Государственный университет телекоммуникаций им. Бонч-Бруевича
кафедра «Цифровой обработки сигналов»

И. И. Гук,
доцент кафедры ЦОС
gook_igor@mail.ru

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

к лабораторной работе № 4

«Реализация файловой модели КИХ фильтра в интегрированной среде разработки CCS»

2008 год

Оглавление

***Замечание**. Все приведенные материалы относятся к использованию интегрированной отладочной среды Code Composer Studio (CCS) версии 3.x. При использовании CCS версии ниже 3.x необходимо воспользоваться материалами статей «Реализация алгоритмов ЦОС на ассемблере TMS320C6000», «Оптимизация программного кода для ЦСП TMS320C6000» и «Тестирование программного кода для ЦСП TMS320C6000» напечатанной в журнале «Компоненты и Технологии» в 2007 году. Электронные версии данных статей и дополнительные материалы к ним можно найти на по адресу:*

http://www.scanti.ru/univ_stati.html

1. Начальная настройка интегрированной среды разработки Code Composer Studio

Начальная настройка интегрированной среды разработки (ИСР) *Code Composer Studio* (CCS) производиться только один раз при ее первом запуске. После установки CCS на рабочем столе будет две иконки 

и 

(аналогичные иконки есть и в меню *СТАРТ* операционной системы Windows). Необходимо запустить программу начальной настройки CCS, нажав на значок, подписанный как «*Setup Code Composer Studio*». Появиться окно, показанное на рис.1.

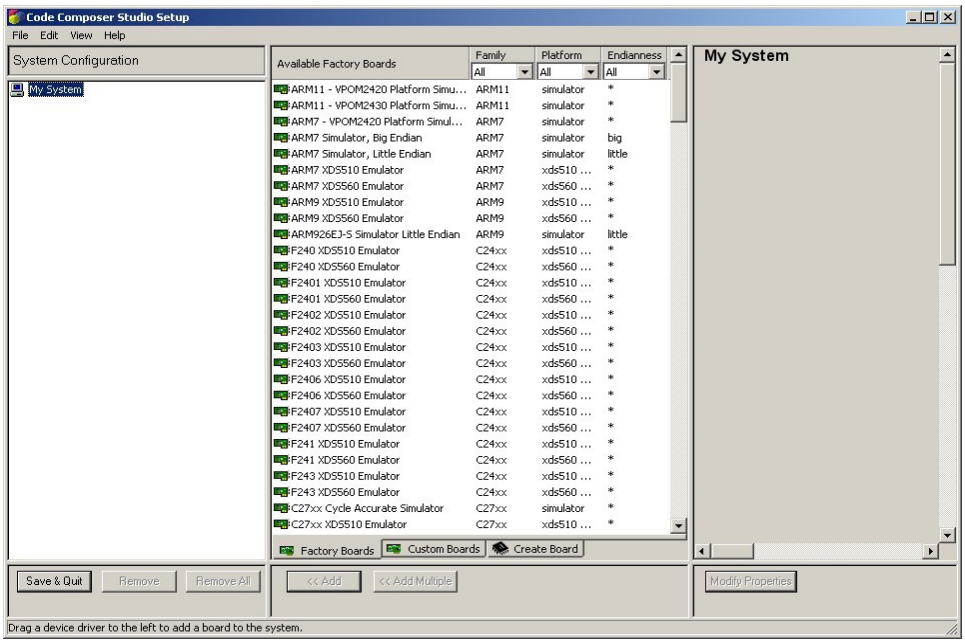

Рис.1. Вид окна начальной настройки программы CCS.

В окне «*Available Factory Boards*» необходимо выбрать тип процессора и способ подключения. Для облегчение поиска можно воспользоваться фильтрами «*Family*», «*Platform*» и «*Endianness*». Вид окна начальной настройки с выбранными параметрами фильтров показан на рис.2.

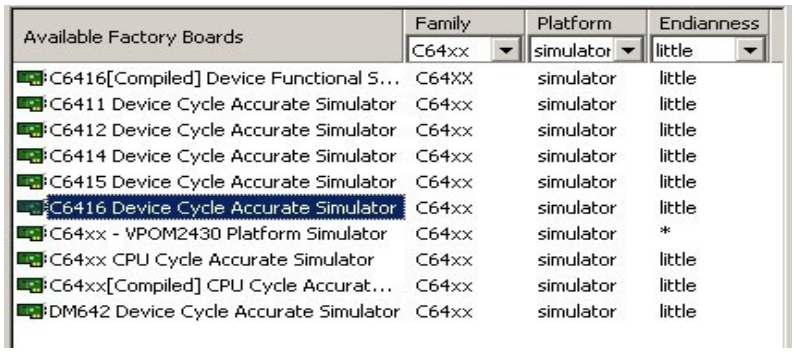

Рис.2. Выбор типа процессора и способа подключения.

Фильтры необходимо настроить следующим образом:

- «*Family*» — определяет тип процессора, выбираем «*C64xx*»,
- «*Platform*» — определяет способ подключения, выбираем «*simulator*», что означает работу не с реальной платой, а с программной моделью процессора, то есть подключаемся к «симулятору»,
- «*Endianness*» — определяет чередования старшего и младшего байтов в 32 разрядном слове данных, необходимо выбрать «*little*».

Затем необходимо выбрать «*C6416 Device Cycle Accurate Simulator*» и нажать внизу кнопку «*Add*». Окно начальной настройки примет вид, показанный на рис.3

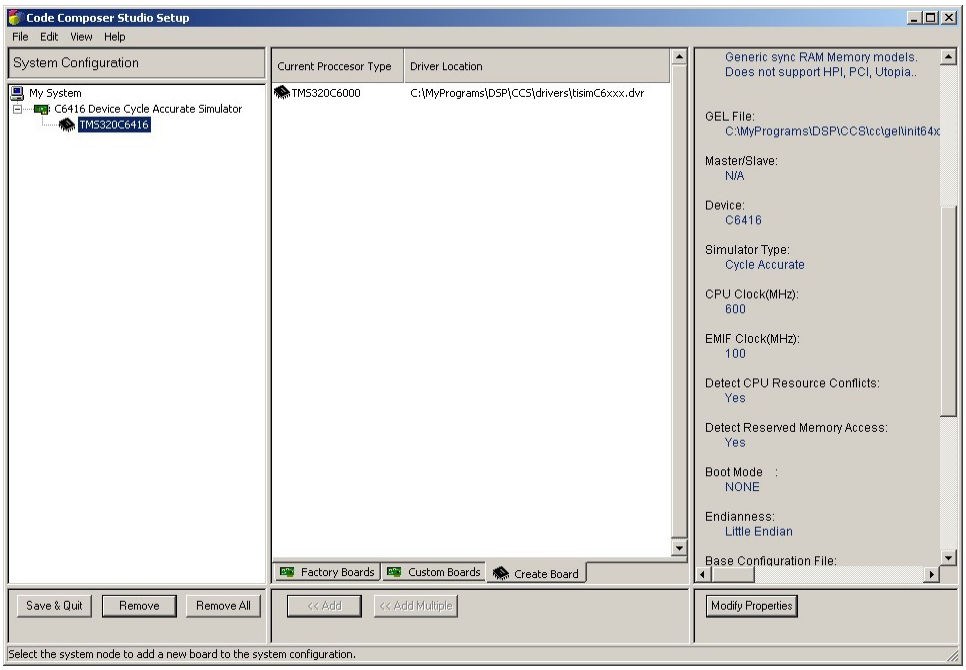

Рис.3. Настройка параметров запуска Code Composer Studio.

Параметры запуска отладочной среды CCS заданы. Для запуска среды необходимо нажать кнопку «*Save & Quit*». Появиться окно запроса запуска CCS, показанное на рис.4, здесь необходимо подтвердить запуск, нажав кнопку «*Да*».

Рис.4. Подтверждение запуска Code Composer Studio.

Через некоторое время запуститься CCS и появиться окно, показанное на рис.5.

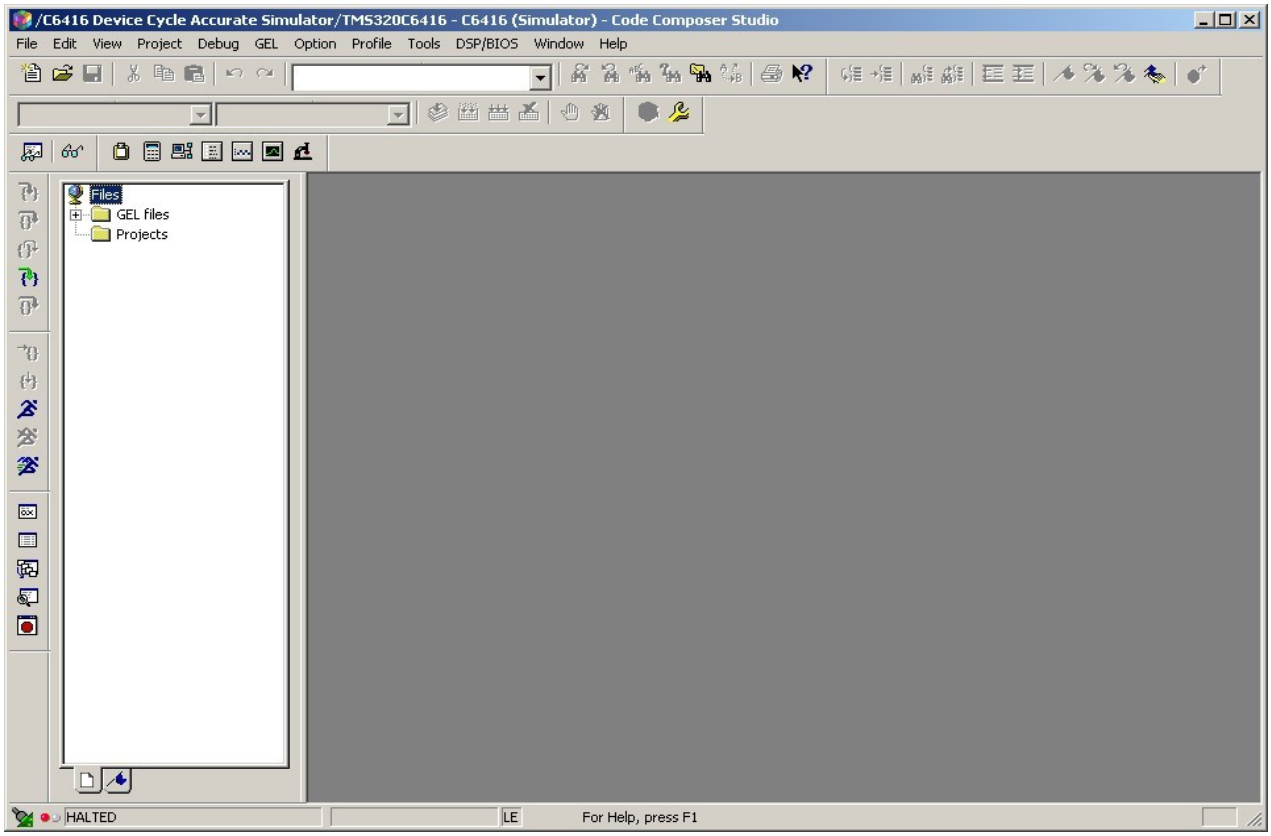

Рис.5. Code Composer Studio.

2. Создание проекта

Для создания нового проекта в среде CCS необходимо в главном меню программы выбрать пункт «*Project->New...*». Как это показано на рис.6.

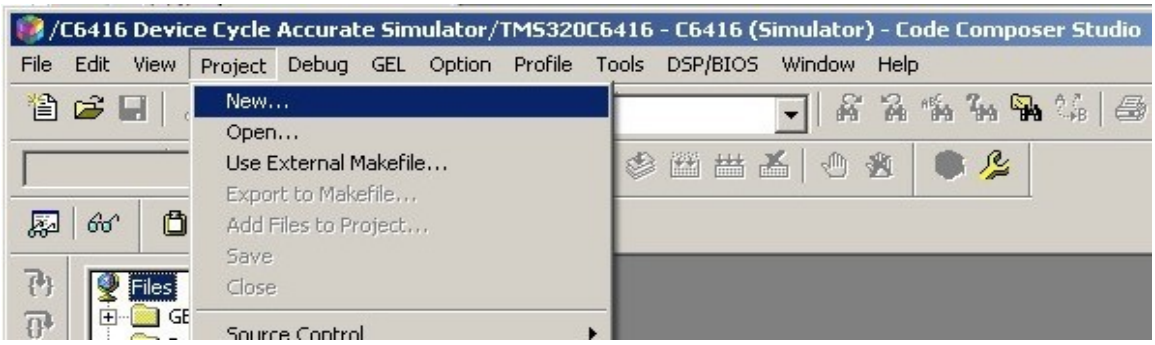

Рис.6. Создание нового проекта в CCS.

Появиться окно «*Project Creation*», где задаются основные параметры проекта. В начале в поле «*Location*» определяется место расположения папки проекта, как отмечалось в предыдущих работах, желательно выбрать место на FLASH-накопителе студента. Затем определяется имя проекта в поле «*Project Name*». В рассматриваем примере — это «*prjKIX_CCS*». Проверьте, что бы параметры в полях «*Project Type*» и «*Target*» соответствовали показанным на рис.7. Обычно их менять не требуется.

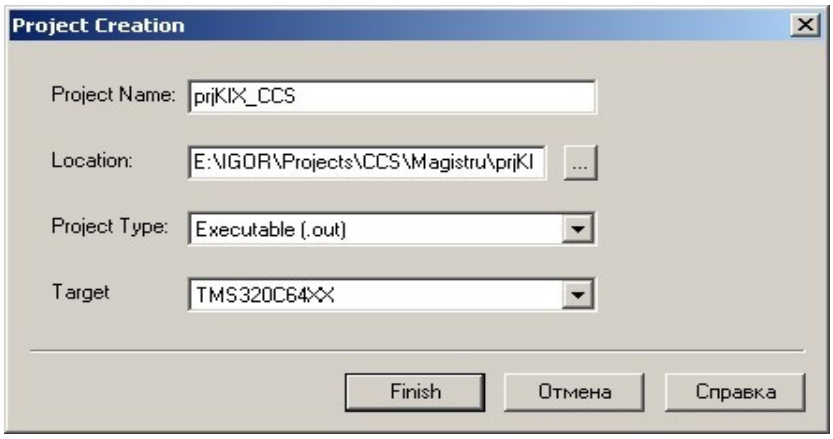

Рис.7. Настройки создаваемого проекта.

Обратите внимание, что в случае использования CCS нет необходимости предварительно, до создания проекта, создавать папку для его хранения. При определении имени проекта папка создается автоматически. Для завершения создания проекта нажмите кнопку «*Finish*». Вид CCS после создания проекта показан на рис.8.

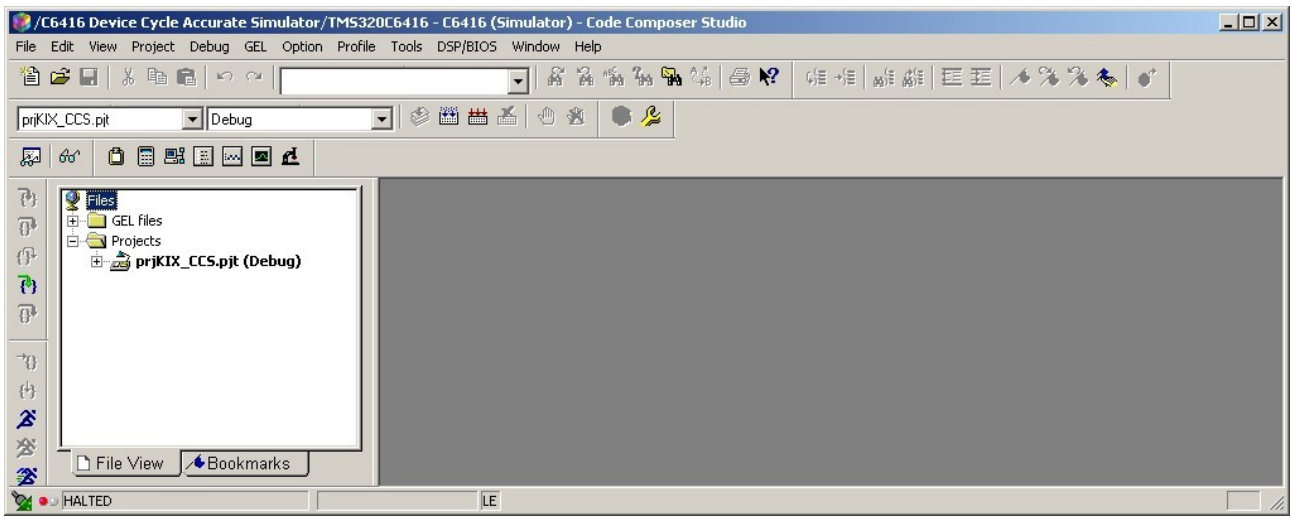

Рис.8. Вид программы CCS после после процедуры создания проекта.

