7	Character	– идентификатор секции
8-11	Long	– физический адрес секции
12-15	Long	– размер в байтах исполняемого кода
16-19	Long	– размер секции в байтах
20-23	Long	– файловый указатель к начальным данным
24-27	Long	– файловый указатель на точку входа
28-31	Long	– резерв
32-35	Unsigned long	– число точек входа
36-39	Unsigned long	– резерв
40-43	Unsigned long	– служебные флаги
44-45	Unsigned short	– резерв
46-47	Unsigned short	– номер страницы памяти

Утилита преобразования объектного файла

```
hex2000 [options] filename
```

Типы опций:

основные опции: -map – генерируется файл карты памяти

-o – определяется выходной файл

опции отображения: -fill – определяется значение для заполнения пустот

-image – устанавливается режим отображения

опции памяти: -memwidth – определяется разрядность памяти системы

-romwidth – устанавливается разрядность ПЗУ устройства

опции выходного формата: -a – формат ASCII-Hex

-i – формат Intel

-m – формат Motorola-S

Пример

```
hex2000.exe -i -romwidth 16 -o .\Release\MK_COI_release_ver1.hex  
.\Release\MK_COI_release_ver1.out
```

TMS320C28x Optimizing C/C++ Compiler v5.0.0

User's Guide

Literature Number: SPRU514C
September 2007

- 1 Использование компилятора**
- 2 Оптимизация кода**
- 3 Линкование C/C++ кода**
- 4 Оптимизация после линкования**
- 5 Реализация C/C++ в 28х**
- 6 Среда реального времени**
- 7 Функции и библиотека для поддержки реального времени**
- 8 Словарь**

Уровни оптимизации

Уровень оптимизации 0

- производится упрощение графа управляющей логики программы
- переменные размещаются в регистрах
- исключается неиспользуемый код
- упрощаются выражения и операторы
- раскрывается вызов функций, объявленных inline
- производится чередование циклов

Уровень оптимизации 1

- производится локальное копирование/константное размножение
- удаляются неиспользуемые присваивания
- исключаются локальные обобщенные выражения

Уровень оптимизации 2

- производится оптимизация циклов
- исключаются глобальные общие подвыражения
- исключаются глобальные неиспользуемые присваивания
- производится разворачивание циклов

Уровень оптимизации 3

- удаляются невызываемые функции
- inline вызов маленьких функций
- переупорядочивание деклараций в функциях; атрибуты вызываемой функции известны, когда вызов оптимизирован
- распространяет аргументы в тела функций когда все вызовы передают ту же самую величину в той же позиции аргумента
- упрощаются функции с возвращаемым значением, которое нигде не используется
- определяются характеристики переменных на уровне файла
- производится разворачивание циклов

**Компилятор поддерживает
стандарты языка Международной
организации стандартизации:
C -1989 года;
C++ - 1998 года**

Поддерживаемые типы данных

Type	Size	Representation	Range	
			Minimum	Maximum
char, signed char	16 bits	ASCII	-32 768	32 767
unsigned char	16 bits	ASCII	0	65 535
short	16 bits	2s complement	-32 768	32 767
unsigned short	16 bits	Binary	0	65 535
int, signed int	16 bits	2s complement	-32 768	32 767
unsigned int	16 bits	Binary	0	65 535
long, signed long	32 bits	2s complement	-2 147 483 648	2 147 483 647
unsigned long	32 bits	Binary	0	4 294 967 295
long long, signed long long	64 bits	2s complement	-9 223 372 036 854 775 808	9 223 372 036 854 775 807
unsigned long long	64 bits	Binary	0	18 446 744 073 709 551 615
enum	16 bits	2s complement	-32 768	32 767
float	32 bits	IEEE 32-bit	1.19 209 290e-38 ⁽¹⁾	3.40 282 35e+38
double	32 bits	IEEE 64-bit	1.19 209 290e-38 ⁽¹⁾	3.40 282 35e+38
long double	64 bits	IEEE 64-bit	2.22 507 385e-308 ⁽¹⁾	1.79 769 313e+308
pointers	32 bits	Binary	0	0xFFFF
far pointers	22 bits	Binary	0	0x3FFFFFF

Размещение данных в памяти

Размещение данных

```
#pragma DATA_SECTION ( symbol , " section name " );
```

Пример

```
#pragma DATA_SECTION ( SysOpt, "my_data" );  
struct NAVIGATION_DATA SysOpt;
```

```
#pragma DATA_SECTION ( dT, "my_data" );  
float dT;
```

```
#pragma DATA_SECTION ( canbRxBuffer, "can_data" );  
Uint16 canbRxBuffer [ CANB_RECEIVE_BUFFER_SIZE ];
```

Размещение кода в памяти

Размещение данных

```
#pragma CODE_SECTION ( symbol , " section name " );
```

Пример

```
#pragma CODE_SECTION(CanCalcCrc, "my_code1");
Uint16 CanCalcCrc(Uint16 *pData, Uint16 nData)
{
    Uint32 sumCAN;
    Uint16 i, sum16;

    for (i = 0, sumCAN = 0; i < nData; i++)
    {
        sumCAN += pData[i];
    }
    sumCAN = (sumCAN & 0x0000FFFF) + (sumCAN >> 16);
    sum16 = (Uint16)((sumCAN & 0x0000FFFF) + (sumCAN >> 16));

    return sum16;
}
```