CIC61508 Applets Examples

How to use the CIC61508 Secure SPI Mode Applets Concept

PRELIMINARY

Application Note

CIC61508, V0.9, 2011-06

FreeTextDocumentTopic

Edition 2011-06

Published by:
Hitex (U.K.) Limited.
University Of Warwick Science Park, Coventry, CV4 7EZ, UK

Legal Disclaimer

The information given in this document shall in no event be regarded as a guarantee of conditions or characteristics. With respect to any examples or hints given herein, any typical values stated herein and/or any information regarding the application of the product, Hitex (UK) Ltd. hereby disclaims any and all warranties and liabilities of any kind, including without limitation, warranties of non-infringement of intellectual property rights of any third party.

Information

For further information on technology, delivery terms and conditions and prices, please contact the nearest Hitex Office (www.hitex.com).

Document Change History

Date	Version	Changed By	Change Description
2010-11-21	0.2	M Beach	Initial Keil Version
2011-03-18	0.9	M Beach	Dave Bench SDCC added

We Listen to Your Comments
Is there any information in this document that you feel is wrong, unclear or missing?
Your feedback will help us to continuously improve the quality of this document.
Please send your comments (including a reference to this document) to:
comments@hitex.co.uk

Table of Contents

List of Figures

List of Tables

1 Introduction

The CIC61508's secure SPI mode is an alternative SPI protocol for programming the DFLASH calibration data area and advanced diagnostics. It is available in NOTREADY and DISABLED mode and is entered after writing 0x94 to the MODE SFR and using a password. It provides the following facilities:

1. DFLASH read, write, erase, programming
2. Read and write of IDATA and XDATA RAM
3. Read of PFLASH
4. Jump to address
5. Reset CIC61508

It is intended primarily for production-line use and can only be left via device reset.

1.1 Advanced Hardware Test Functions

For software development and production line programming of the DFLASH, these facilities are sufficient. However for low-level board testing during development and in the early stages of production test, additional functions are sometimes required. Due to limited ROM space in the CIC61508, a special RAM area has been reserved into which users can download simple C51 programs to control CIC61508 IO during manufacture and test. These programs can be used to implement common hardware test functions such as reading all the IO pins to check for short circuits or driving the SYSDIS pins to specified patterns to allow connected hardware to be tested.

1.1.1 CIC61508 Applet Runtime Environment

- Uses Keil C51 evaluation or Dave-Bench SDCC toolchains
- Eclipse IDE and batchfile compile and link for SDCC tools
- Max code size 256 bytes
- Max data size 128 bytes
- Applets inherit stack and SPI settings from CIC61508
- All IO pins (except SPI) accessible
- Applets can use existing SPI driver
- No interrupts allowed
- IO and memory configuration from CIC61508 normal operation mode is inherited.
- Applets can return to secure SPI mode, if required.

The remainder of this document describes the basic applet examples for the TC1782 evaluation board using the Keil, Dave-Bench and batchfile-based SDCC toolchains. The example applications used in this application note can be downloaded from:

http://www.hitexuk.co.uk/download/cic61508/CIC61508_Applets.zip

The unzip password is "hitexhitex".

2 CIC61508 Applet Examples

2.1 Pre-Requisites For Using Examples

To run the examples, you will need:

1. Tasking Tricore VX v3.3r1 Eclipse compiler
2. Keil PK51 evaluation version (free) or Dave Bench v2.01 SDCC tools
3. Hex2Carry.exe custom utility to convert C51 HEX386 files to a compilable C-array (supplied).
4. TC1782 SafeTkit board
5. PLS UDE UAD2 debugger
6. Hitex CIC61508 secure SPI mode driver library “CIC_SecSPI_Drv.a” for the TC1782 or example driver.
7. CIC61508 v2.6 or greater firmware.
8. Oscilloscope (not essential)

The supplied installer “CIC61508_Applets.exe” contains all of these elements except the toolchains. DAVE Bench and Keil PK51 Eval are available for download from www.infineon.com. The Tasking VX Eclipse tools are available for download from www.tasking.com.

The installation of the examples is covered later in this document.

2.2 CIC61508 Applet Memory Map

The CIC61508 memory available to applets is shown below.

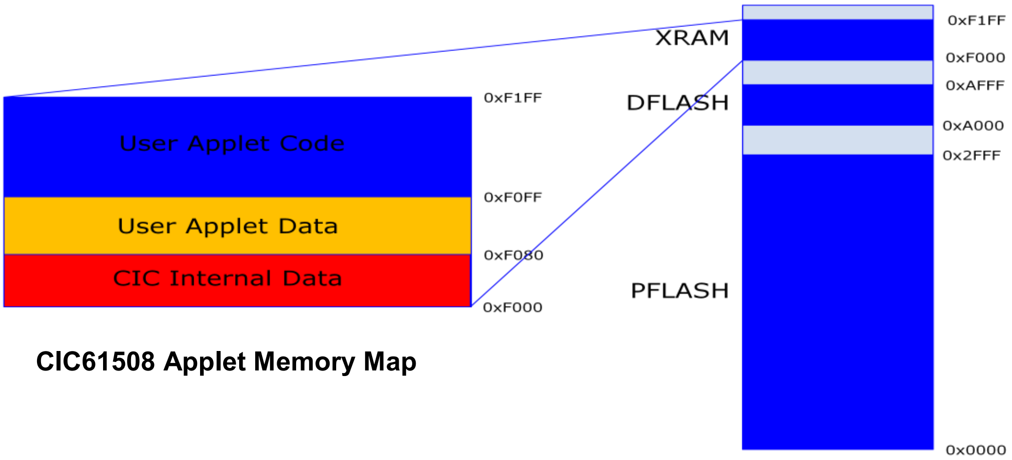


Figure 1 CIC61508 Applet Memory Map

Table 1 Applets Memory Areas

Area	Range
CODE Executable and constants:	0xF100 – 0xF1FF
XDATA variables:	0xF080 – 0xF0FF

This is described to the Keil PK51 compiler via a custom linker control file:

```
/* CIC Applet Linker Control File */
CLASSES (XDATA (X:0xF080-X:0xF0FF),
       HDATA (X:0xF080-X:0xF0FF),
       CODE (C:0xF100-C:0xF1FF),
       CONST (C:0xF100-C:0xF1FF),
       ECODE (C:0xF100-C:0xF1FF),
       HCONST (C:0xF100-C:0xF1FF))

/* Create a symbol to represent the CIC's secure SPI handler */
ASSIGN(vRead_PolledSSC(0x1000),
      XC866_FLASH_TYPE1_Erase_Func(0xDFF9),
      uiPolledSSC_TxData(0xF024),
      uiPolledSSC_RxData(0xF022),
      uiSecSPI_data(0xF028),
      uiSecSPI_addr(0xF02A))
```

For SDCC, these settings are held in the project settings memory map page or in the case of a batchfile-based system, in the linker command line:

```
C:/DAVE-Bench-201\SDCC_XC800\bin\sdcc main.rel SecSPIDefs.rel IO_ReadTest.rel crtstart.rel
crtxinit.rel crtxstack.rel startupxc866.rel --no-xinit-opt --main-return --debug -
I"C:\c51dev\CICApps\App2_SDCC\Common\inc" -I"C:/DAVE-Bench-201\SDCC_XC800\include" -
I"C:/DAVE-Bench-201\SDCC_XC800\include\xc800" -I"C:/DAVE-Bench-
201\SDCC_XC800\include\asm\xc800" -mXC800 -pXC866_4FR --model-small --iram-size 0x100 -Wl -
bBSEG=0x20 --xram-loc 0xF080 --xram-size 0x80 --code-loc 0xF100 --code-size 0x100 --data-
loc 0x00 --idata-loc 0x80 --stack-loc 0x80 -Wl -bPSEG=0xF000 -o "App2_SDCC.ihx"
```

Provided these setups are used in any applet, the memory map requirements will be met.