Теперь необходимо скопировать в папку проекта файлы, созданные на предыдущих занятиях, а именно — «*main.cpp*», «*cnvKIX.cpp*», «*runKIX.cpp*», «*initKIX.cpp*», «*constant.cpp*» и «*prjKIX.h*».

При создании кода для цифровых сигнальных процессоров (ЦСП) необходимо распределить память процессора. В семейства TMS320C6000 для этого существует так называемый механизм секций: все функции и данные размещаются в поименованных секциях, а сами секции размещаются в различных областях памяти. Существует стандартный набор секций, который должен быть определен всегда. Однако можно назначить дополнительные секции и разместить в них код функций и данные разработанного алгоритма. Реализуется данный механизм при помощи файла «*standard.cmd*». В лабораторной работе используется стандартное распределение памяти и стандартный набор секций для режима симуляции (то есть при использовании программной модели процессора).

Для рассматриваемого примера файл «*standard.cmd*» можно создать следующим образом. В главном меню CCS выбрать пункт «*File->New->Source File*», как это показано на рис.9.

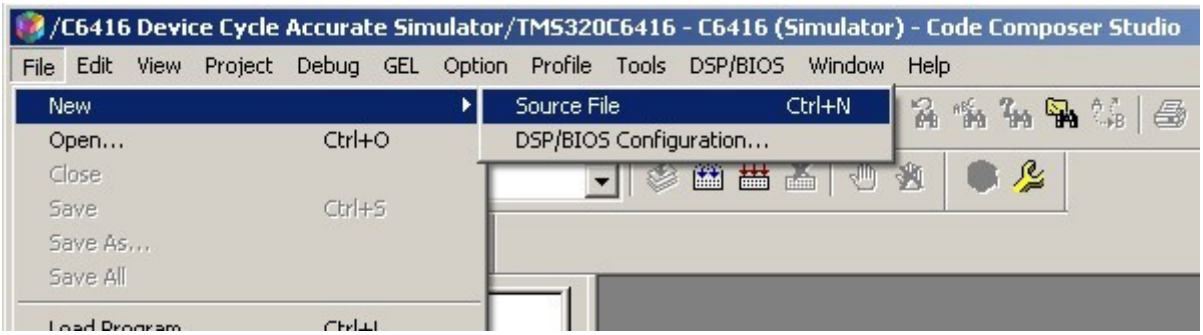

Рис.9. Создание новых файлов.

В CCS появиться окно для редактирования нового файла с именем «*Untitled1**». В нем необходимо набрать требуемый текст. Вид CCS с окном редактирования нового файла и набранным в нем текстом показан на рис.10. Листинг файла «*standard.cmd*» приведен в приложении 1.

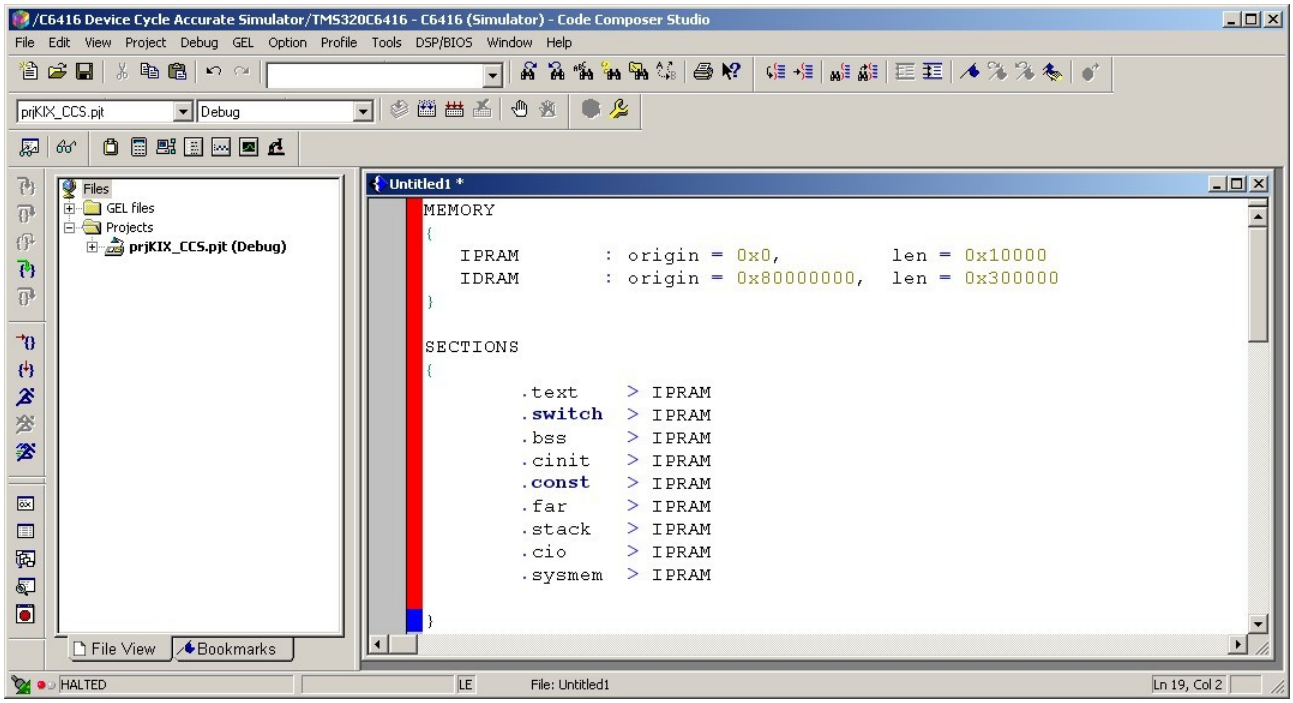

Рис.10. Подключение файлов к проекту.

В блоке MEMORY указаны имена областей памяти, их начальный адрес и длина. В блоке SECTIONS – имена секций и область памяти, где они расположены. Наберите этот текст и сохраните файл. Для чего в главном меню CCS выбрать пункт «*File->Save As...*», как это показано на рис.11.

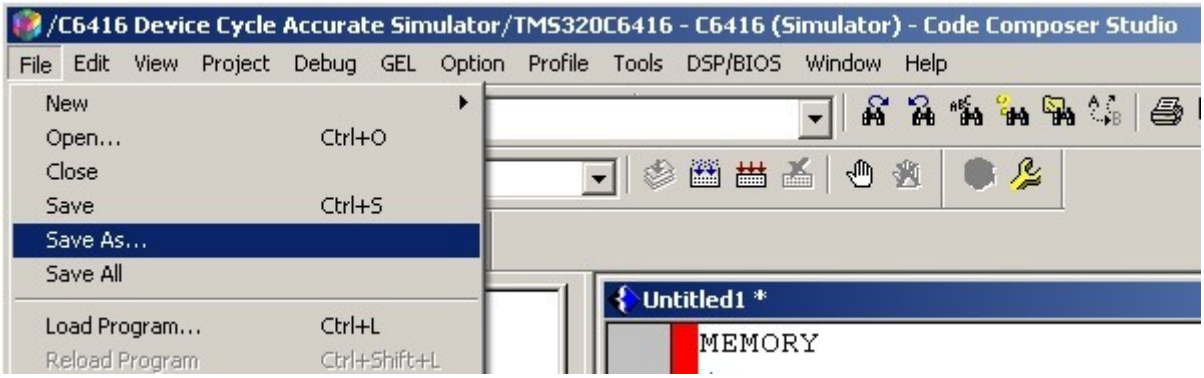

Рис.11. Подключение файлов к проекту.

Появиться окно «*Сохранить как*», где необходимо указать имя файла «*standard.cmd*» (см. рис.12) и нажать кнопку «*Сохранить*».

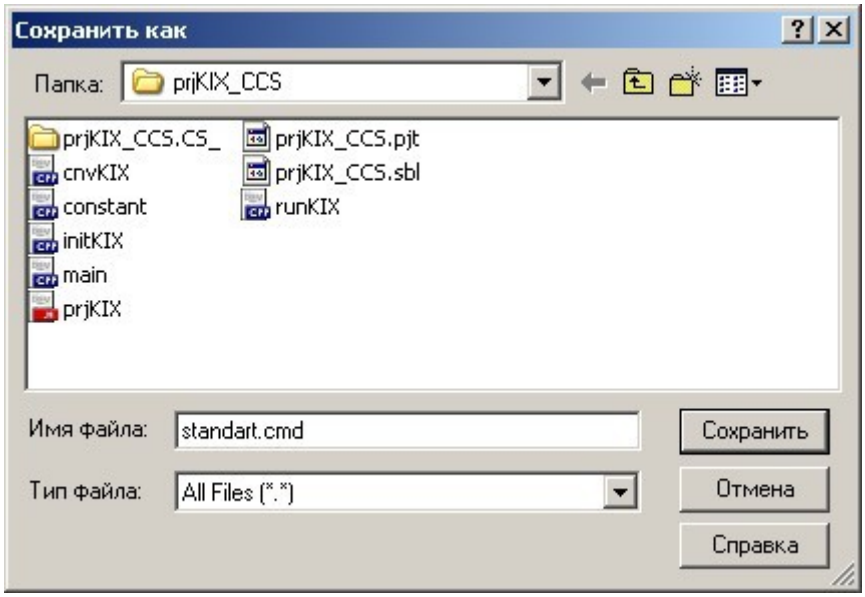

Рис.12. Подключение файлов к проекту.

Затем необходимо подключить к проекту созданный файл. Это можно сделать через главное меню CCS выбрав пункт «*Project->Add Files to Project...*», как это показано на рис.13.

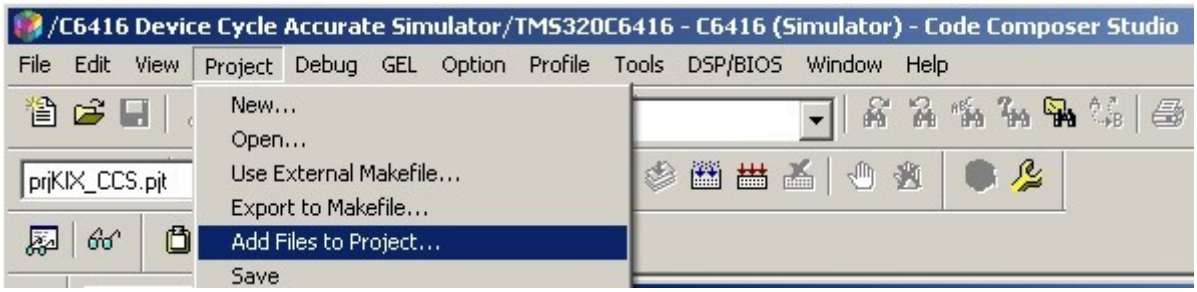

Рис.13. Подключение файлов к проекту.

Появиться окно «*Add Files to Project*» (см.рис.14). Здесь в поле «*Папка*» необходимо выбрать папку, где расположен проект. В поле «*Тип файла*» выбрать из выпадающего списка позицию «*Linker Command File (*.cmd, *.lcf)*». В поле выбора файлов должен остаться только файл «*standard*». Щелкаем на нем мышкой. Имя выбранного файла должно появиться в поле «*Имя файла*». Затем нажимаем кнопку «*Открыть*». Все, файл подключен к проекту.

Аналогичным образом подключают файлы «*main.cpp*», «*cnvKIX.cpp*», «*runKIX.cpp*», «*initKIX.cpp*», «*constant.cpp*». При этом в поле «*Тип файла*» необходимо выбрать — «*C++ Source Files (*.cpp, *.cc, *.cxx)*». Обратите внимание, что файл «*prjKIX.h*» к проекту не подключается. В этом нет необходимости, так как в CCS подключение заголовочных файлов, которые в файлах проекта указаны директивой «*#include*», происходит автоматически.

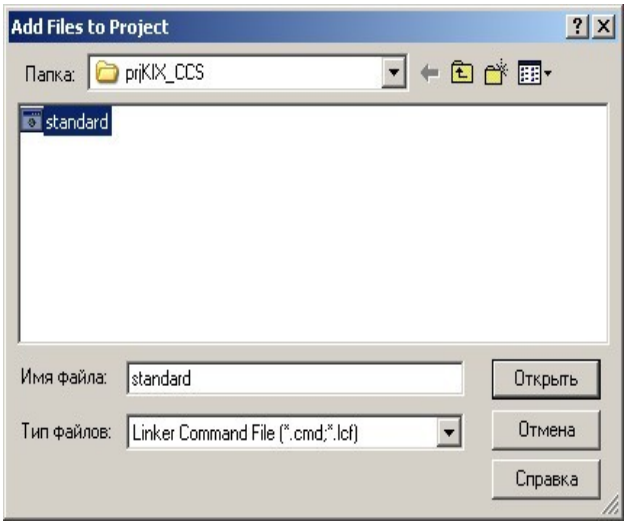

Рис.14. Выбор подключаемых файлов.

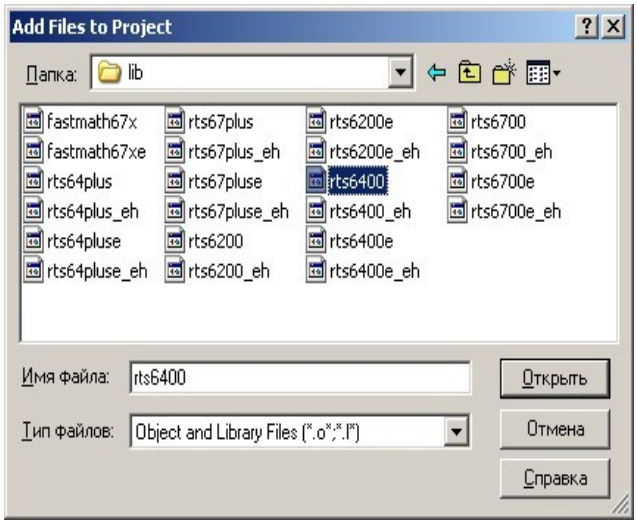

Рис.15. Выбор подключаемой библиотеки.

И последний файл, который подключается к проекту — это файл библиотеки «*rts6400.lib*». Этот файл необходим для того, что бы CCS смог корректно преобразовать C++ кода к бинарному виду для выбранного типа сигнального процессора. Подключение производиться так же как это было рассмотрено выше. Необходимый файл находиться в директории, где установлена отладочная среда CCS, в папке «*Директория установки CCS\C6000\cgtools\lib*». Выберите папку размещения файла библиотеки в поле «*Папка*», Укажите в поле «*Тип файла*» — «*Object and Library Files (*.o; *.l)*», Щелкните по имени «*rts6400*» (после этого оно должно появиться в поле «*Имя файла*») и нажмите кнопку «*Открыть*». Окно «*Add Files to Project*» для случая подключения библиотеки «*rts6400.lib*» показано на рис.15.

Вид программы CCS после всех операций по подключению файлов показан на рис.16. Обратите внимание на закладку «*File View*» с правой стороны окна CCS, В ней представлена файловая структура проекта. Для просмотра дерева файлов необходимо нажать на значок «+» напротив выбранного пункта. Например, в данном случае необходимо нажать на «+» напротив пунктов «*prjKIX_CPP (Debug)*», «*Libraries*» и «*Source*».

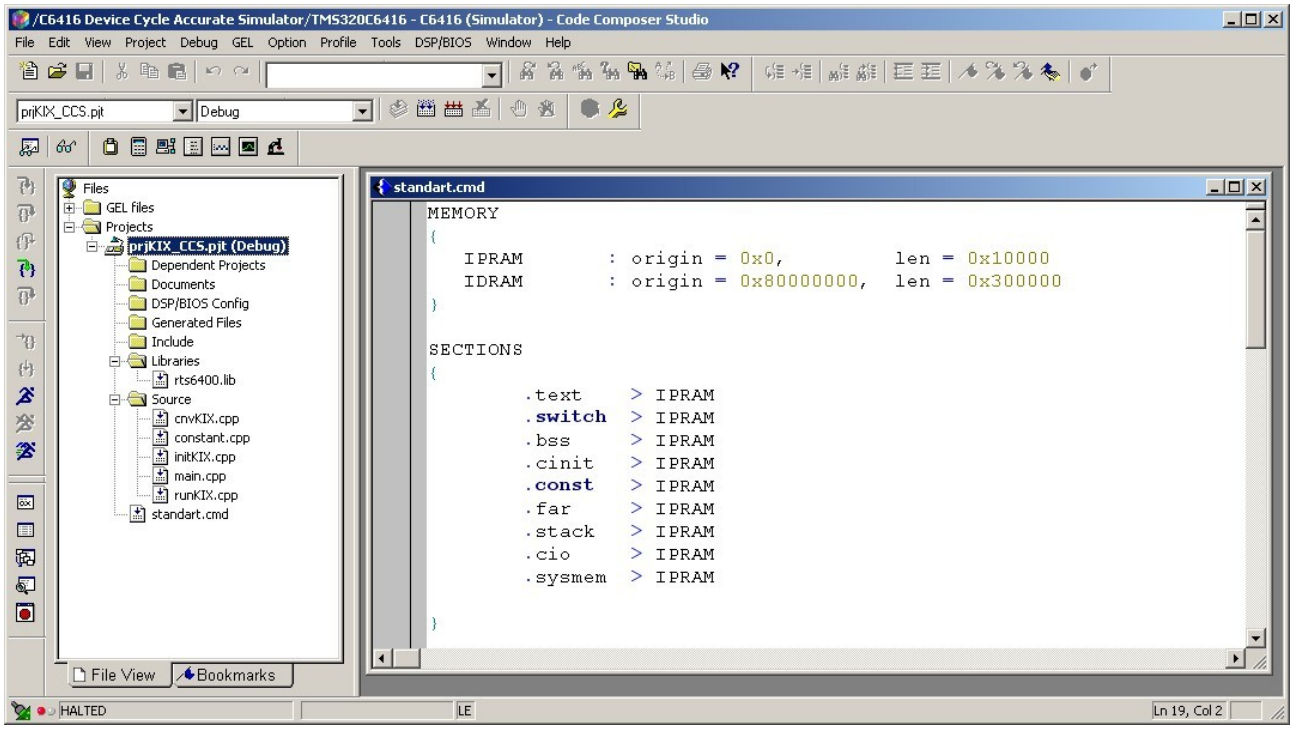

Рис.16. Вид CCS после подключения всех необходимых файлов.

Последний этап создания проекта — это уточнение его настроек. Для выполнения этого необходимо в главном меню CCS выбрать пункт «*Project->Build Options...*», как это показано на рис.17.

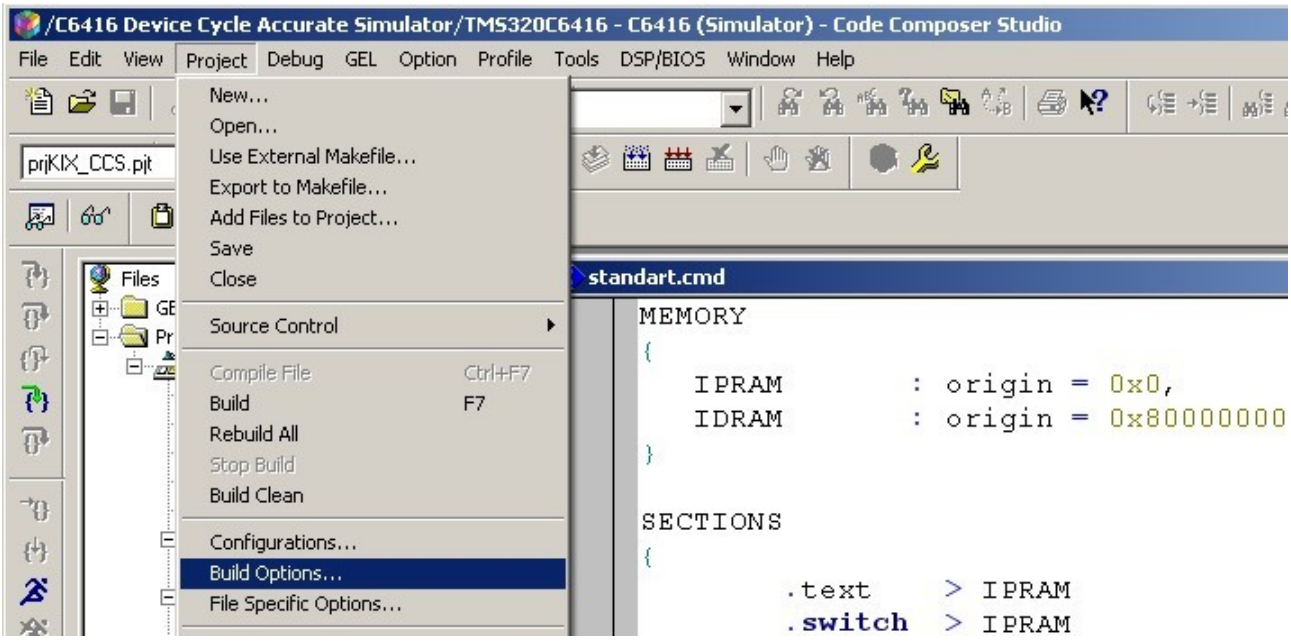

Рис.17. Уточнение настроек проекта.

В появившемся окне настроек проекта «*Build Options for prjKIX_CCS.prj (Debug)*» перейти на закладку «*Compiler*» и уточнить настройки пунктов «*Basic*» и «*Advanced*» в разделе «*Category*» так как это показано на рис.18, рис.19.

В пункте «*Basic*» обратите внимание на выставление опции «*Full Symbolic Debug (-g)*» в окне «*General Debug Info*», а так же на соответствие типа процессора заданному при начальной настройки отладочной среды в окне «*Target Version*».

В пункте «*Advanced*» необходимо в окне «*Endianness*» установить опцию «*Little Endian*», а в «*Memory Models*» — значение «*Far Aggregate*».

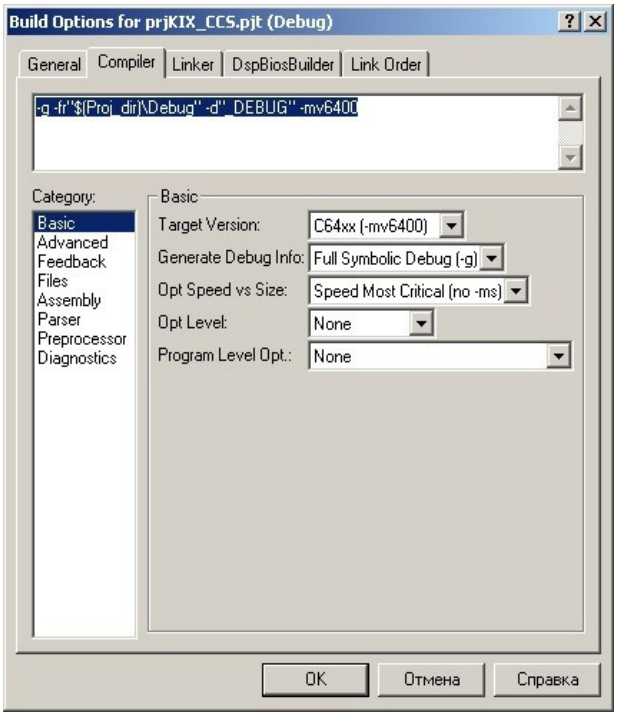

Рис.18. Настройки категории «Basic» на закладке «Compiler».

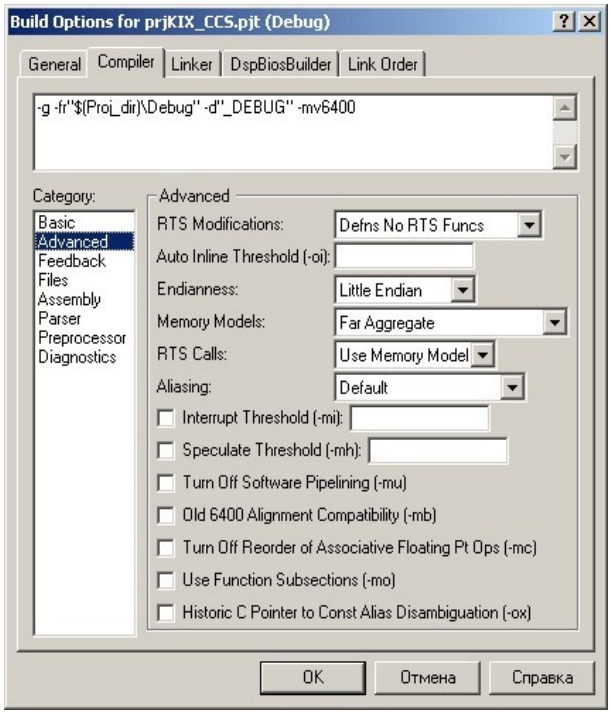

Рис.19. Настройки категории «Advanced» на закладке «Compiler».

Затем перейти на закладку «*Linker*» и привести настройки в соответствие с тем, что показано на рис.20.

Главное, в разделе «*Category*» для пункте «*Basic*» необходимо выставить значение «0x1000» в окнах «*Heap Size (-heap)*» и «*Stack Size (-stack)*».

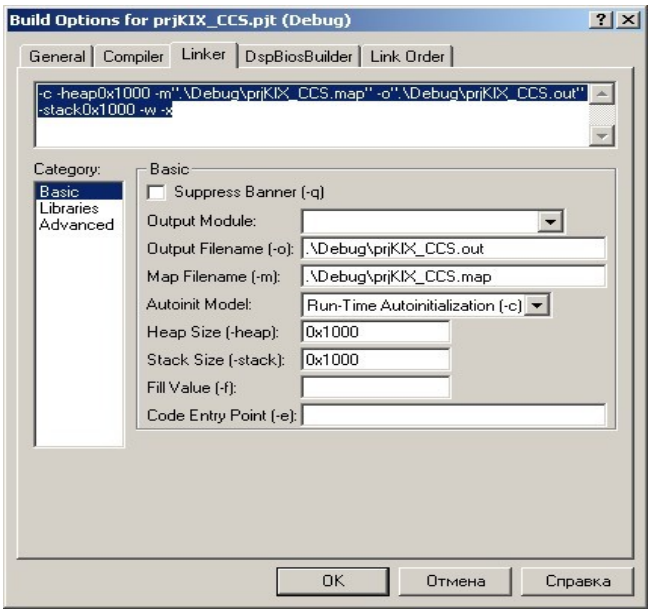

Рис.20. Настройки категории «Basic» на закладке «Linker».

Дальнейшее уточнение настроек проекта связано с настройками памяти процессора. Для этого необходимо в главном меню CCS выбрать пункт «*Option->Memory Map*» (рис.21) и появившемся окне «*Memory Map*» привести настройки в соответствие с рис.22. Обратите внимание на наличие записи «0x00000000 - 0xFFFFFFFF : RAM» в области поля «*Memory Map List*». Если запись не содержит слова «RAM» необходимо попробовать выставить или снять галочку «*Enable Memory Mapping*».

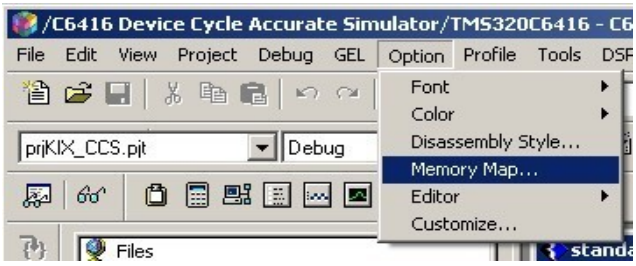

Рис.21. Вызов окна настройки памяти.

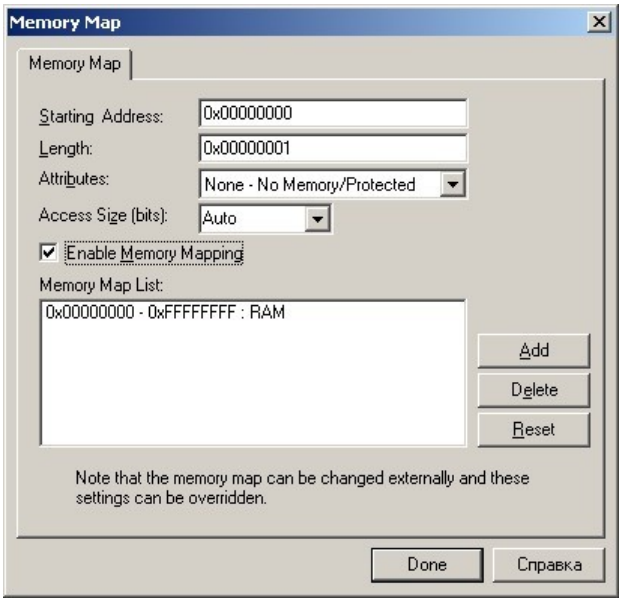

Рис.22. Уточнение настроек памяти процессора.

И в завершении этапа уточнения настроек проекта вызываем окно дополнительных настроек «*Customize*», как это показано на рис.23.

Рис.23. Вызов окна дополнительных настроек.

В появившемся окне «*Customize*» необходимо установить галочку в пункте «*Load Program After Build*» на закладке «*Load Program*» на закладке «*Program/Project/CIO*», как это показано на рис.24.

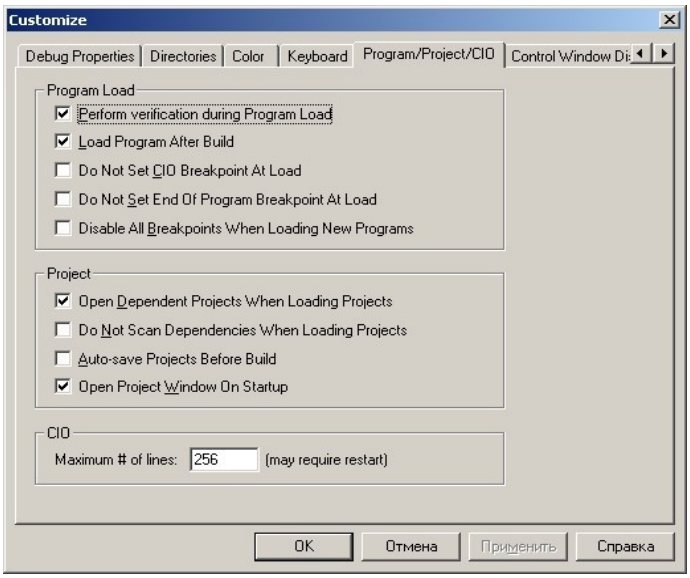

Рис.24. Настройка автоматической загрузки программного кода после его компиляции.

Процес настройки проекта завершен. Можно приступать к компиляции проекта.

Запуск процесса компиляции осуществляется либо через главное меню CCS путем выбора пункта «*Project->Rebuild All*», либо нажатием кнопки полной компиляции проекта  на панели быстрых кнопок (см.рис.25). Процесс компиляции отображается на закладке «*Build*» (см.рис.25).

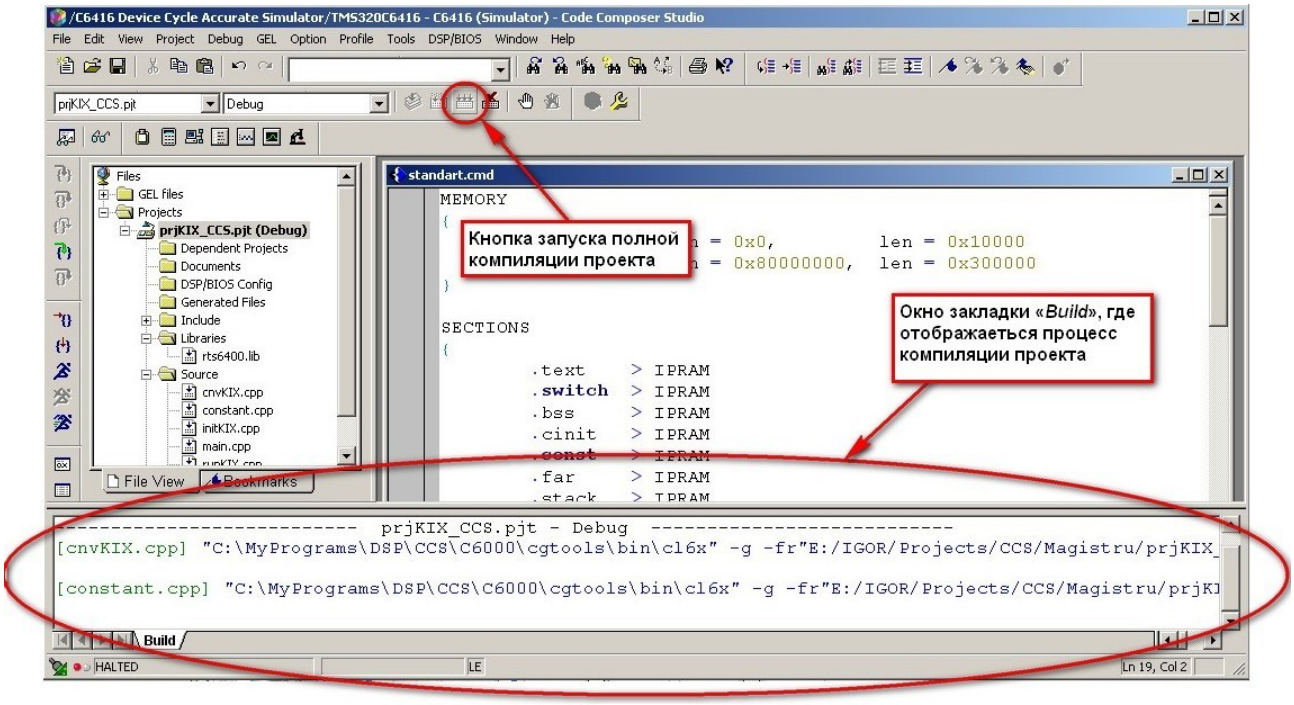

Рис.25. Запуск процесса компиляции проекта.