2.3 Principle Of Operation

The overall applet sequence of operations is set out in the diagram below.

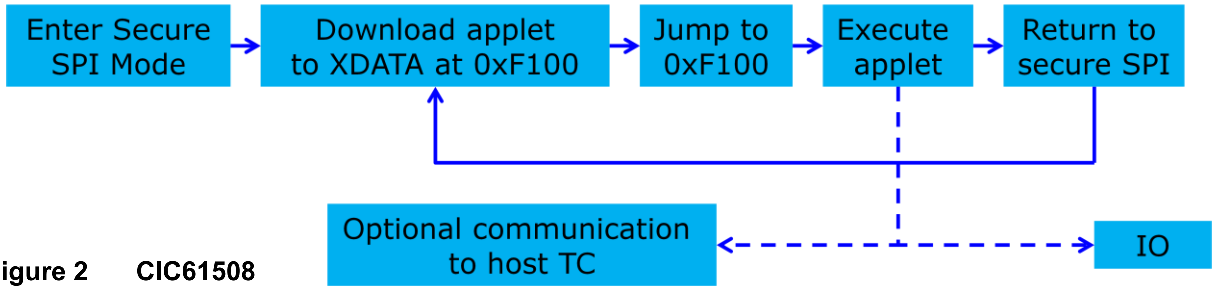


Figure 2 CIC61508 Applet Development Cycle

The applet development cycle is:

1. Create applet in PK51 or DAVE-Bench/SDCC, using the special linker file or memory map settings.
2. Create an Intel 386 format Hexfile.
3. Convert the hexfile to a compilable C array using the custom Hex2Carry.exe utility (supplied), called from the uVISION "User" tab or from the DAVE Bench Build-Steps-Pre-Build steps command page.
4. Copy the created C51CodeAppX.C and C51CodeAppX.H to the Tricore application's CIC61508_Applet directory.
5. Write a suitable interface to the applet (i.e. driver), including the provided applet loader from the CIC_SecSPI_Drv.a library.
6. Compile the Tricore application
7. Load application into Tricore and run.

The examples provided here use this method but have pre-written applets and Tricore TC1782 drivers.

3 Applets Examples Installation

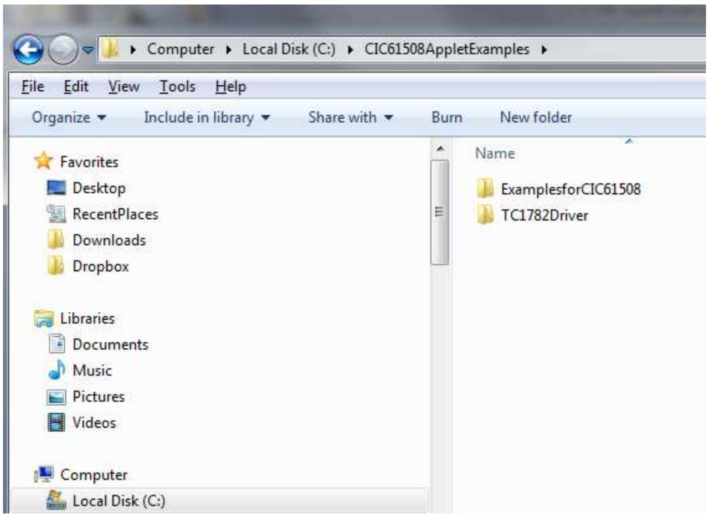
The applet examples programs that execute on the CIC61508 and the TC1782 example driver are supplied in a single installation utility “CIC61508_Applets.exe”.

3.1 Running The Applets Examples Installer

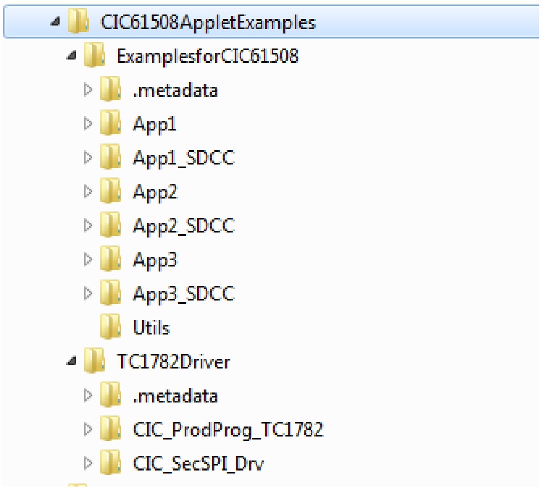
Click on the installer .EXE file and follow the on-screen instructions. While you will be given the opportunity to select an alternative installation directory, correct operation can only guaranteed by taking the default “C:\ CIC61508AppletExamples”.

The installer will finish by running the Microsoft Visual C redistribution utility. If you already have this present on your PC, the installer will ask if you want to “Remove” or “Repair” the existing version. Please choose “Repair”.

The result will be a directory structure as shown below:



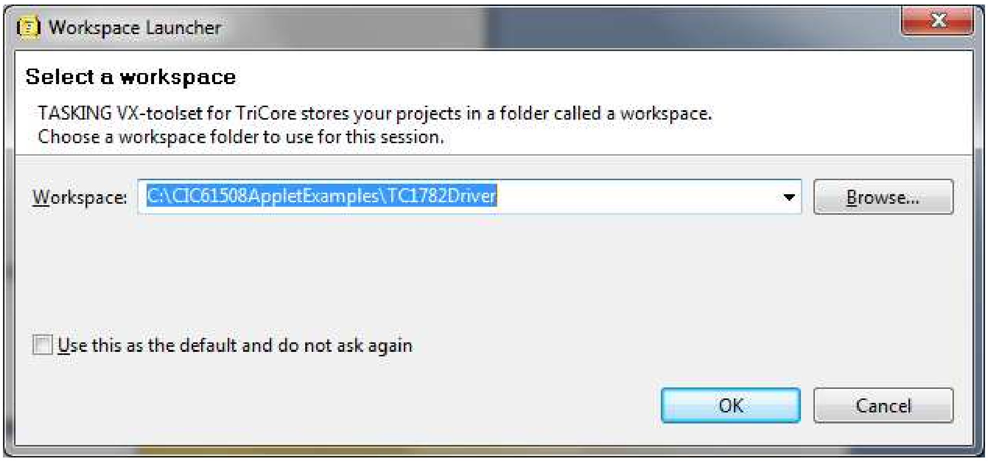
This consists of a directory tree for the CIC61508 applet programs and another for the TC1782 driver:



No special installation steps are required for the Keil PK51 and DAVE Bench SDCC toolchains. However the Tasking Tricore VX toolchain requires a setup stage, covered in the next section.

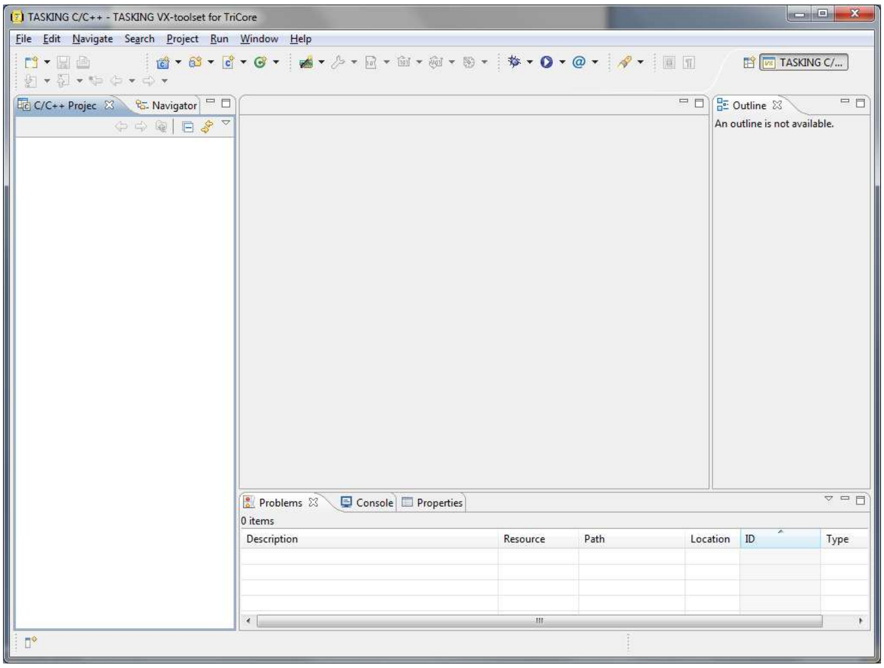
3.2 Configuring The Tasking VX TC1782 Toolchain