После успешного завершения компиляции программный код загружается в сигнальный процессор, что отображается появлением окна с индикатором загрузки (см.рис.26).

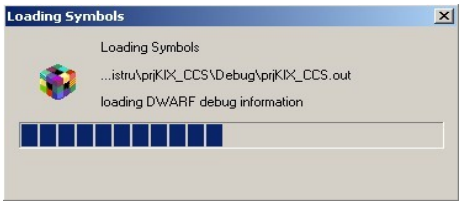

Рис.26. Отображение процесса загрузки программного кода.

Вид CCS после успешной компиляции и загрузки кода в процессор показана на рис.27: открылось новое окно в области редактирования файлов. Это окно дисъассемблера, где показан код программы и адреса по которым он расположен в памяти процессора.
Обратите внимание, что в окне закладки «*Build*» отображается результат процесса компиляции. В нашем случае можно наблюдать абсолютный успех процесса компиляции.

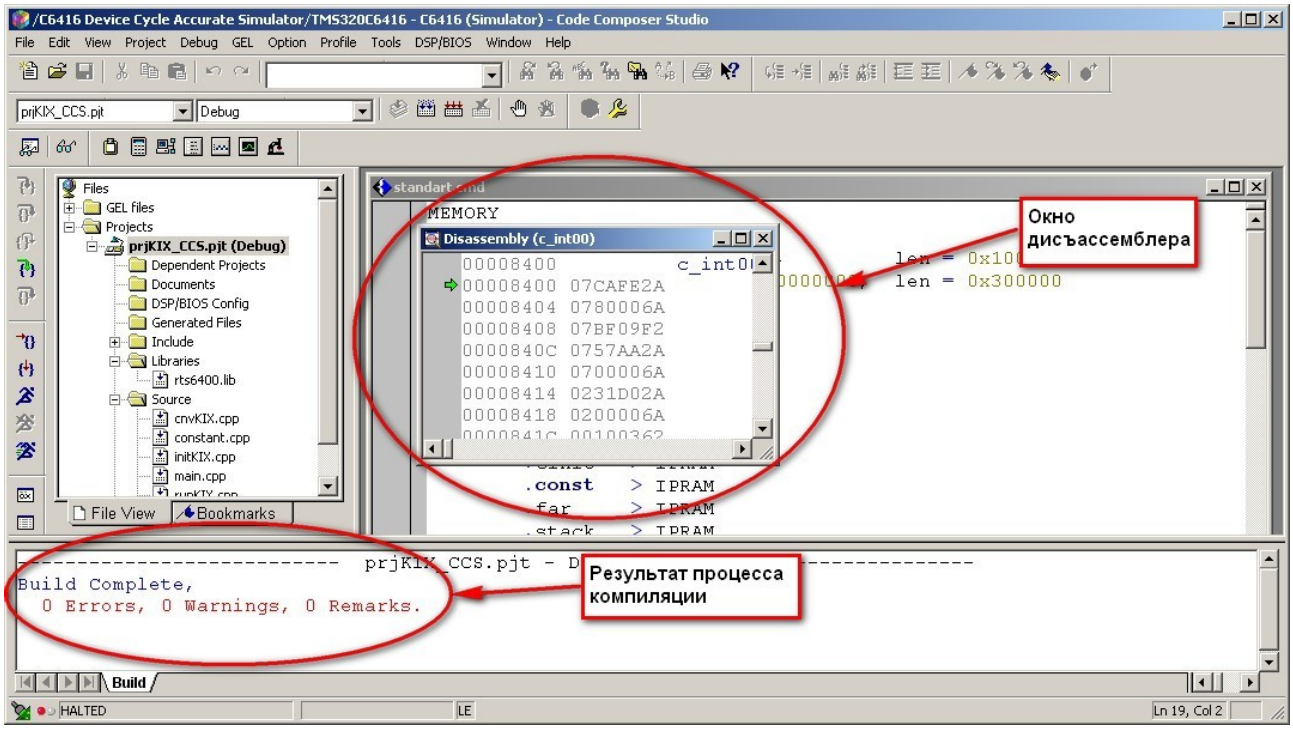

Рис.27. Вид CCS после успешной компиляции и загрузки программного кода в цифровой сигнальный процессор.

Открытых файлов в области редактирования становиться все больше. Отладочная среда ССЫ позволяет более удобно настроить их отображение. Вначале разверните окно дисъассемблера во все область редактирования нажав соответствующую кнопку, как это показано на рис.28.

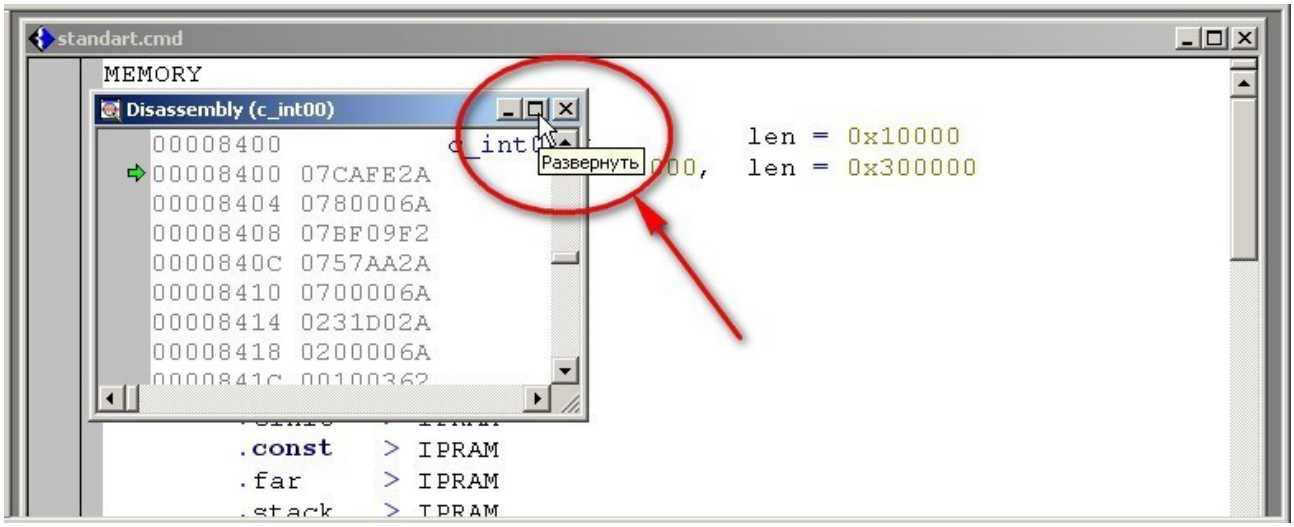

Рис.28. Развертывание окна дисъассемблера во всё область редактирования.

После этого CCS примет вид, показанный на рис.29. В окне дисъассемблера отображается точка входа в программу (*c_int00*:), которая соответствует началу функции *main()*. Кроме этого, можно видеть адрес команды, числовой код команды, соответствующую коду мнемонику команды и значение операндов команды.
Окно дисъассемблера занимает все пространства окна редактирования. Что бы освободить место для отображение других файлов необходимо щелкнуть правой кнопкой мыши в области окна дисъассемблера и в появившемся контекстном меню снять галочку с пункта «*Float In Main Windows*». Этот процесс иллюстрирует рис.29.

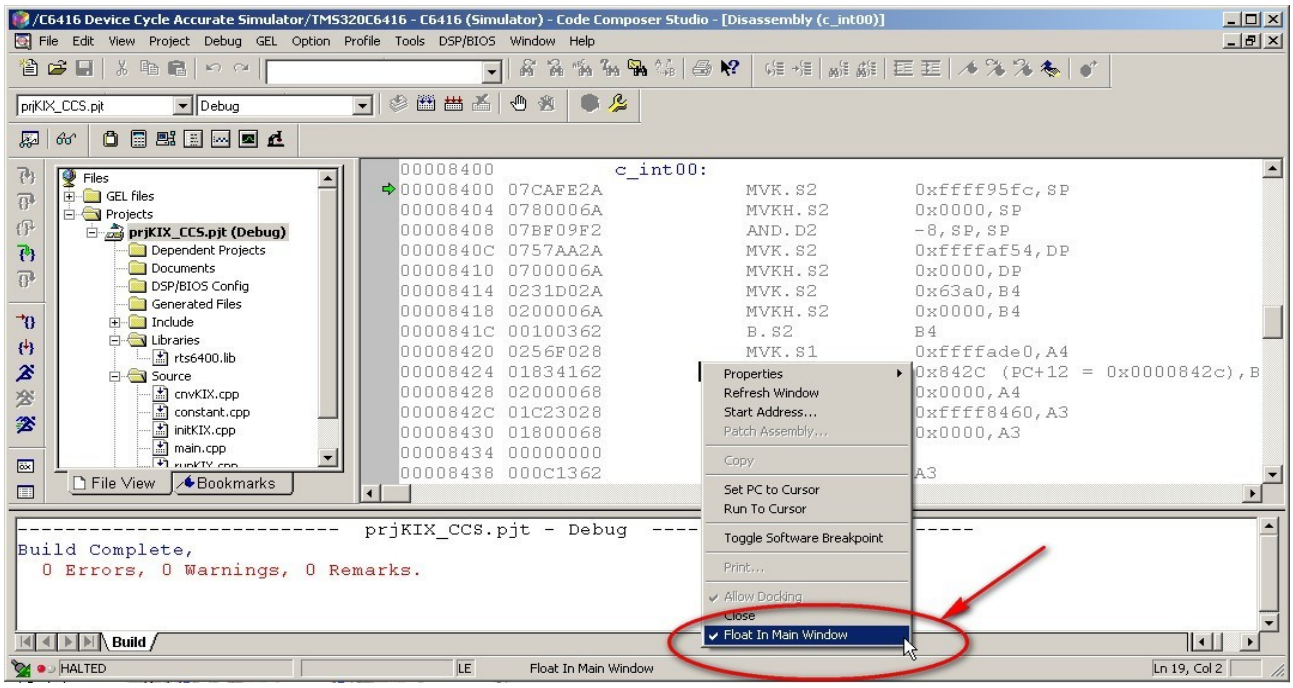

Рис.29. Настройка отображения окна дисъассемблера.

После всех манипуляций CCS примет вид, показанный на рис.30. Теперь можно запустить выполнение программного кода. Для этого необходимо выбрать в главном меню CCS пункт «*Debug->Run*», или нажать кнопку запуска проекта в веде синего бегущего человечка  (см.рис.30). Кстати, то же самое можно выполнить нажав кнопку «*F5*».

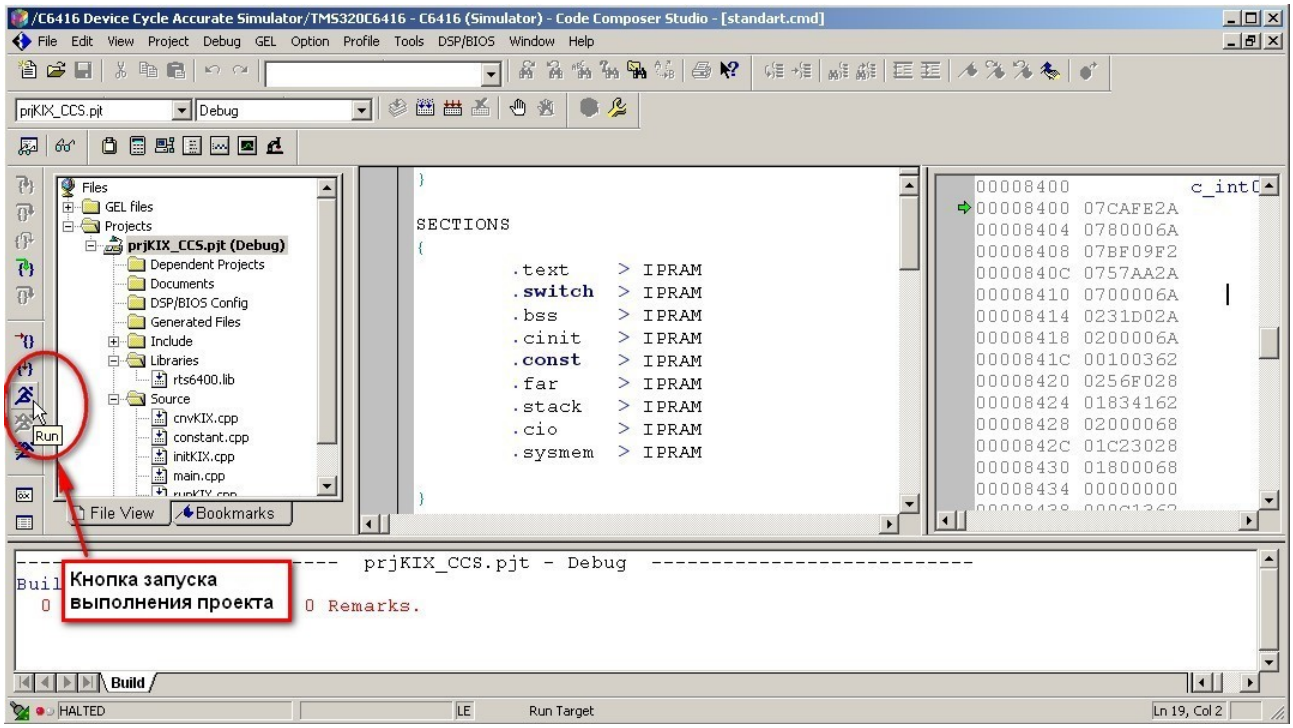

Рис.30. Вид CCS после настройки отображения окна дисъассемблера.

К сожалению, первый запуск для рассматриваемого примера прошел неудачно. Внизу окна программы CCS появилась новая закладка «*Stdout*», где отображается процесс выполнения программного кода, а в данном случае выводиться сообщение об ошибке открытия входного файла, что обусловлено отсутствием такого файла в папке проекта (см. рис.31).

Кроме этого, при запуске программного кода на выполнения появилось окно запроса ввода символа — «*Standard Input Dialog Box*» (см.рис.31). Это окно функции *getchar()*, которая вызывается в функции *main()*.

Завершить выполнение кода программы можно нажав кнопку «*Enter*» на клавиатуре, либо щелкнув кнопку «*OK*» в окне «*Standard Input Dialog Box*».

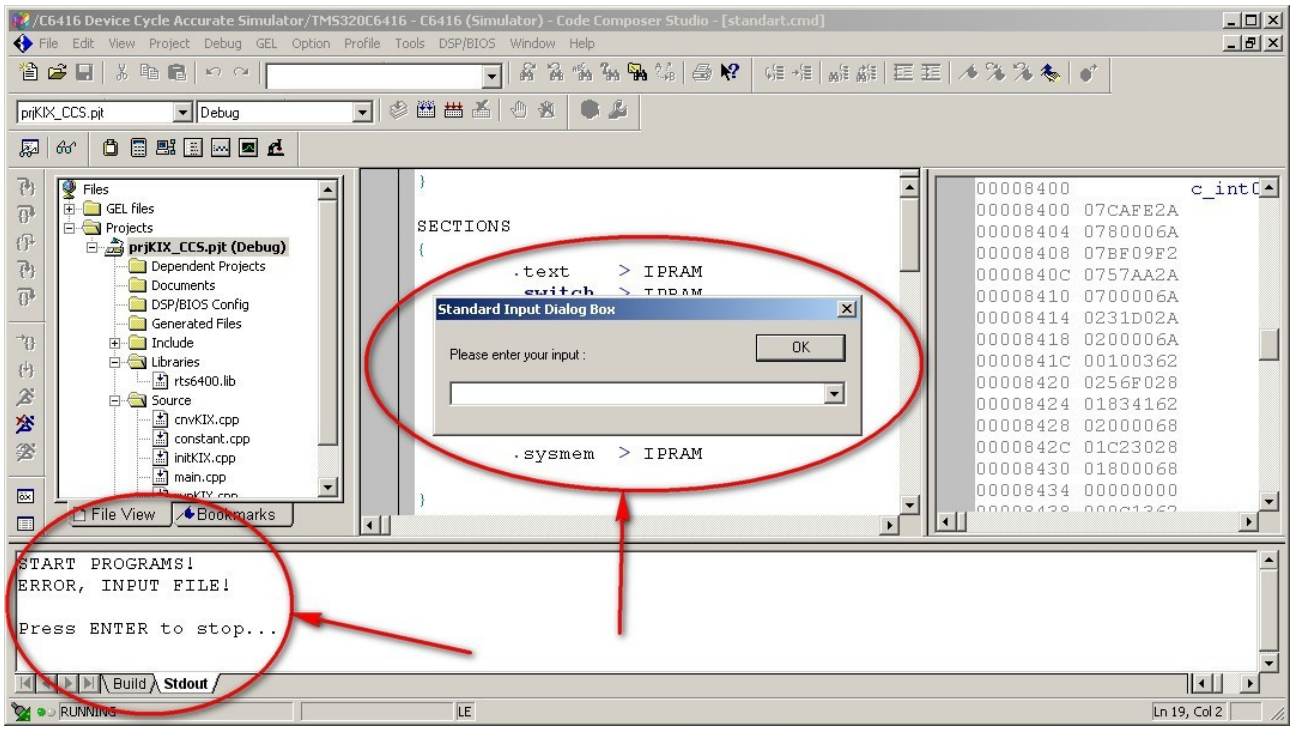

Рис.31. Ошибка выполнения программного кода, связанная с отсутствием входного файла.