The TC1782 driver application must be imported into the Tasking Eclipse IDE from the directory "C:\CIC61508AppletExamples\TC1782Driver". Start the Eclipse IDE and when prompted for a Workspace, enter this directory name:

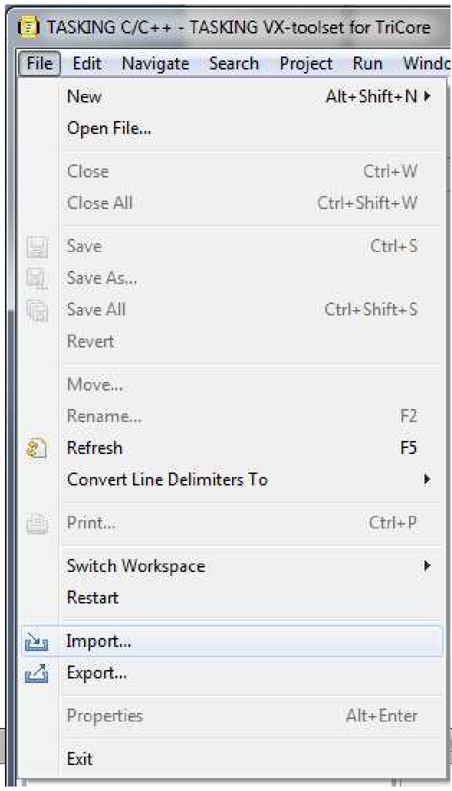


Once the Eclipse has initialized, it will display a Welcome screen.

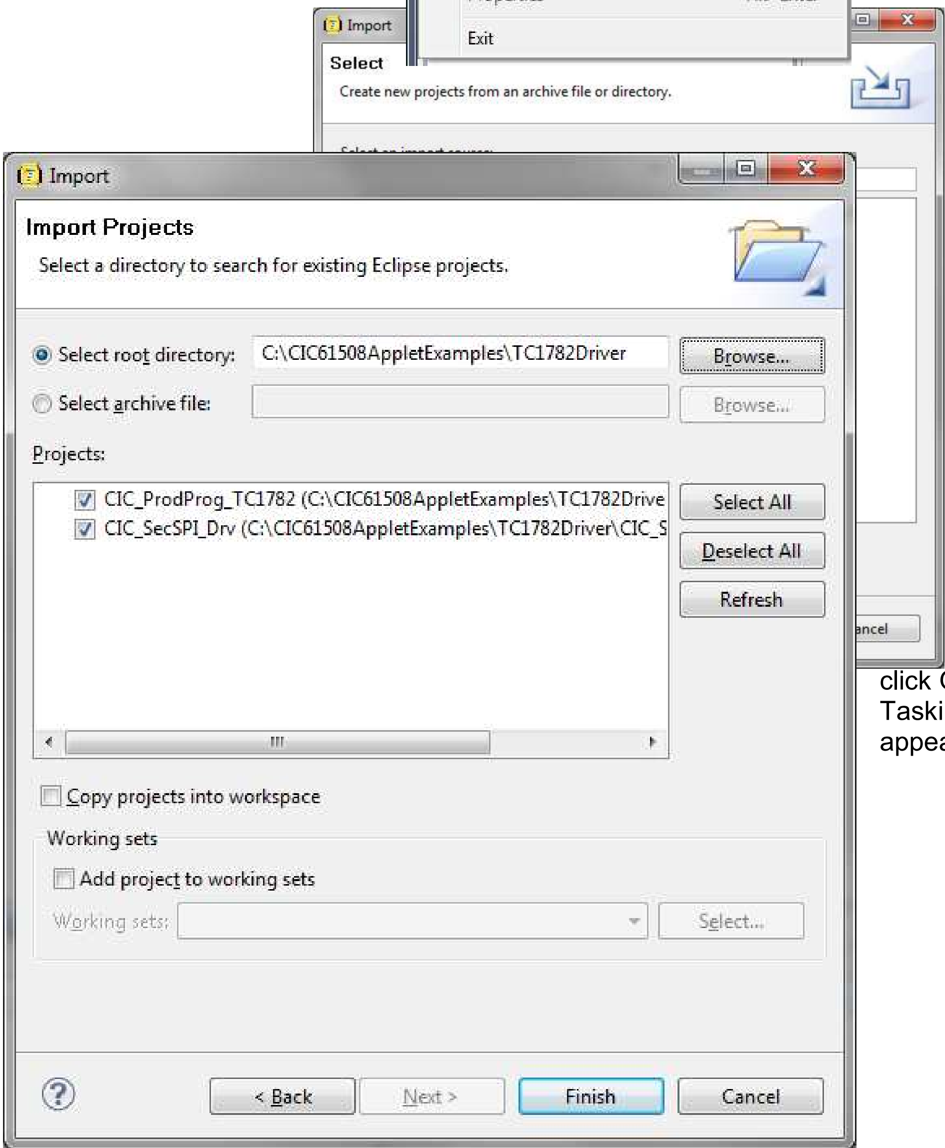
Remove this by clicking on the cross to the right of “Welcome” to reveal an empty workspace.



The TC1782 driver application is supplied as a complete existing project so it can be imported directly into the Workspace by using the File-Import function and



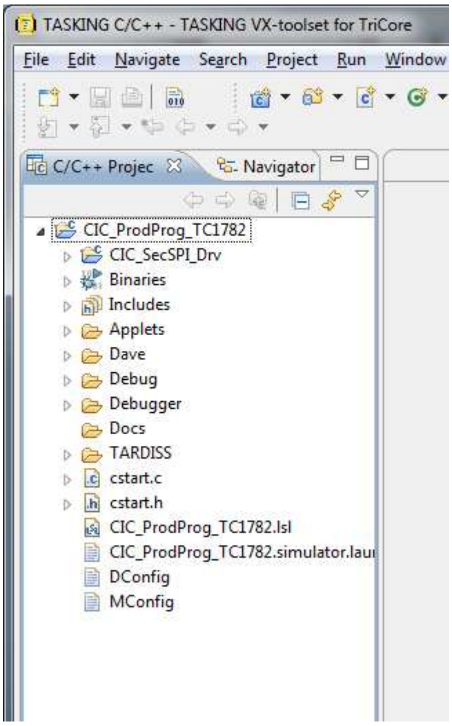
then selecting General-Existing Projects Into Workspace:



Browse to the directory and click OK. The names of the two Tasking TC1782 projects required will appear in the Projects box.

Make sure they are selected before clicking Finish.

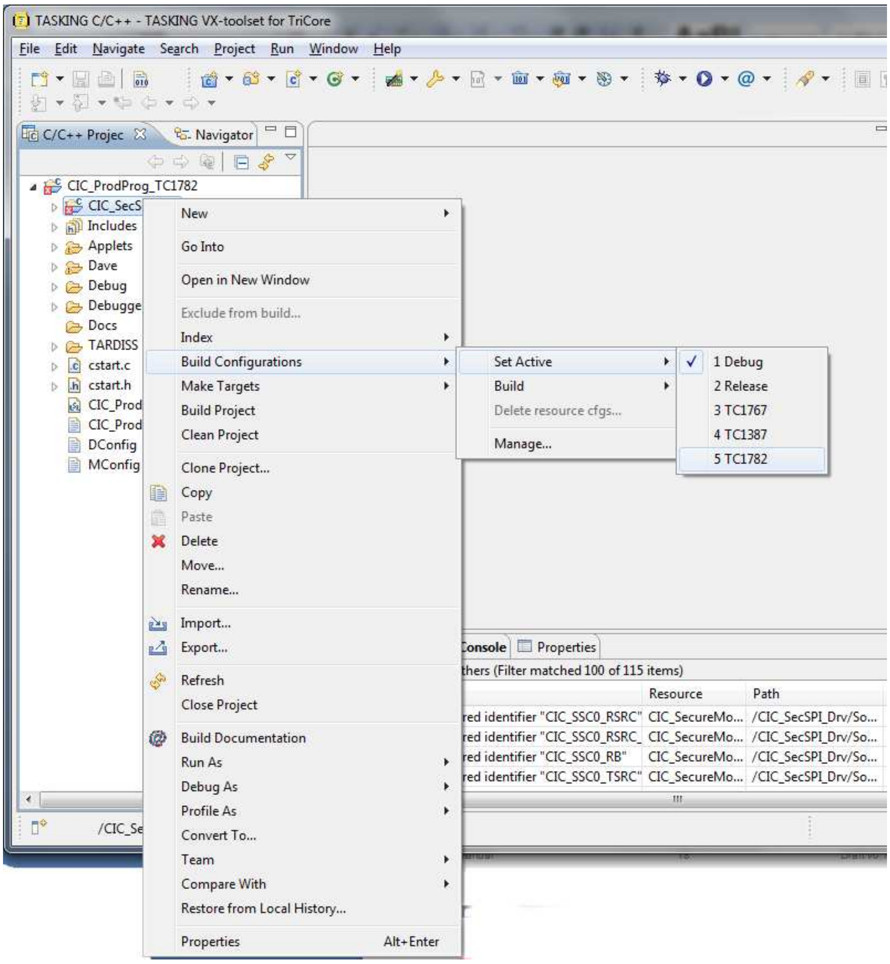
Two projects will now be present in the workspace.



The project “CIC_ProdProg_TC1782” is the main project that contains the loading and execution of the CIC61508 applets. The project CIC_SecSPI_Drv is a sub project that provides the CIC61508 secure mode SPI driver. It is only necessary to build the former project as it will automatically build and link in the sub-project.

3.2.1 Checking the Project Configuration

It is necessary to check that the CIC_SecSPI_Drv project is configured for the TC1782. This is done via the Build-Configurations menu. Click on the project
name in the C/C++ Projects panel and then right-click to reveal the Build Configurations menu.



Select
active Build
Finally, to
project
successful,
Project.

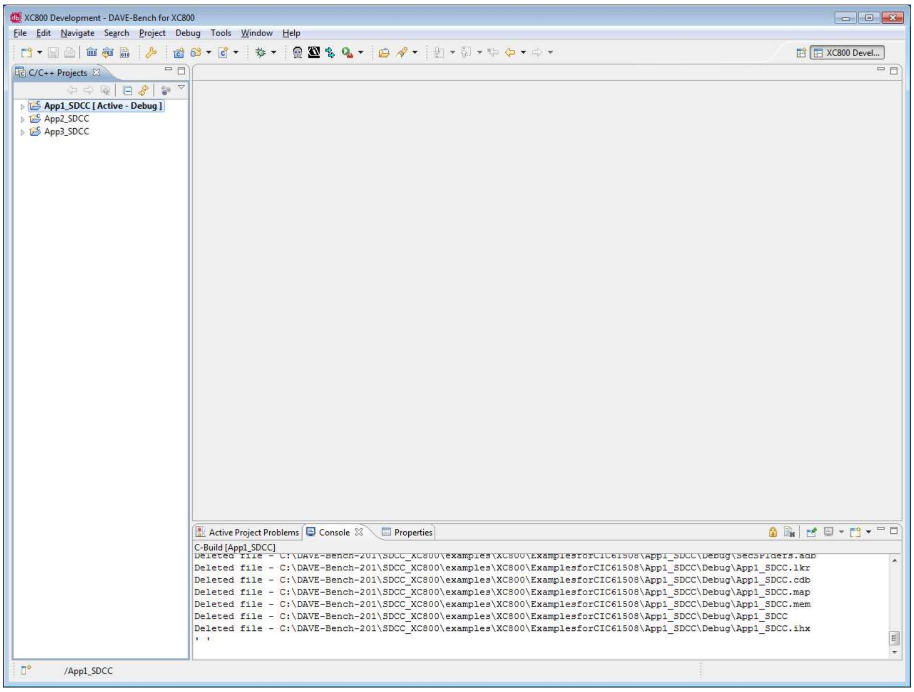
“TC1782” as the
Configuration.
check that the
import has been
select Project-Build

4 Using The Dave-Bench SDCC Toolchain

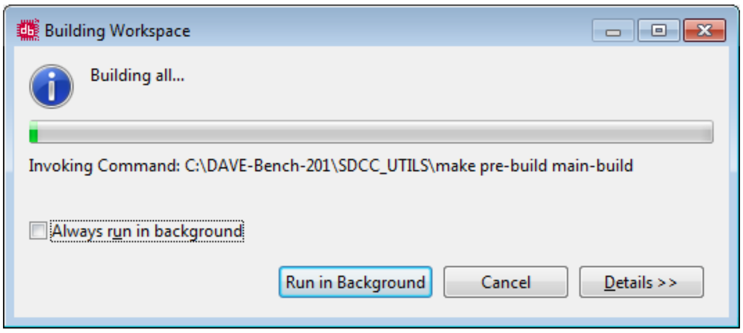
The Eclipse-based Dave Bench XC800 toolchain has been preconfigured to compile and link the 3 applet examples. To build the examples, start Dave Bench and set the Workspace to:

C:\CIC61508AppletExamples\ExamplesforCIC61508

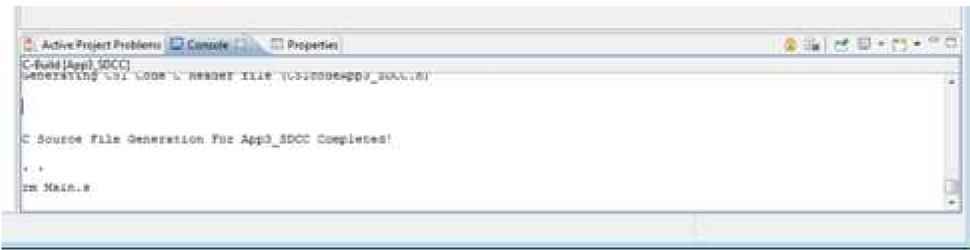
After initializing, the Bench will show three projects:



The project "App1_SDCC" will be shown as the active project. To check that the installation has been successful, select from the top line, Project-Build All (Ctrl +B).



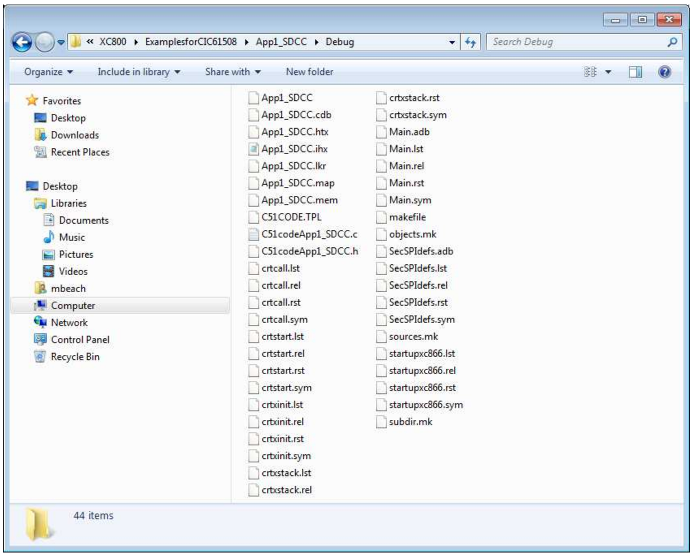
The build operation should complete successfully as shown.