Для исправления возникшей ошибки необходимо скопировать файл «*inp.dat*» из предыдущей лабораторной работы в папку «*Debug*», которая находиться внутри папки разрабатываемого проекта и создается автоматически при первом запуске компиляции проекта. После этого необходимо перекомпилировать проект заново, что позволит загрузить программный код в процессор, и запустить его на выполнения. Отметим, что повторная компиляция может выполняться нажатием на кнопку  . Это позволит провести компиляцию только тех файлов, что были изменены с прошлой компиляции. В реальной практики это значительно сокращает время компиляции. В данном случае, когда файлы не изменялись вообще, произойдет только повторная загрузка кода программы в процессор.

На этот раз все прошло успешно (см. рис.32). Что бы убедиться в корректной работе программы необходимо сравнить файл «*out.dat*», созданный при выполнении кода в CCS, и аналогичный файл из предыдущей лабораторной работы. Они должны полностью совпадать.

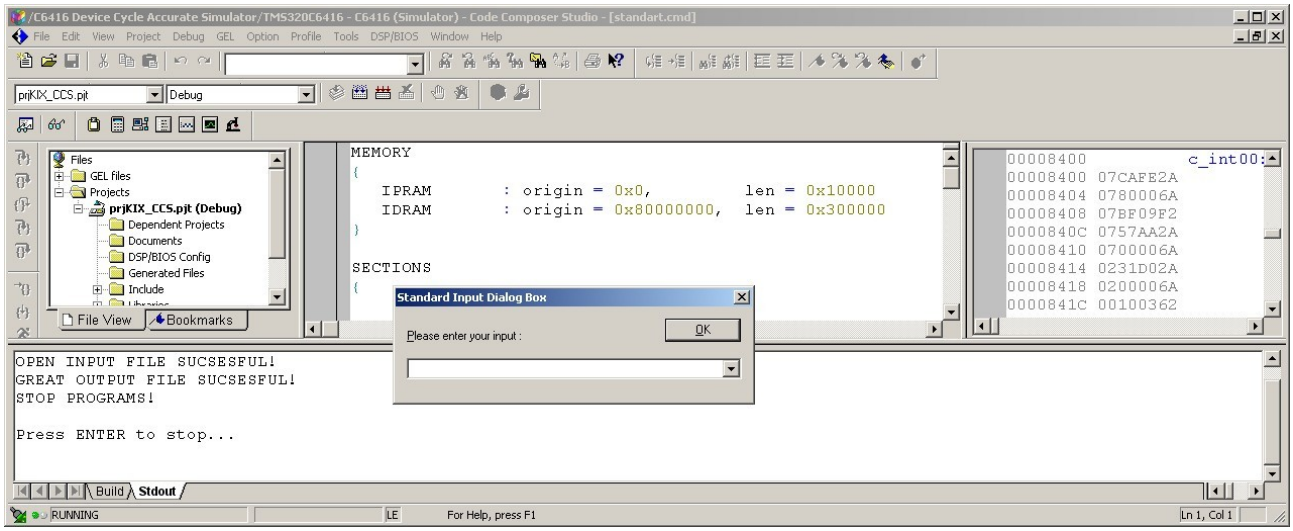

Рис.32. Успешное выполнения программного кода.

Заметим, что перед каждым запуском программы необходимо ее перегрузить или установить на начальный адрес. Это можно сделать различными способами. В нашем случае мы воспользовались тем обстоятельством, что после компиляции происходит автоматическая загрузка программы в процессор.

3. Модификация файла *main()*

Функция *main()* для CCS может быть значительно упрощена. В этом разделе будет показано как.

Первое, что нужно сделать, это подготовить входной файл, так как формат данных измениться. Более подробно этот момент рассмотрен в рекомендованной литературе (статья «Тестирование программного кода для ЦСП TMS320C6000», которая размещена на сайте http://www.scanti.ru/univ_stati.html).

Для преобразования входного файла в нужный формат можно воспользоваться утилитой *«Char_To_Text.exe»*. Которую необходимо скопировать в папку «*Debug*» тестируемого проекта. Запустить ее можно при помощи командного файла «*Char_To_Text.bat*». Этот файл так же должен быть скопирован в папку «*Debug*». В папке проекта появиться новый файл с именем «*inpCCS.dat*».

Второе — это подготовка выходного файла. Отметим, что в отличие от проекта для программы *DevC++*, выходной файл должен существовать. Создается он при помощи утилиты той же утилиты «*Char_To_Text.exe*», но которая запускается уже другим командным файлом — «*Char_To_Text_Out.bat*», который так же должен быть скопирован в папку «*Debug*». После ее запуска появиться файл «*outCCS.dat*».

Третье — модификация функции *main()*. Для этого необходимо открыть файл с функцией *main()* в окне редактирования щелкнув по имени файла в закладке «*File View*», как это показано на рис.33.

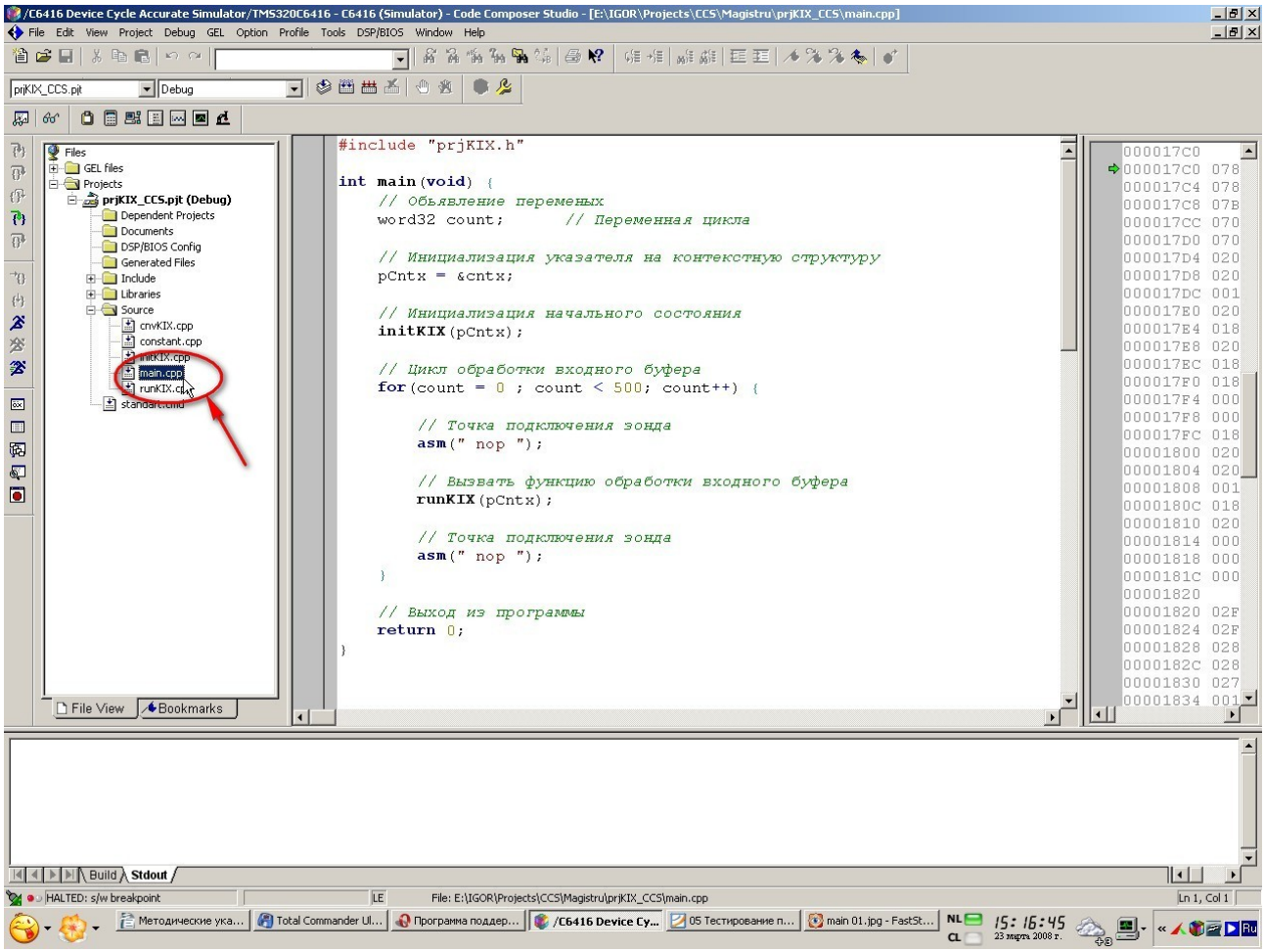

Рис.33. Открытие функции main() в окне редактирования.

В окне редактирования файла «*main.cpp*», где набирается текст, приведенный на рис.33. Кроме этого, листинг функции приведен в приложении 2. Отметим, что старый текст должен быть стерт, то есть приведенный на рис.33 текст набирается вместо старого.

И последнее — это подключение созданных файлов к проекту. Это производиться через использование точек тестирования программного кода (*Breakpoint*).

Вначале устанавливают точки тестирования в коде программы. Для чего на строке, где необходимо установить точку, помещают курсор (см.рис.34), а затем нажимают кнопку на панели быстрых кнопок (см.рис.34). После этого напротив строки, где установлена *Breakpoint*, появиться красная точка (см.рис.34). Повторив описанную процедуру, но нажав кнопку , можно деактивировать точку тестирования. Затем необходимо открыть окно настройки установленных точек тестирования. Это можно выполнить через главное меню CCS, выбрав пункт «*Debug->Breakpoints...*», после чего в CCS появиться дополнительное окно точек тестирования (см.рис.34).

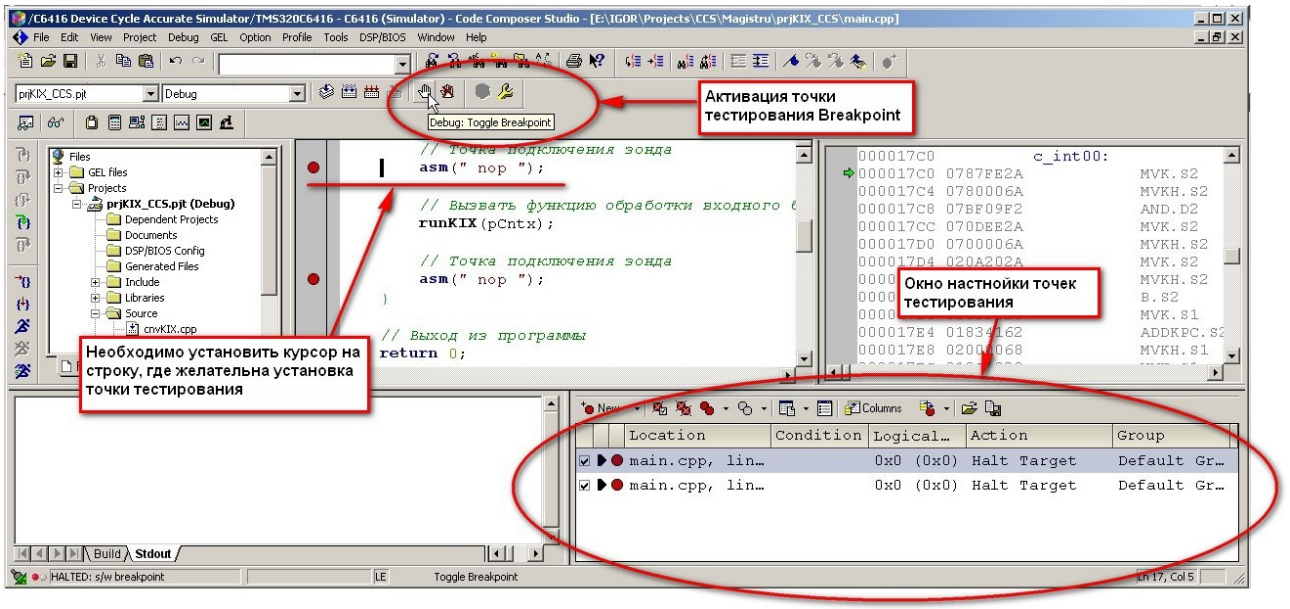

Рис.34. Установка точек тестирования (Breakpoint) программного кода.

Теперь приступаем к подключению входного и выходного файлов к проекту.

Для подключения входного файла выделяем мышкой первую точку тестирования щелкаем правой кнопкой (см.п.1 на рис.35). В появившемся контекстном меню (см.п.2 на рис.35) выбираем пункт «*Property Windows*». Появиться окно настройки параметров первой точки тестирования (см.п.3 на рис.35).

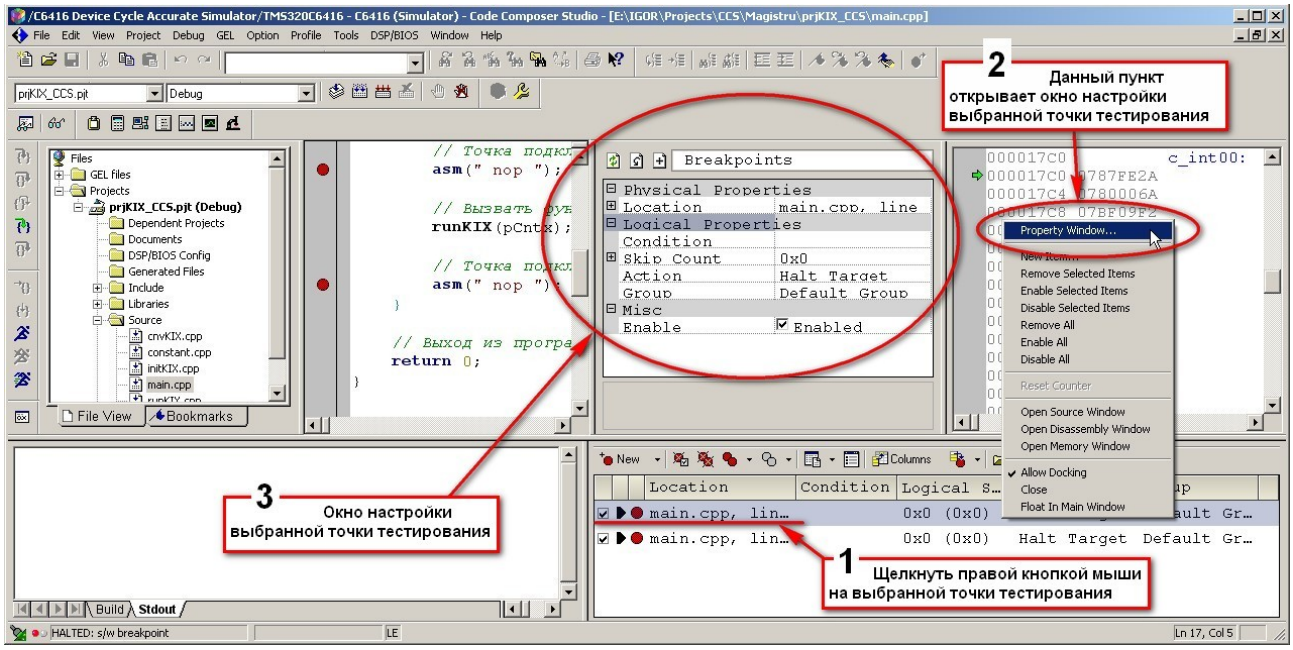

Рис.35. Открытие окна настройки свойств точки тестирования.

В поле «*Action*» необходимо в выпадающем меню выбрать пункт «*Read Data from File*», как это показано на рис.36. После этого в поле «*Action*» появятся дополнительные пункты (см.рис.37). Необходимо в пункте «*File*» нажать кнопку с тремя точками (см.рис.38).

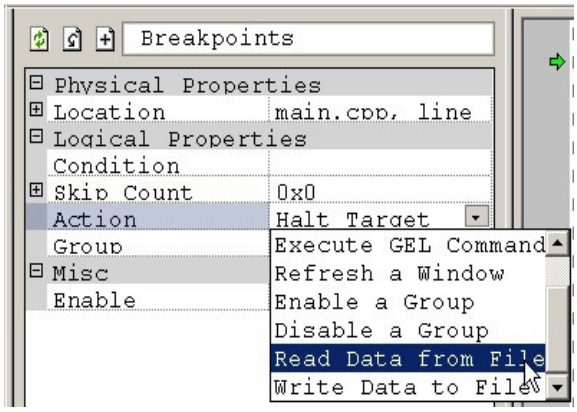

Рис.36. Включение режима подключения входного файла для чтения.

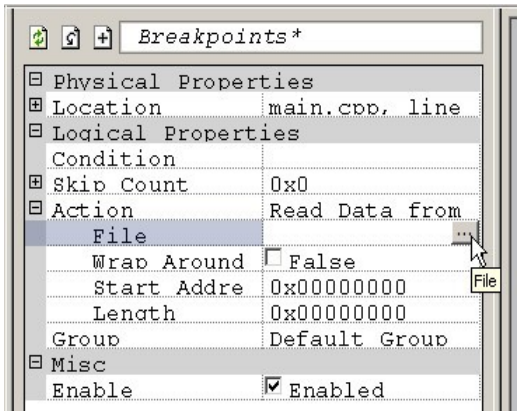

Рис.37. Определение подключаемого файла.

Появиться окно выбора подключаемого файла (см.рис.38). Необходимо зайти в папку «*Debug*» проекта и выбрать файл «*inpCCS*» и нажать кнопку «*Открыть*».

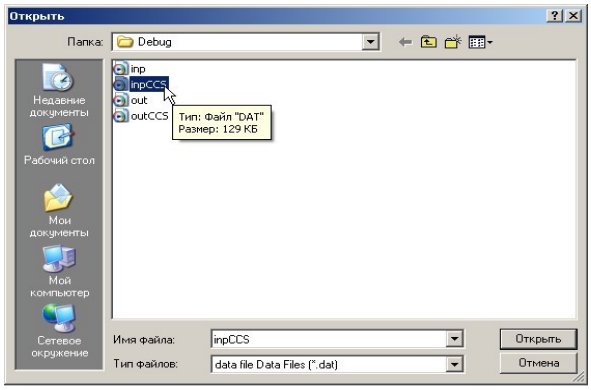

Рис.38. Выбор подключаемого файла.

Затем поставте галочку напротив пункта «*Wrap Around*» (см.рис.39). Это позволит организовать циклическое считывание данных из файла, то есть если в процессе считывания данные в файле закончатся, считывание данных из файла начнется сначала.

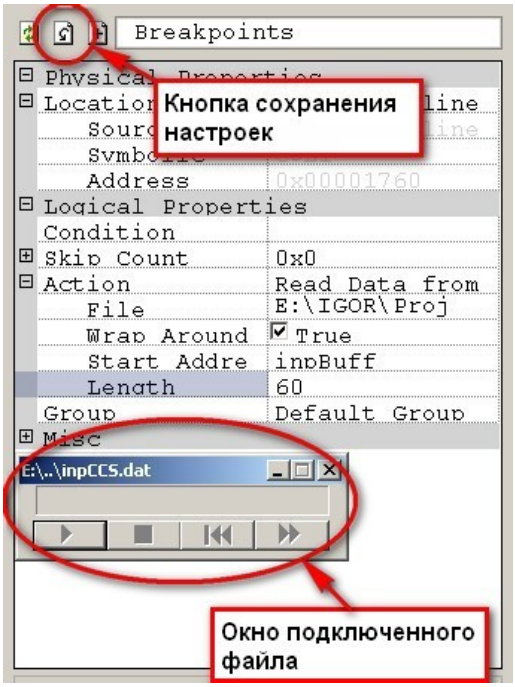

Рис.39. Настройка параметров подключения входного файла.

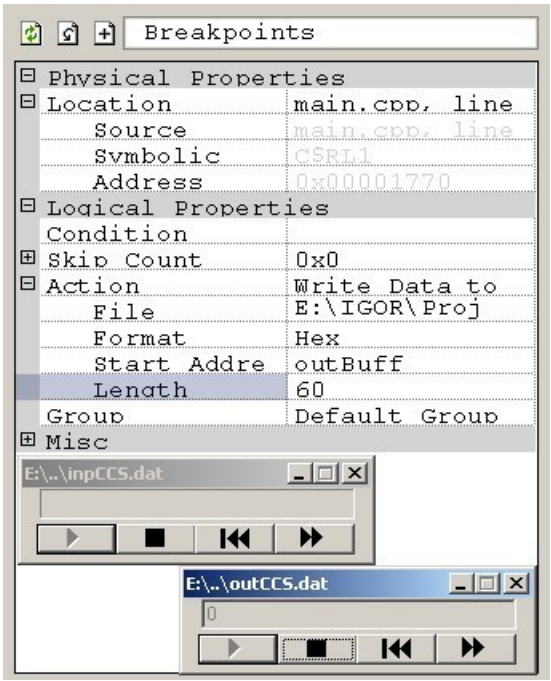

Рис.40. Настройка параметров подключения выходного файла.

Следующий шаг — это указать имя буфера, куда будут считываться данные из файла (поле «*Start Address*» на рис.39), а так же его длину (поле «*Lenath*» на рис.39). Имя буфера в рассматриваемом примере — это «*inpBuff*», так как именно его мы создаем в файле «*constant.cpp*» для хранения входных отсчетов. Длинна указанна равной значению 60. Это определяется тем, что в рассматриваемом примере размер буфера определяется макросом «SAZEBUFF» и равен 120. Но в рассматриваемом примере используются 16-и разрядные входные отсчеты, а при применении данного способа подключение файлов считывание организуется 32-х разрядными числами, то есть за один такт будет загружено два отсчета, поэтому размер буфера указан в два раза меньше, чем был определен в программе. Заметим, что вследствие этого для данного проекта макрос «SAZEBUFF» должен определять четный размер входного/выходного буферов.

В заключении процесса подключения файла необходимо подтвердить внесенные изменения, для чего необходимо нажать кнопку  (см.рис.39). Появиться окно подключенного файла (см.рис.39). Данное окно может появиться в любом месте монитора. Его можно перемещать мышкой и разместить там, где это удобно.

Аналогичную процедуру проводим для подключения выходного файла. Все настройки для подключения выходного файла показаны на рис.40. Обратите внимание, что имя выходного файла — «*outCCS*», имя выходного буфера — «*outBuff*». После подтверждения настроек (нажатие кнопки ), появиться еще одно окно подключенного файла.

Вот теперь все настройки завершены. Можно заново откомпилировать проект (нажав кнопку ) и запустить проект на выполнения (нажав кнопку ). Обратите внимание, что в случае наличие точек тестирования в коде, запускать программу на выполнения необходимо при помощи кнопки анимированного запуска —  (см.рис.41).

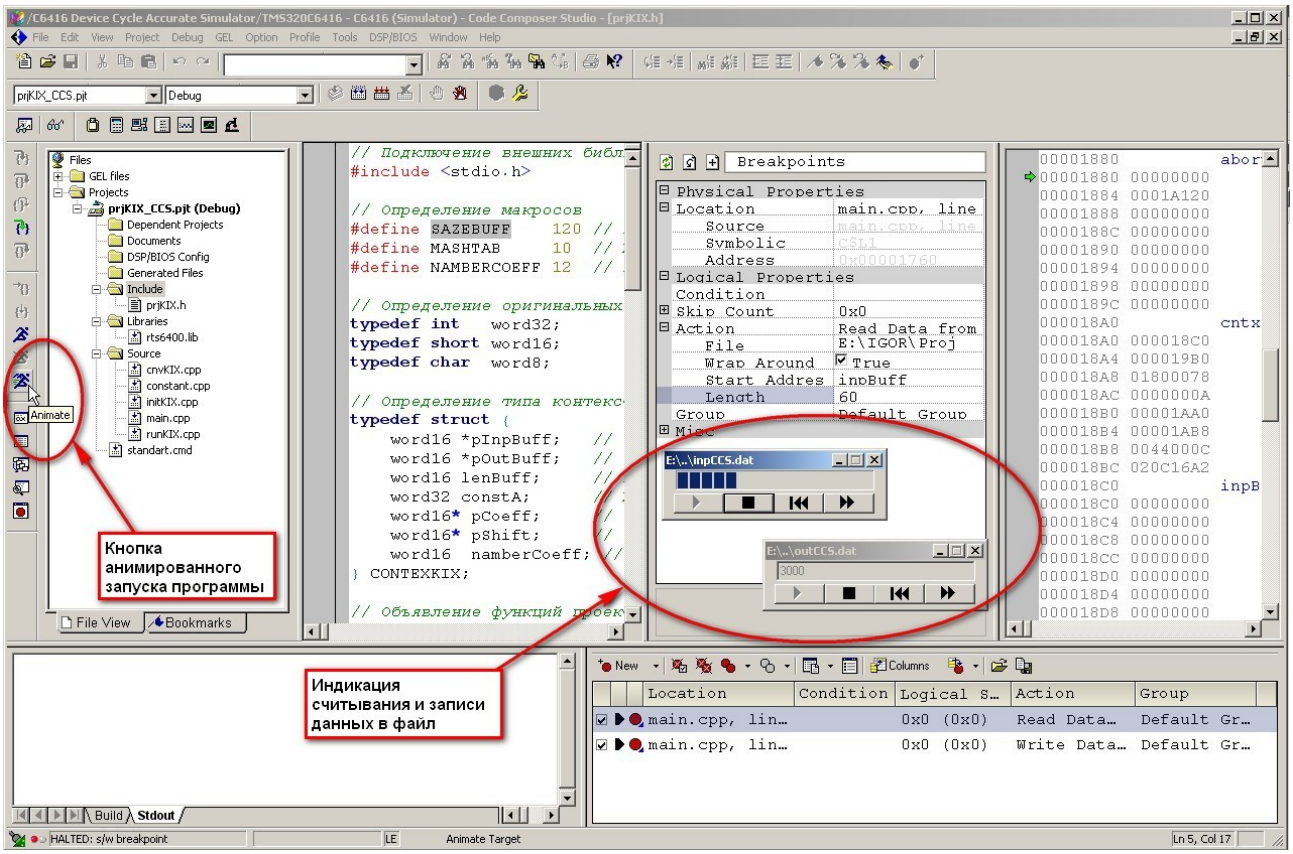

Рис.41. Запуск программы на выполнения с подключенными входным и выходным файлами.

После компиляции проекта положение окон подключенных файлов может измениться. Часто они просто накладываются друг на друга. Необходимо перетащить их мышкой в удобное для наблюдение место экрана. После запуска программы в данных окнах будет отображаться процесс чтения и записи данных в/из подключенных файлов.

После завершения выполнения программного кода в папке «*Debug*» появиться файл «*outCCS.dat*». Для просмотра этого файла в программе EDSW и сравнения его с выходным

файлом из предыдущей лабораторной работы необходимо провести его преобразование к нужному формату представления данных. Для этого надо воспользоваться утилитой «*Text_To_Char.exe*», которая запускается командным файлом «*Text_To_Char.bat*». И утилита и командный файл должны быть скопированы в папку «*Debug*».

После запуска командного файла появится новый файл «*outForCmp.dat*». Его необходимо сравнить с файлом «*out.dat*» из предыдущей лабораторной работы. Файлы должны полностью совпадать. Отличатся они могут только длинной (см.рис.42).

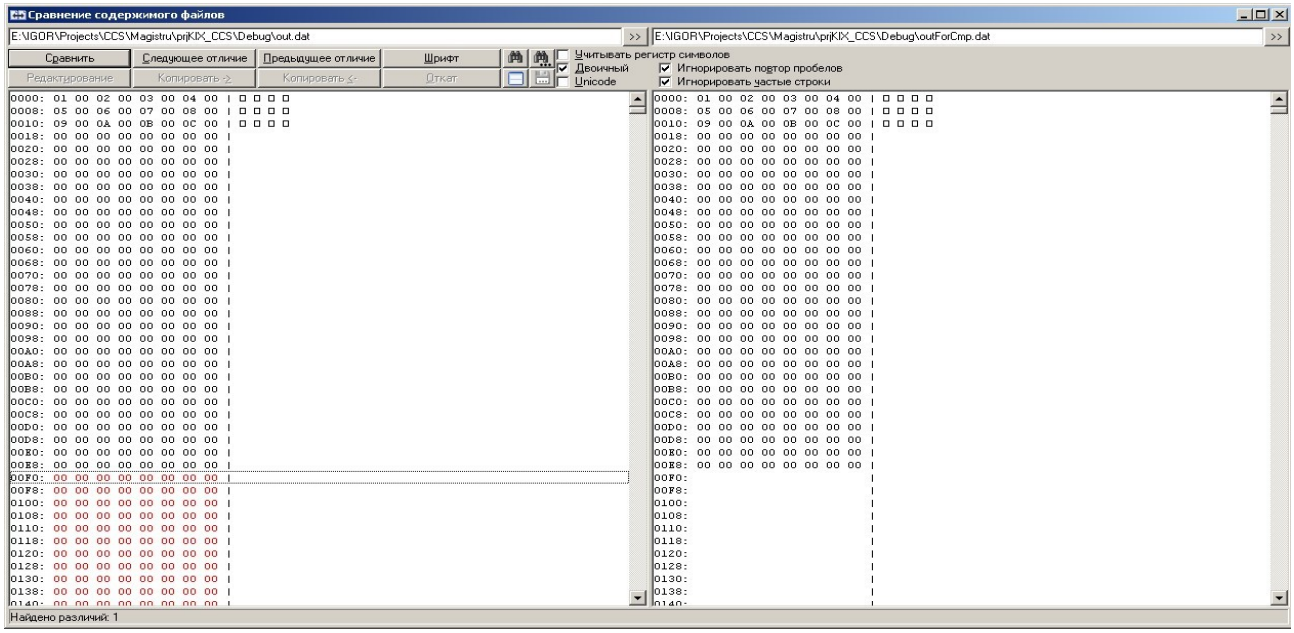

Рис.42. Сравнение полученного результата.

Что бы в дальнейшем при запуске проекта не производить настройки отладочной среды заново, можно сохранить их. Для этого в главном меню CCS необходимо выбрать пункт «*File->Workspace->Save Workspace As...*» (см. рис.43. п.1). Появиться окно «*Save Workspace*», где нужно указать имя файла для хранения настроек, например, «*ws*_01» (см. рис.43. п.2) и нажать кнопку «*Сохранить*» (см. рис.43. п.3). После этого можно закрыть программу CCS, а при повторном запуске открывать уже не проект, выбрав пункт «*Project->Open...*», а рабочее пространство, выбирать пункт «*File->Workspace->Load Workspace*» (см. рис.43. п.4). Это приведет не только к загрузке проекта, но и к восстановлению всех настроек CCS, которые были сохранены в файле «*ws*_01».

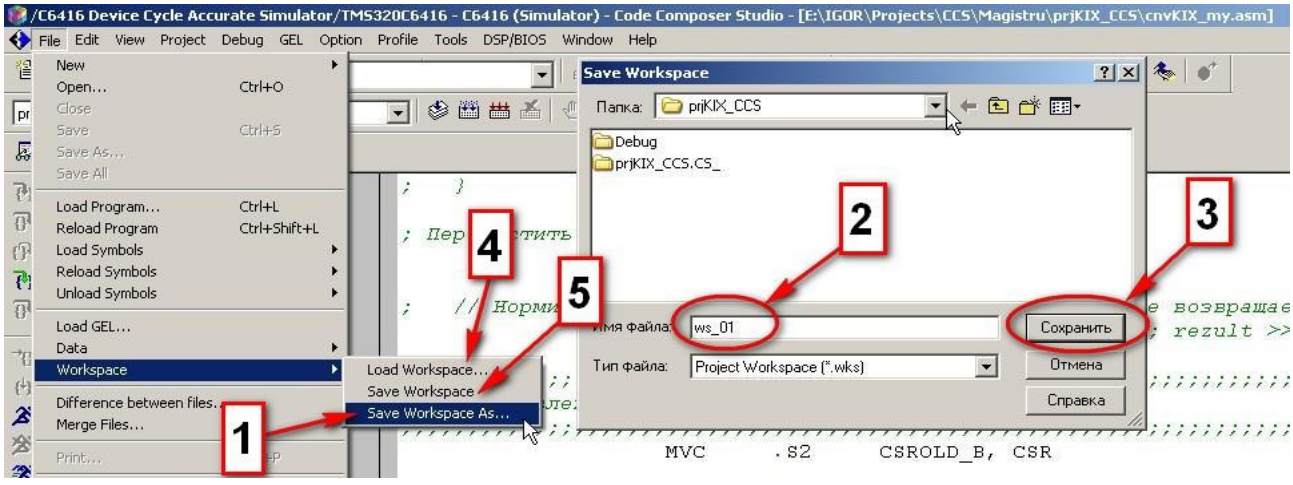

Рис.43. Сохранение рабочего пространства.

Отметим, что в дальнейшем сохранять настройки рабочего пространства желательно перед каждым выключением CCS. При этом можно воспользоваться пунктом «*File->Workspace->Save Workspace*» (см. рис.43. п.5). Это приведет к сохранению настроек в уже существующем файле (для рассматриваемого примера - это файл «*ws*_01»). При этом окно «*Save Workspace*» не появиться.

4. Разработка ассемблерного кода функции *cnvKIX()*

Вначале необходимо создать файл с именем «*cnvKIX_my.asm*» и подключить его к проекту. Процедура аналогична той, что была проведена при создании и подключении файла «*standard.cmd*» (см. в разделе 2). Окна сохранения и подключения файла «*cnvKIX_my.asm*» показаны на рис.44 и 45, соответственно.