For the App1_SDCC project, the output from the build process will be two source files:

"C51codeApp1.C"

"C51codeApp1.H"



These contain the applet as a compilable C const array:

```
// Flag that this is C51code.c
#define _C51CODE_

// Include the necessary include files
#include "C51codeApp1_SDCC.h"

// Create the array image of the applet
unsigned char CIC61508_AppletCodeApp1[CIC61508_APPLET_CODE_SIZE] = {
           0x02, 0xf1, 0x0a, 0x12, 0xf1, 0x0d, 0x80, 0xfe,
           0xe4, 0x73, 0x02, 0xf1, 0x03, 0xaa, 0xb1, 0x7b,
           0x00, 0x43, 0x02, 0x03, 0x8a, 0xb1, 0x75, 0x89,
           0x02, 0x75, 0x8a, 0x00, 0x75, 0x8c, 0x00, 0xd2,
           0x8c, 0xaa, 0xb0, 0x7b, 0x00, 0x63, 0x02, 0x02,
           0x8a, 0xb0, 0x10, 0x8d, 0x02, 0x80, 0xfb, 0x80,
           0xf0, 0x22, };



/* Size of this applet */

unsigned int const uiCIC61508_AppletSizeApp1 = 0x32U;
```

The name of the C array created by Hex2Carray.exe will have a suffix related to the second parameter passed on the Hex2Carray.exe command line. Here it was:

```
Hex2Carray.exe App1_SDCC App1
```

These auto-generated files are then included in the TC1782 driver example for download and execution in the CIC61508.

4.1 Adapting Dave Bench For CIC61508 Applet Creation

The adaptations made to the default XC800 project settings to allow applets to be created are covered below.

4.1.1 Setting The CIC61508 Applet Memory Map

The applets memory map must be entered:

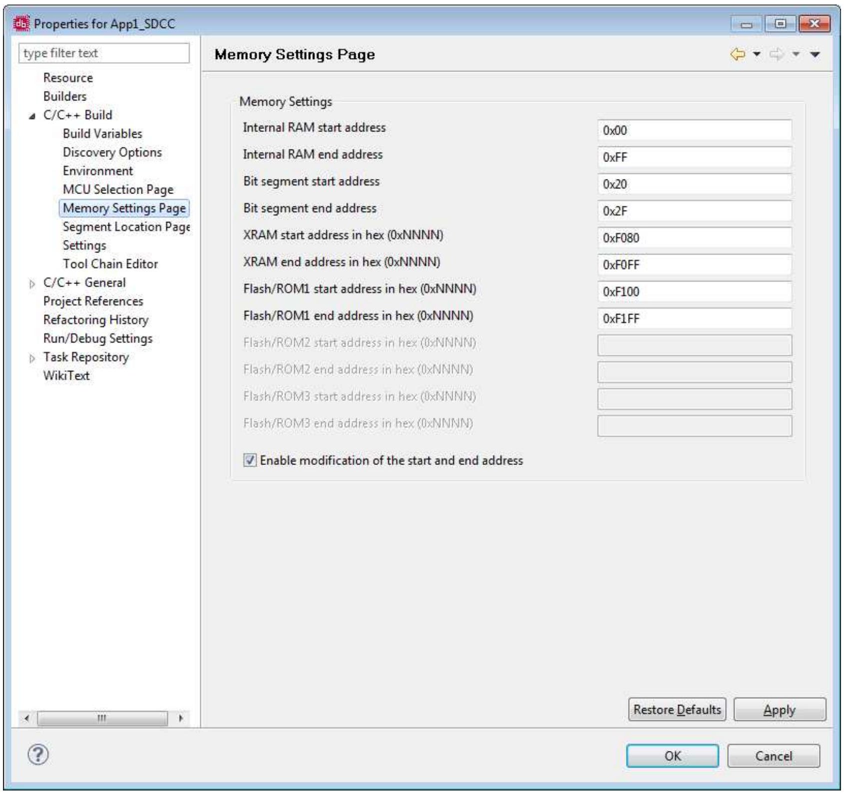


Figure 3 DAVE Bench SDCC Memory Map Page

4.2 Running The Hex To C Converter

The Hex2Carray.exe utility is assumed to be in the applet project root directory. Here the HEX file output from the SDCC toolchain is “App1_SDCC.ihx”. The name of the C array created will be appended with the name “App1” (second parameter).

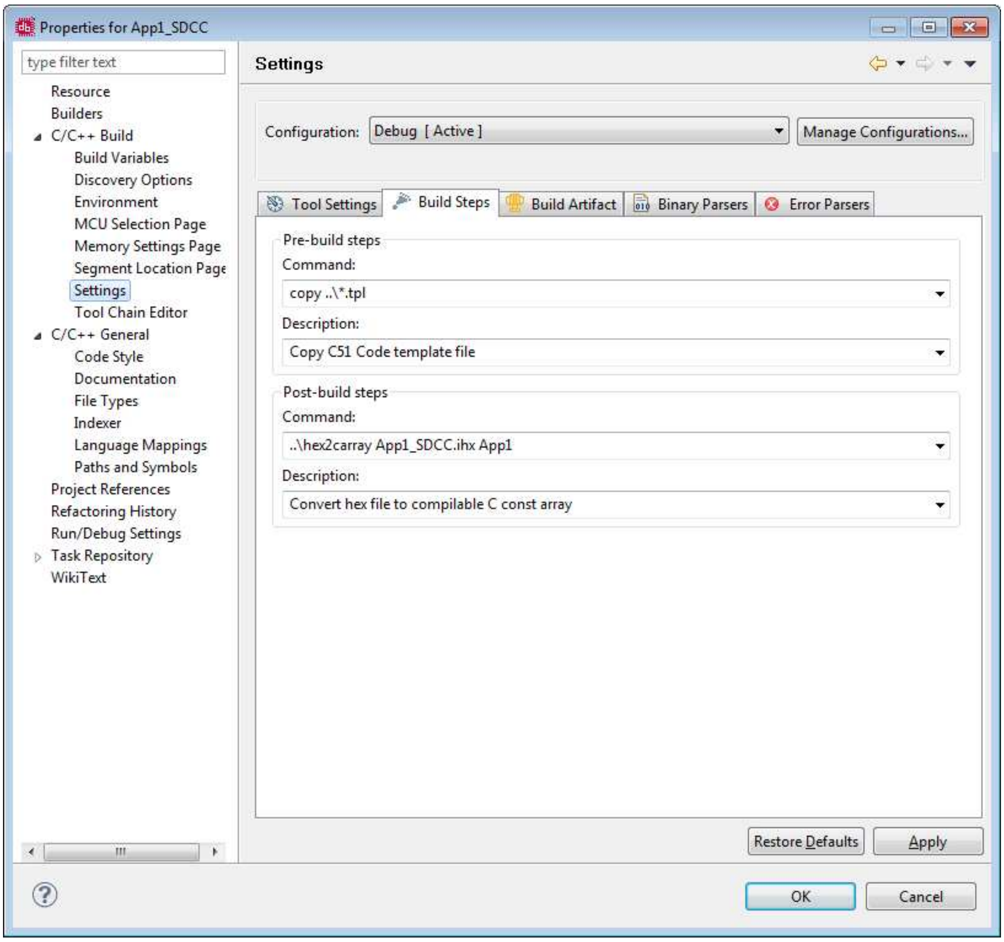


Figure 4 DAVE Bench SDCC Build Steps Page

4.2.1 Naming The Hex File Output

The SDCC tools must be told to create a HEX output with the extension “.IHX”:

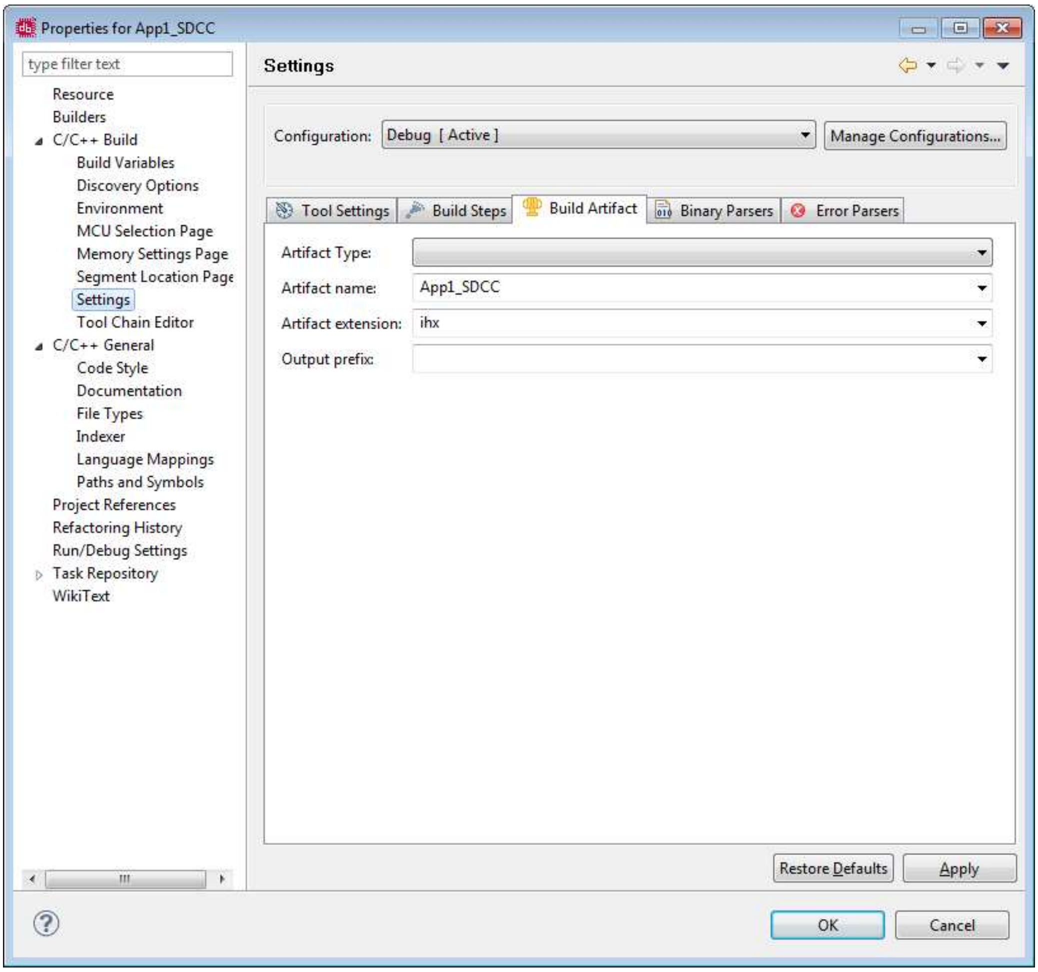


Figure 5 DAVE Bench SDCC Output File Naming Page

5 Special Settings Required In Keil VISION

The uVISION projects are configured for the XC866-4FR, using the LX51 linker. As supplied, the uVISION IDE has been pre-configured with the necessary special settings. To aid the creation of user's own applets, the relevant settings are shown in the following screen captures:

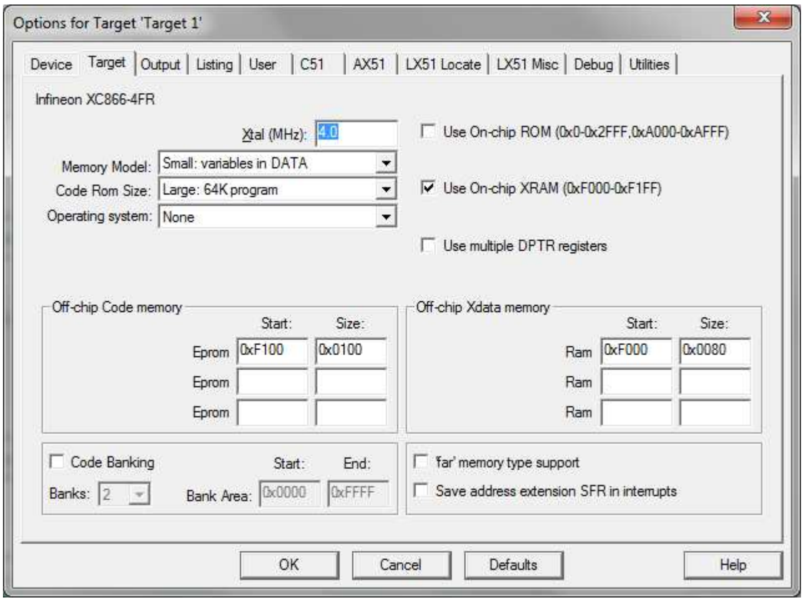


Figure 6 uVISION Target Tab

The HEXfile format must be Intel386. The name of the applet is also specified here.

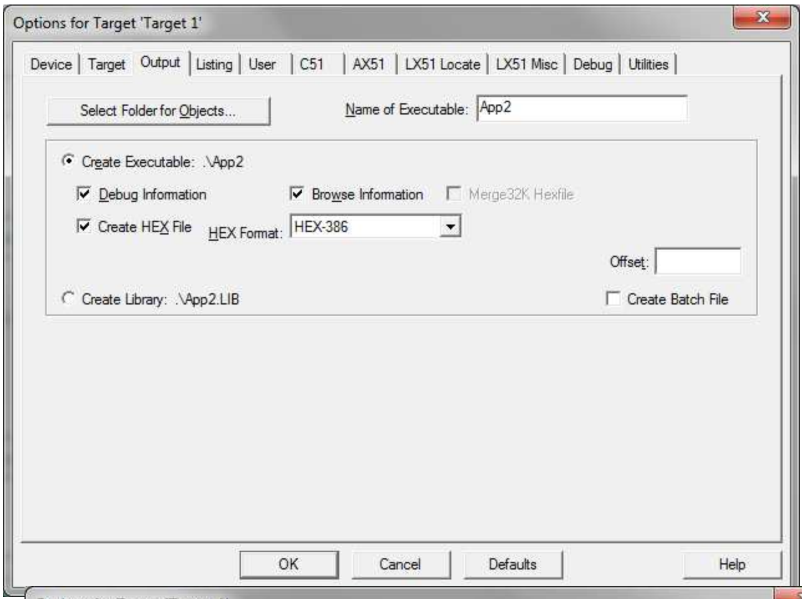


Figure 7 uVISION Output Tab

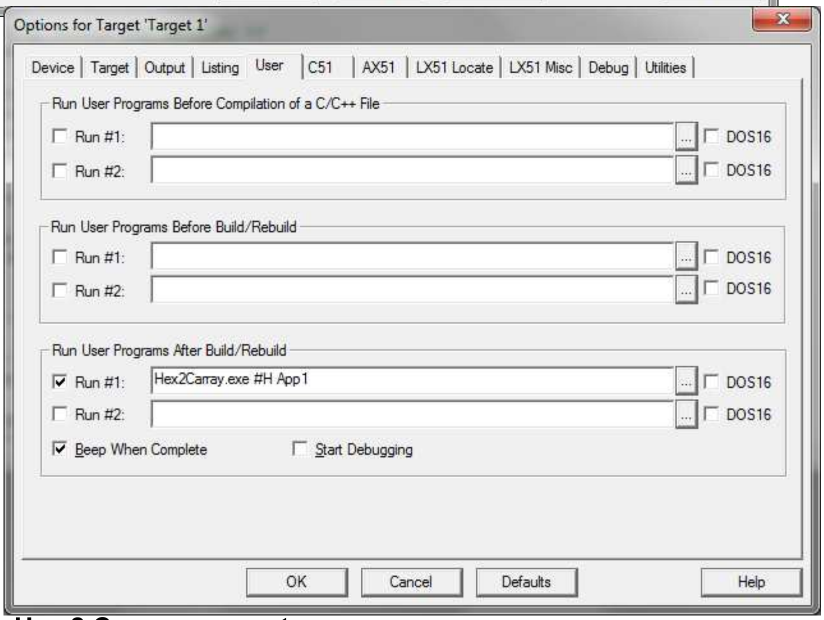


Figure 8 Interfacing the Hex 2 C-array converter

Here, the Hex2Carray tool is called from the User –Run box after the HEXfile has been created. The command line is:

```
Hex2Carry <HEXFILENAME> <APPLETNAME>
```