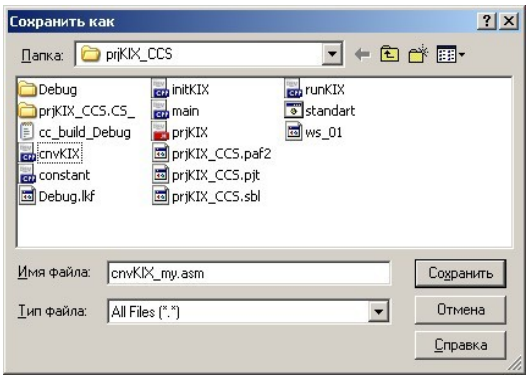

Рис.44. Окно сохранения файла.

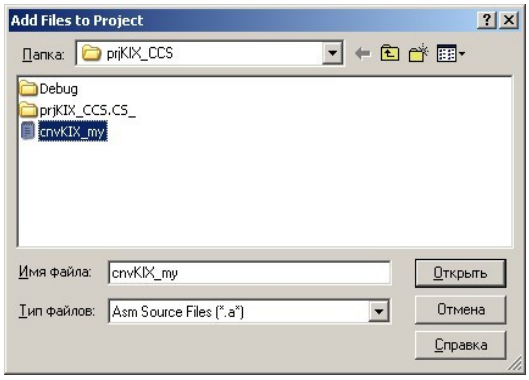

Рис.45. Окно выбора подключаемого к проекту файла.

Следующий шаг — скопировать содержимое файла «*cnvKIX.cpp*» в открывшееся окно редактирования файла «*cnvKIX_my.asm*» и закомментировать каждую строчку знаком «;»:

```
;// Функция вычисление свертки
;#include "prjKIX.h"

;word16 cnvKIX(word16* pCoeff, word16* pShift, word16 namberCoeff, word32 constA){
;       // Объявление локальных переменных
;       word16 coeffX;
;       word16 coeffB;
;       word32 count;
;       word32 coeff;
;       word32 rezult;

;       // Инициализация локальных переменных
;       rezult = 0;

;       // Цикл обработки линии задержки
;       for(count = 0; count < namberCoeff; count++){

;               // Чтение текущего отсчета
;               coeffX = *pShift++;

;               // Чтение соответствующего коэффициента фильтра
;               coeffB = *pCoeff++;

;               // Умножение отсчета на коэффициент
;               coeff = coeffX * coeffB;

;               // Накопление результата
;               rezult += coeff;
;       }

;       // Корректировка значения указателя
;   pShift--;

;       // Сдвиг линии задержки
;       for(count = namberCoeff - 1; count > 0; count--){

;               // Чтение предыдущего отсчета
;               coeffX = pShift[-1];

;               // Запись предыдущего отсчета в текущую ячейку
;               *pShift-- = coeffX;
;       }

;       // Нормирование результата суммирования
;       rezult >>= constA;

;       // Выход из функции
;       return rezult;
;}
```

Сохраните изменения в файле «*cnvKIX_my.asm*», нажав кнопку (см.рис.46).
Вид CCS после создания, подключения, модификации и сохранения файла «*cnvKIX_my.asm*» показан на рис.46.

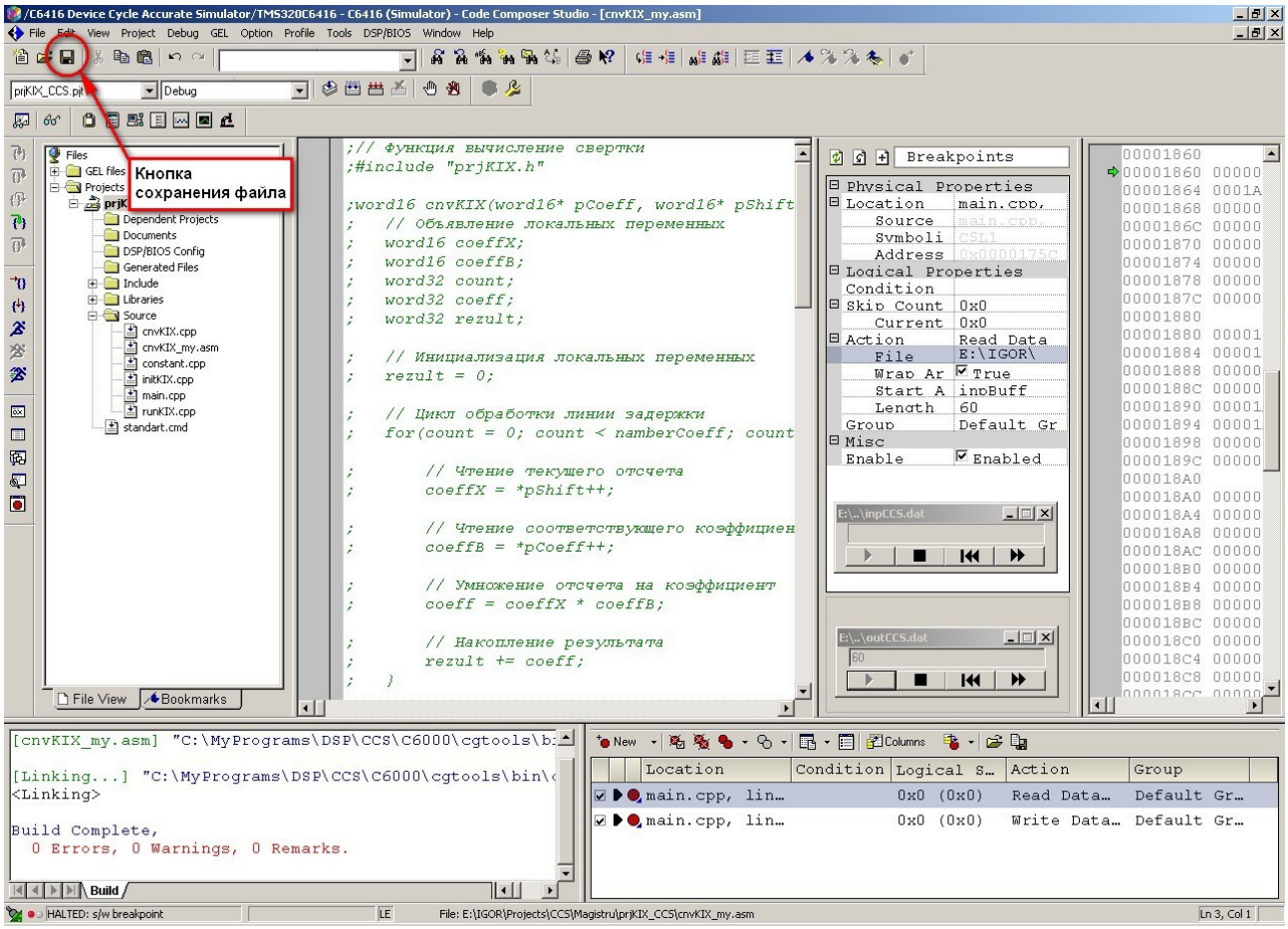

Рис.46. Вид CCS после создания, подключения, модификации и сохранения файла «cnvKIX_my.asm».

Так как окно настроек свойств точек тестирования пока не нужно, можно его закрыть. Для этого нужно щелкнуть правой кнопкой мыши в области этого окна и в появившемся контекстном меню выбрать пункт «*Hide*» (см.рис.47). После этого окно CCS примет вид, показанный на рис.48.

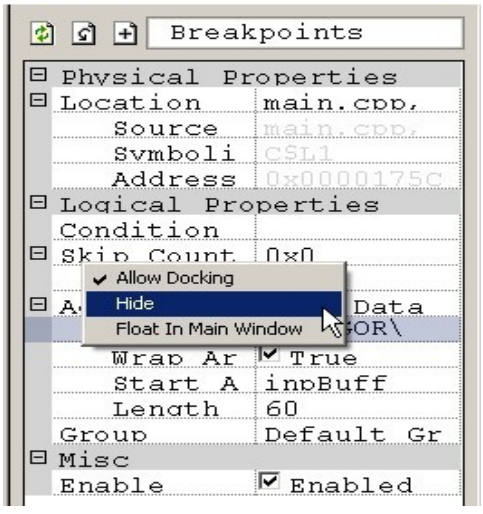

Рис.47. Выключение окна настроек точек зондирования.

Для написания ассемблерного кода функции «*cnvKIX()*» необходимо знать ее ассемблерное имя. Подсмотреть это имя можно в промежуточном ассемблерном файле, который создает сама отладочная среда CCS. Однако, этот файл уничтожается после завершения процесса компиляции. Что бы этого не происходило, необходимо щелкнуть правой кнопкой мыши на имени файла «*cnvKIX.cpp*» в закладке «*File View*» и появившемся контекстном меню

вызываем окно настройки параметров компиляции щелкнув по пункту «*File Specific Options*» (см.рис.48). Появиться окно настройки параметров компиляции файла (рис.49).

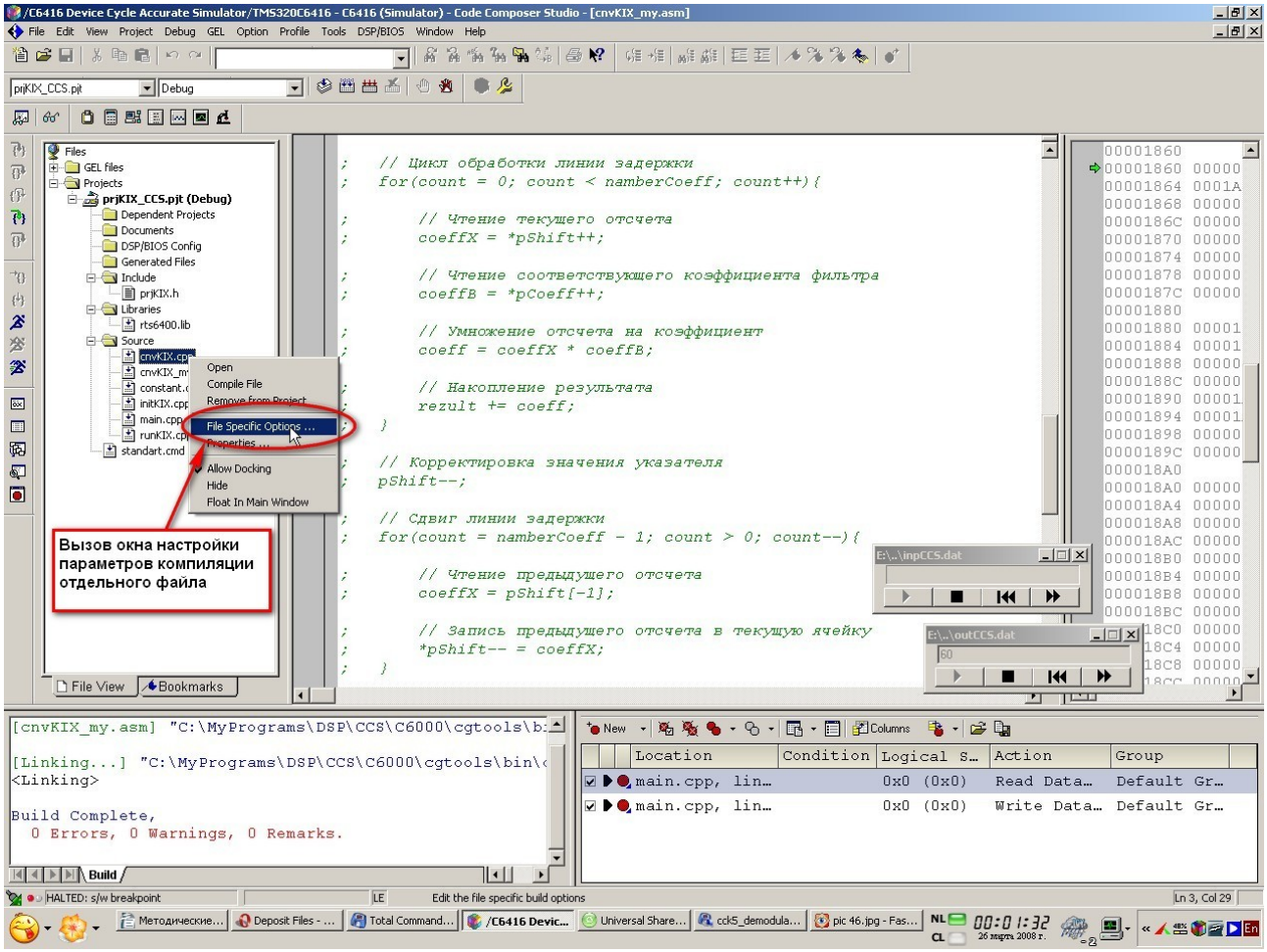

Рис.48. Вид CCS после закрытия окна настроек точек зондирования.

В этом окне переходим на закладку «*Compiler*» (п.1 на рис.49), выбираем категорию «*Assembly*» (п.2 на рис.49), ставим галочку «*Keep Generated .asm Files (-k)*» (п.3 на рис.49) и подтверждаем изменение параметров компиляции выбранного файла, нажав кнопку «*OK*» (п.4 на рис.49). После этого компилируем выбранный файл независимо от всего проекта нажав кнопку  (рис.50).

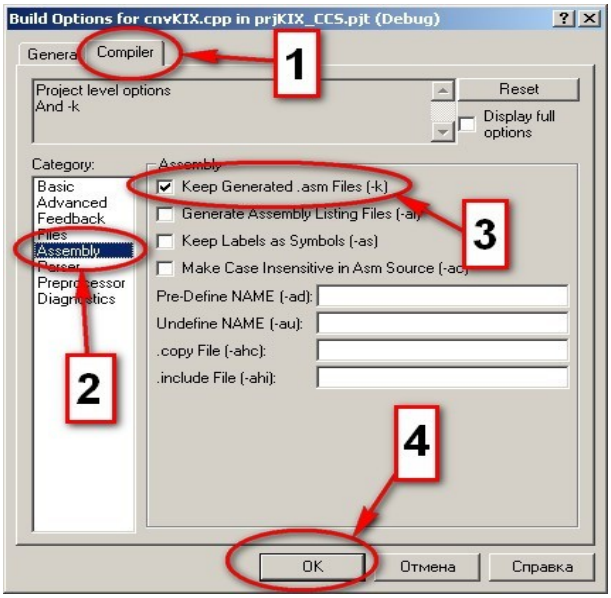

Рис.49. Окно настройки параметров компиляции отдельного файла.

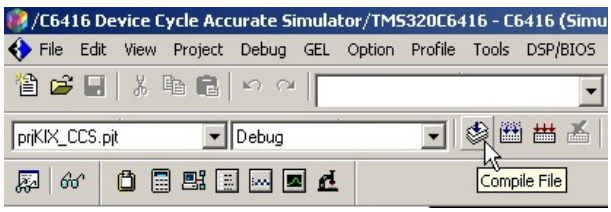

Рис.50. Компиляция отдельного файла.

В папке проекта появиться еще один файл с именем «*cnvKIX.asm*». Открываем его в любом

текстовом редакторе. Находим строчки вида:

```
      .sect     ".text"
      .global   _cnvKIX__FPsT1si
_cnvKIX__FPsT1si:
```

Первая строчка определяет секцию, где будет располагаться код функции *cnvKIX()*. Вторая — ассемблерное имя функции, а последняя — точку входа в функцию.

Копируем эти строчки и вставляем в файл «*cnvKIX_my.asm*». Кроме этого, необходимо назначить имена регистрам общего назначения в соответствии с параметрами функции и локальными переменными. Это можно сделать при помощи директивы «*.asg*».

Часть ассемблерного кода функции *cnvKIX()* с внесенными изменениями имеет вид:

```
;// Функция вычисление свертки
;#include "prjKIX.h"

;word16 cnvKIX(word16* pCoeff, word16* pShift, word16 namberCoeff, word32 constA){
; Объявление локальных переменных
    .asg    A3,  coeff_A        ; word32 coeff;
    .asg    A4,  pCoeff_A       ; 1-ый прм. фнк. -  word16* pCoeff
    .asg    A5,  coeffB_A       ; word16 coeffB
    .asg    A6,  namberCoeff_A  ; 3-ый прм. фнк. -  word16 namberCoeff
    .asg    A7,  rezult_A       ; word32 rezult;

    .asg    B0,  count_B        ; word32 count
    .asg    B3,  adrReturn_B    ; Адрес возврата
    .asg    B4,  pShift_B       ; 2-ый прм. фнк. - word16* pShift
    .asg    B5,  coeffX_B       ; word16 coeffX;
    .asg    B6,  constA_B       ; 4-ый прм. фнк. - word32 constA
    .asg    B15, SP             ; Указательна вершину стека

    .sect   ".text"
    .global _cnvKIX__FPsT1si

_cnvKIX__FPsT1si:

;   // Инициализация локальных переменных
;   rezult = 0;
```

При выходе из функции необходимо записать результат вычислений в регистр A4 (согласно регистровым соглашениям) и перейти по адресу возврата, хранящемуся в регистре B3 (так же в соответствии с регистровыми соглашениями). Заключительные строки кода примут вид:

```
;   // Нормирование результата суммирования
;   rezult >>= constA;

;   // Выход из функции
;   return rezult;
                    ; Перемещение возвращаемого значения в регистр A4
                    MV      .S1     rezult_A, A4

                    B       .S2     adrReturn_B

; Ожидания завершения операции безусловного перехода (5-ть тактов)
                    NOP 5
;}
```

Прежде чем приступить к написанию ассемблерного кода непосредственно соответствующего С-коду, необходимо сделать еще одно подготовительное действие — запретить прерывания на время выполнения ассемблерного кода функции в начале:

```
    .asg    B3,  adrReturn_B    ; Адрес возврата
    .asg    B4,  pShift_B       ; 2-ый прм. фнк. - word16* pShift
    .asg    B5,  coeffX_B       ; word16 coeffX;
    .asg    B6,  constA_B       ; 4-ый прм. фнк. - word32 constA
    .asg    B7,  CSROLD_B       ; Регистр для хранения значения регистра CSR
    .asg    B15, SP             ; Указательна вершину стека

    .sect   ".text"
    .global _cnvKIX__FPsT1si

_cnvKIX__FPsT1si:

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Запрет прерываний
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                MVC     .S2     CSR, CSROLD_B
                AND     .S2     CSROLD_B, -2, B1
                MVC     .S2     B1, CSR
```

Обратите внимание, что определен еще один регистр (*B*7) для хранения значения управляющего регистра «*CSR*» — «*CSROLD_B*».
В конце ассемблерного кода функции нужно восстановить значение управляющего регистра «*CSR*»:

```
;   // Нормирование результата суммирования
;   rezult >>= constA;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Восстановление служебного регистра CSR
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                    MVC        .S2     CSROLD_B, CSR

;   // Выход из функции
;   return rezult;
                    ; Перемещение возвращаемого значения в регистр A4
                    MV      .S1     rezult_A, A4

                    B       .S2     adrReturn_B

; Ожидания завершения операции безусловного перехода (5-ть тактов)
                    NOP 5
;}
```

Запрет прерываний обеспечивает корректную работу функции при оптимизации кода с использованием программного конвейера (более подробно этот вопрос рассмотрен в рекомендованных статьях). Отметим, что в конце кода не разрешаются прерывания, а восстанавливается то значение управляющего регистра «*CSR*», которое было в момент вызова функции *cnvKIX()*.
Теперь можно приступить к написанию ассемблерного кода тела функции. В силу того, что C-код был написан с учетом его преобразования к ассемблерному виду, каждому C-оператору легко можно поставить в соответствие его ассемблерный аналог. Например, для части C-кода соответствующего инициализации локальных переменных и циклу обработки линии задержки его ассемблерный аналог будет иметь вид:

```
;   // Инициализация локальных переменных
                    ZERO    .D1     rezult_A                ; rezult = 0;

; Определение начального значения переменной цикла
                    SUB     .L2X    namberCoeff_A, 1, count_B

;   // Цикл обработки линии задержки
loop01: ;   for(count = 0; count < namberCoeff; count++){

;       // Чтение текущего отсчета
                    LDH     .D2T2   *pShift_B++, coeffX_B   ; pShift = *pShift++;

;       // Чтение соответствующего коэффициента фильтра
                    LDH     .D1T1   *pCoeff_A++, coeffB_A   ; coeffB = *pCoeff++;

; Ожидание завершения операции чтения данных из памяти
                    NOP 4

;       // Умножение отсчета на коэффициент
                    MPY     .M1X    coeffX_B, coeffB_A, coeff_A ; coeff = coeffX * coeffB;

; Ожидание завершения операции умножения
                    NOP

;       // Накопление результата
                    ADD     .S1     rezult_A, coeff_A, rezult_A ; rezult += coeff;

; Условный переход при циклических вычислениях и
; одновременная условная декрементация переменной цикла
        [count_B]   BDEC    .S2     loop01, count_B

; Ожидание завершения операции условного перехода
                    NOP 5
;   }
```

Листинг ассемблерной функции *cnvKIX()* приведен в приложении 3.
Обратите внимание на два момента.
Первое — это наличие пустых операторов «*NOP*» после команд «*LDH*», «*MPY*» и «*BDEC*».

И второе — как организуются циклические вычисления. Если выделить цикл *for()* вида:

```
// Начало цикла
for(count = 0; count < namberCoeff; count++){
.....................
     // Тело цикла;
.....................
} // Конец цикла
```

то ему будет соответствовать ассемблерный код:

```
; // Начало цикла
; Определение начального значения переменной цикла
          SUB  .L2X      namberCoeff_A, 1, count_B
loop01:
.............................
; // Тело цикла
.............................
; Условный переход при циклических вычислениях и
; одновременная условная декрементация переменной цикла
[count_B] BDEC .S2       loop01, count_B
; Ожидание завершения операции условного перехода
          NOP 5 ; // Конец цикла
```

Так как к настоящему моменту в проект включены две функции *cnvKIX()* — одна написанная на С, а вторая — на ассемблере, необходимо исключить из процесса компиляции файл с С-кодом функции. Для этого необходимо щелкнуть правой кнопкой мыши на имени файла «*cnvKIX.cpp*» в закладке «*File View*» (см. рис.51 п.1) и через появившееся контекстное меню вызывать окно настройки параметров компиляции щелкнув по пункту «*File Specific Options...*» (см. рис.51 п.2). В появившемся окне перейти на закладку «*General*» (см. рис.51 п.3) и поставить галочку в пункте «*Exclude file from build*» (см. рис.51 п.4). Заметим на будущее, что повторив описанную процедуру, но сняв галочку в пункте «*Exclude file from build*», можно подключить файл к проекту, то есть снова включить его в процесс компиляции.

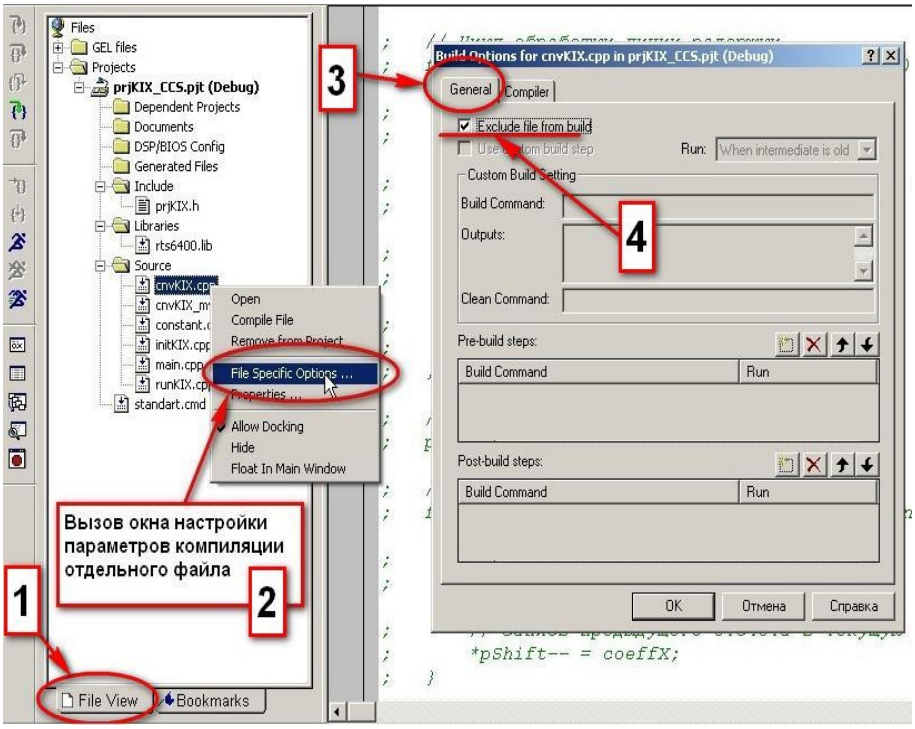

Рис.51. Исключения файла из процесса компиляции проекта.

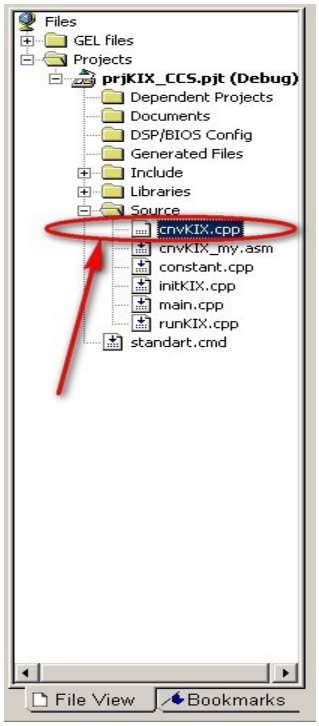

Рис.52. Закладка «File View» после отключения файла «cnvKIX.cpp».

Обратите внимание, что в окне «*File View*» изменилось отображение файла «*cnvKIX.cpp*» (см. рис.52). Файл может быть вызван в окно редактирования, его можно изменять, сохранять и т.д., но в процессе компиляции он не участвует.

Откомпилируйте проект заново, нажав кнопку полной компиляции проекта —  , и запустите проект на выполнение, нажав кнопку анимированного запуска —  . Если Вы все набрали верно, то компиляция должна пройти без ошибок, а после исполнения кода Вы получите выходной файл, идентичный файлу полученному в случае подключения к проекту С-кода функции *cnvKIX()*.

Сохраните настройки рабочего пространства, выбрав пункт «*File->Workspace->Save Workspace*», и закройте CCS.

5. Дополнительное задание

Запустите CCS. Откройте сохраненное рабочее пространство проекта, выбрав пункт «*File->Workspace->Load Workspace*». Исключите из процесса компиляции файл «*cnvKIX_my.asm*» и подключите к проекту файл «*cnvKIX.cpp*».

Замените С-код функции *cnvKIX()* на С-код приведенный в приложении 4. Это одноцикловый вариант функции. Откомпилируйте проект, запустите его на выполнение и убедитесь в корректной работе проекта.

Необходимо нарисовать блок-схему алгоритма, соответствующего одноцикловому варианту функции *cnvKIX()*, а так же написать ее ассемблерный аналог.

6. Приложения

Приложение 1.

Листинг файла *standard.cmd*

```
MEMORY
{
   IPRAM       : origin = 0x0,          len = 0x10000
   IDRAM       : origin = 0x80000000,   len = 0x300000
}

SECTIONS
{
        .text    > IPRAM
        .switch  > IPRAM
        .bss     > IPRAM
        .cinit   > IPRAM
        .const   > IPRAM
        .far     > IPRAM
        .stack   > IPRAM
        .cio     > IPRAM
        .sysmem  > IPRAM

}
```

Приложение 2.

Листинг функции *main()*

```
#include "prjKIX.h"

int main(void) {
    // Обьявление переменых
    word32 count;       // Переменная цикла

    // Инициализация указателя на контекстную структуру
    pCntx = &cntx;

    // Инициализация начального состояния
    initKIX(pCntx);

    // Цикл обработки входного буфера
    for(count = 0 ; count < 500; count++) {

        // Точка подключения зонда
        asm(" nop ");

        // Вызвать функцию обработки входного буфера
        runKIX(pCntx);

        // Точка подключения зонда
        asm(" nop ");
    }

    // Выход из программы
    return 0;
}
```

Приложение 3.

Листинг ассемблерного варианта функции *cnvKIX()*

```
;// Функция вычисление свертки
;#include "prjKIX.h"

;word16 cnvKIX(word16* pCoeff, word16* pShift, word16 namberCoeff, word32 constA){
; Объявление локальных переменных
        .asg    A3,  coeff_A             ; word32 coeff;
        .asg    A4,  pCoeff_A            ; 1-ый прм. фнк. -      word16* pCoeff
        .asg    A5,  coeffB_A            ; word16 coeffB
        .asg    A6,  namberCoeff_A       ; 3-ый прм. фнк. -  word16 namberCoeff
        .asg    A7,  rezult_A            ; word32 rezult;

        .asg    B0,  count_B             ; word32 count
        .asg    B3,  adrReturn_B         ; Адрес возврата
        .asg    B4,  pShift_B            ; 2-ый прм. фнк. - word16* pShift
        .asg    B5,  coeffX_B            ; word16 coeffX;
        .asg    B6,  constA_B            ; 4-ый прм. фнк. - word32 constA
        .asg    B7,  CSROLD_B            ; Регистр для хранения значения регистра CSR
        .asg    B15, SP                         ; Указательна вершину стека

        .sect   ".text"
        .global _cnvKIX__FPsT1si

_cnvKIX__FPsT1si:

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Запрет прерываний
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                                        MVC     .S2             CSR, CSROLD_B
                                        AND     .S2             CSROLD_B, -2, B1
                                        MVC     .S2             B1, CSR

;       // Инициализация локальных переменных
                                        ZERO    .D1     rezult_A        ; rezult = 0;

; Определение начального значения переменной цикла
                                        SUB     .L2X    namberCoeff_A, 1, count_B

;       // Цикл обработки линии задержки
loop01: ;       for(count = 0; count < namberCoeff; count++){

;               // Чтение текущего отсчета
                                        LDH     .D2T2   *pShift_B++, coeffX_B   ; pShift = *pShift++;

;               // Чтение соответствующего коэффициента фильтра
                                        LDH     .D1T1   *pCoeff_A++, coeffB_A   ; coeffB = *pCoeff++;

; Ожидание завершения операции чтения данных из памяти
                                        NOP 4

;               // Умножение отсчета на коэффициент
                                        MPY     .M1X    coeffX_B, coeffB_A, coeff_A     ; coeff = coeffX * coeffB;

; Ожидание завершения операции умножения
                                        NOP

;               // Накопление результата
                                        ADD     .S1     rezult_A, coeff_A, rezult_A     ; rezult += coeff;

; Условный переход при циклических вычислениях и
; одновременная условная декрементация переменной цикла
                [count_B]       BDEC    .S2     loop01, count_B

; Ожидание завершения операции условного перехода
                                        NOP 5
;       }

;       // Корректировка значения указателя
                                        SUB     .L2     pShift_B, 2, pShift_B   ; pShift--;

; Определение начального значения переменной цикла
                                        SUB     .L2X    namberCoeff_A, 2, count_B

;       // Сдвиг линии задержки
loop02: ; for(count = namberCoeff - 1; count > 0; count--){

;               // Чтение предыдущего отсчета
                                        LDH     .D2T2   *pShift_B[-1], coeffX_B ; coeffX = pShift[-1];

; Ожидание завершения операции чтения данных из памяти
                                        NOP 4

;               // Запись предыдущего отсчета в текущую ячейку
                                        STH     .D2T2   coeffX_B, *pShift_B--   ; *pShift-- = coeffX;

; Условный переход при циклических вычислениях и
; одновременная условная декрементация переменной цикла
                [count_B]       BDEC    .S2     loop02, count_B

; Ожидание завершения операции условного перехода
                                        NOP 5
;       }

; Переместить constA_B на сторону A.
                                        MV      .S1X     constA_B, A2

;       // Нормирование результата суммирования и перемещение возвращаемого значения в регистр A4
                                        SHR     .S1     rezult_A, A2, A4        ; rezult >>= constA;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Восстановление служебного регистра CSR
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                                        MVC     .S2             CSROLD_B, CSR

;       // Выход из функции
;       return rezult;
                                        B       .S2             adrReturn_B

; Ожидания завершения операции безусловного перехода (5-ть тактов)
                                        NOP 5
;}
```

Листинг одноциклового варианта С-кода функции *cnvKIX()*

```
// Функция вычисление свертки
#include "prjKIX.h"

word16 cnvKIX(word16* pCoeff, word16* pShift, word16 namberCoeff, word32 constA){
    // Объявление локальных переменных
    word16 coeffX;
    word16 coeffB;
    word32 count;
    word32 coeff;
    word32 rezult;

    // Корректировка значения
    namberCoeff--;

    // Установка указателей на конец буферов отсчетов и коэффициентов
    pShift += namberCoeff;
    pCoeff += namberCoeff;

    // Чтение текущего отсчета
    coeffX = *pShift--;

    // Чтение соответствующего коэффициента фильтра
    coeffB = *pCoeff--;

    //  Инициализация начального значения результата
    rezult = coeffX * coeffB;

    // Цикл обработки линии задержки
    for(count = 0; count < namberCoeff; count++){

        // Чтение текущего отсчета
        coeffX = *pShift--;

        // Запись текущего отсчета в следующую ячейку
        pShift[2] = coeffX;

        // Чтение соответствующего коэффициента фильтра
        coeffB = *pCoeff--;

        // Умножение отсчета на коэффициент
        coeff = coeffX * coeffB;

        // Накопление результата
        rezult += coeff;
    }

    // Нормирование результата суммирования
    rezult >>= constA;

    // Выход из функции
    return rezult;
}
```