The uVISION IDE supplies the HEXfile name from the Output tab via the "#H" token. The application name chosen by the user is supplied by the second command line parameter.

```
Hex2Carray.exe #H App2
```

The result is that the PK51 program will exist within a C source file called in this case, C51CodeApp2.H and have a header file C51CodeApp2.H. These provide the input to the Tasking CTC application that will load and run the applet in TC1782 board.

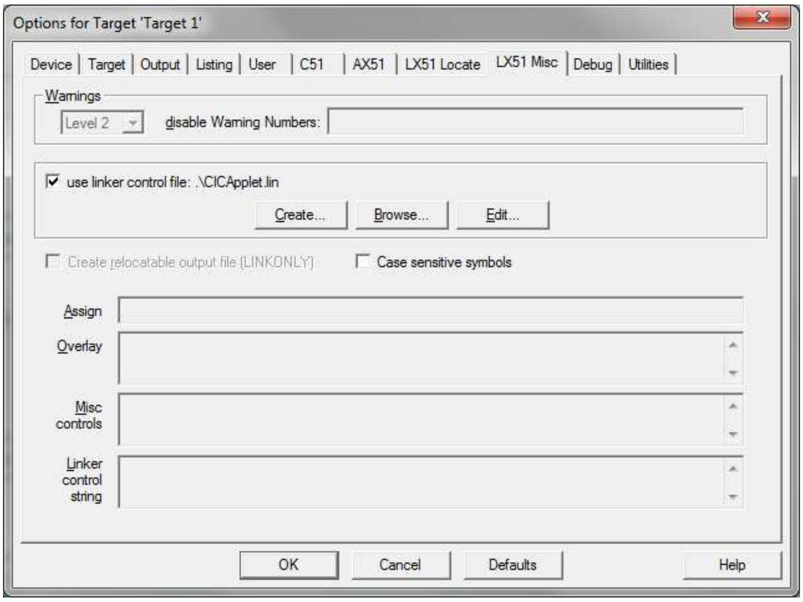


Figure 9 Specifying the linker control file

6 Using Batchfiles For CIC61508 Applets in SDCC

Each of the three supplied SDCC applet examples may also be compiled and linked using simple MS-DOS batchfiles. In each applet subdirectory, make batchfiles can be found. These are of the form:

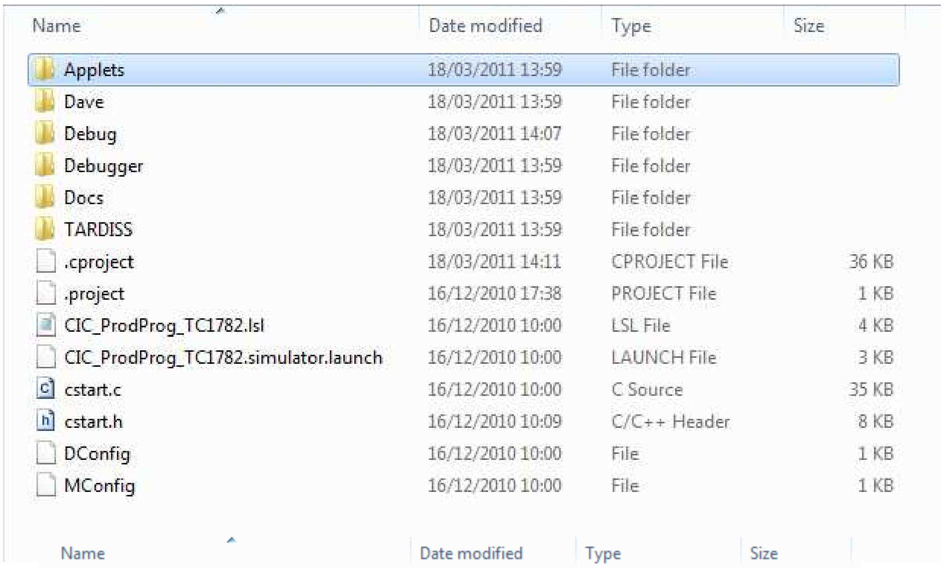
```
echo off
cls
echo *
echo * Assemble Startup Files
C:/DAVE-Bench-201\SDCC_XC800\bin\as-xc800 -plosgffcx crtstart.s
C:/DAVE-Bench-201\SDCC_XC800\bin\as-xc800 -plosgffcx crtxinit.s
C:/DAVE-Bench-201\SDCC_XC800\bin\as-xc800 -plosgffcx crtxstack.s
C:/DAVE-Bench-201\SDCC_XC800\bin\as-xc800 -plosgffcx startupxc866.s
echo *
echo * Compile Applet Files
C:/DAVE-Bench-201\SDCC_XC800\bin\sdcc -c SecSPIDefs.c --main-return --debug -I"C:\c51dev\CICApps\App3_SDCC" -I"C:/DAVE-Bench-201\SDCC_XC800\include" -I"C:/DAVE-Bench-201\SDCC_XC800\include\xc800" -I"C:/DAVE-Bench-201\SDCC_XC800\include\asm\xc800" -mXC800 -pXC866_4FR --model-small --iram-size 0x100
C:/DAVE-Bench-201\SDCC_XC800\bin\sdcc -c main.c --main-return --debug -I"C:\c51dev\CICApps\App3_SDCC" -I"C:/DAVE-Bench-201\SDCC_XC800\include" -I"C:/DAVE-Bench-201\SDCC_XC800\include\xc800" -I"C:/DAVE-Bench-201\SDCC_XC800\include\asm\xc800" -mXC800 -pXC866_4FR --model-small --iram-size 0x100
C:/DAVE-Bench-201\SDCC_XC800\bin\sdcc main.rel SecSPIdefs.rel crtstart.rel crtxinit.rel crtxstack.rel startupxc866.rel --no-xinit-opt --main-return --debug -I"C:\c51dev\CICApps\App3_SDCC\Common\inc" -I"C:/DAVE-Bench-201\SDCC_XC800\include" -I"C:/DAVE-Bench-201\SDCC_XC800\include\xc800" -I"C:/DAVE-Bench-201\SDCC_XC800\include\asm\xc800" -mXC800 -pXC866_4FR --model-small --iram-size 0x100 -Wl -bBSEG=0x20 --xram-loc 0xF080 --xram-size 0x80 --code-loc 0xF100 --code-size 0x100 --data-loc 0x00 --idata-loc 0x80 --stack-loc 0x80 -Wl -bPSEG=0xF000 -o "App3_SDCC.ihx"
echo *
echo * Make a copy of the hex file
copy App1_SDCC.ihx App1_SDCC.hex
copy *.ihx debug
copy *.CDB debug
echo *
echo * Convert HEX to C const array
Hex2Carray.exe App1_SDCC.ihx App1_SDCC
echo *
echo * Completed
```

They show at the lowest level how the SDCC tools are used.

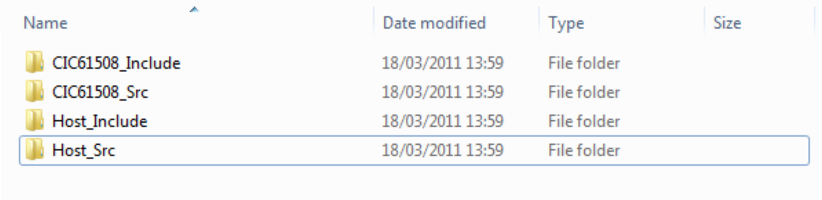
7 Applet Examples In Detail

7.1 Including CIC61508 Applets In Tricore Applications

In the example, the files containing the applet C const arrays are place in a directory “Applets”, subdirectory “CIC61508_Src”:



Within this further directory are 4 directories:



The “Host_Src” directory contains a module “Host_Applet_Manager.c” that holds the Tricore code that loads the applet into the CIC65108 and runs it. For example, App1 is managed by:

```
void vExecute_App1(void)
{

        /* Load a CIC61508 applet */
        CIC_status= uiLoad_CIC61508Applet(CIC61508_AppletCodeApp1,
                                          uiCIC61508_AppletSizeApp1,
                                          CIC61508_APPLET_CODE_BASE);
        if(CIC_status != CIC_No_Error)
        {

                /* User can decide how to handle errors */
                while(1) { ; }

        }
}
```

The function uiLoad_CIC61508Applet() is located in the CIC_SecSPI_Drv project and takes care of all the preparatory actions required to get an applet into the CIC61508 RAM, apart from entering Secure SPI Mode, which is done by the main() function.

7.2 Example Applets Tricore Program

The example applets are called from a single demonstration main() function so that they run sequentially.

```
   /* Run Applet Examples */

   /* Toggle SYSDISA every 19.2 us */
#ifdef _ENABLE_APP1_

   vExecute_App1();  /* Warning - this does not allow the CIC to return so a hard reset will
be needed */

#endif

   /* Continuously read all CIC61508 pins (except SPI) for 30secs */
   vExecute_App2();

   /* Restart CIC as App2 destroys IO setup */
   uiReset_CIC();

   /* Enter Secure SPI Mode */
   CIC_status = uiCIC_SPI_EnterSecureMode() ;

   /* Continuously toggle the SYSpins for 30 secs */
   vExecute_App3();

   /* Wait for a short time by using the reset timer in the SPI driver */
   CIC_SPI_state = Start_CS_Wait_For_CIC_Reset ;

   /* Set state of SYSDIS Pins */
   vSYSDIS_State(0,0,0);   /* SYSDISA = 0, SYSDISB = 0, SYSDISC = 0 */

   /* Wait for a short time */
   CIC_SPI_state = Start_CS_Wait_For_CIC_Reset ;

   vSYSDIS_State(0,0,1);   /* SYSDISA = 0, SYSDISB = 0, SYSDISC = 1 */

   /* Wait for a short time */
   CIC_SPI_state = Start_CS_Wait_For_CIC_Reset ;

   vSYSDIS_State(0,1,0);   /* SYSDISA = 0, SYSDISB = 1, SYSDISC = 0 */

   /* Wait for a short time */
   CIC_SPI_state = Start_CS_Wait_For_CIC_Reset ;

   vSYSDIS_State(0,1,1);   /* SYSDISA = 0, SYSDISB = 1, SYSDISC = 1 */

   /* Wait for a short time */
   CIC_SPI_state = Start_CS_Wait_For_CIC_Reset ;

   vSYSDIS_State(1,0,0);   /* SYSDISA = 1, SYSDISB = 0, SYSDISC = 0 */

   /* Wait for a short time */
   CIC_SPI_state = Start_CS_Wait_For_CIC_Reset ;

   vSYSDIS_State(1,0,1);   /* SYSDISA = 1, SYSDISB = 0, SYSDISC = 1 */

   /* Wait for a short time */
   CIC_SPI_state = Start_CS_Wait_For_CIC_Reset ;

   vSYSDIS_State(1,1,1);   /* SYSDISA = 1, SYSDISB = 1, SYSDISC = 1 */

   /* End of test area */
```

Please note that Example 1 is a special case as it does not return. This is why it has to be specifically enabled by adding:

```
#define _ENABLE_APP1_
```

At the top of the MAIN.C.

All the example applet drivers are located in “Host_Applet_Manager.C”.

7.3 Applet Example 1

7.3.1 C51 Applet

This is the simplest example. It makes sure that the SYSDISA (Port3.0) pin is enabled as an output and then toggles it every 19.2us. The toggle rate is controlled by the CIC61508's unused Timer0. This applet does not return to the normal secure SPI mode.

```
/* Main program */
void main(void)
{

   /* Declare locals */

     /* Set the direction of Port3.0 and Port3.1 to output */
     P3_DIR = (P3_DIR | SPC_P3_DIR_MASK);

   CPU_TMOD = 0x02U; /* 8-bit autoreload mode */
   CPU_TL0 = 0x00U;  /* 19.2us period */
   CPU_TH0 = 0x00U;

   TR0 = 1U; /* Start timer */

   while(1U)
   {

      /* Toggle P3.3 */
      P3_DATA ^= SPC_P3_SYSDIS_A;

      /* Wait for 19.2us timeout */
      while(TF0 == 0U)
      {
        ;
      }

        /* Clear timer0 overflow flag */
      TF0 = 0U;

   }

} /* Total code size = 31 bytes */
```

7.3.2 Tricore Driver

This is the simplest possible applet driver. It loads the applet contained in C51CodeApp1.C to 0xF100 in the CIC61508 and then executes a JumpToAddress command.

```
void vExecute_App1(void)
{

   /* Load a CIC61508 applet */
   CIC_status = uiLoad_CIC61508Applet(CIC61508_AppletCodeApp1,
                                      uiCIC61508_AppletSizeApp1,
                                      CIC61508_APPLET_CODE_BASE);

   if(CIC_status != CIC_No_Error)
   {

      /* User can decide how to handle errors */
      while(1) { ; }

   }
}
```

The loader function exists in the Secure SPI mode library. It consists of just:

```
CIC_ErrorType uiLoad_CIC61508Applet(uint8 *uiAppletCode, uint16 uiAppletSize,
                                    uint16 uiAppletCodeBaseAddr)
{

   /* Declare locals */
   uint16 uiI;
   CIC_ErrorType uiCIC_status;

   /* Load a CIC61508 applet */

   for(uiI = 0x00 ; uiI < uiAppletSize ; uiI++)
   {

      uiCIC_status = uiCIC_WriteSecureAddress(uiAppletCode[uiI],
                                              (uiAppletCodeBaseAddr + uiI), XDATA);

   }

   if(uiCIC_status == CIC_No_Error)
   {

      /* Jump to applet */
      uiCIC_status = uiCIC_JumpAddress(0x00U, uiAppletCodeBaseAddr , Jump_Address);

   }

   return(uiCIC_status);
}
```

7.3.3 Running Example 1

This example is disabled as supplied. To enable it, add:

```
#define _ENABLE_APP1_
```

At the top of the MAIN.C and rebuild the Tricore application. Start the PLS debugger and load the C:\CIC_ProdProg_TC1782\Debugger\UDE.WSX workspace. Load the Tricore application and run it. You should find that SYSDISA now toggles every 19.2us.

This kind of applet is useful where the applet cannot return. An example may be where the SPI pins are being checked on an automatic tester, under which circumstances no further CIC communications would be possible.

7.4 Applet Example 2

The standard secure SPI mode read and write functions do not allow the state of CIC61508 IO pins to be directly checked. This applet provides a simple way to read all the CIC IO port pins (except those used for SPI) via the normal secure READ function in the Secure Mode library.

The example relies on an XDATA array at X:0xF080 which is loaded by the applet with the values currently on the IO ports. It makes sure that all port pins (except SPI) are converted into input mode first. Thus this allows pins that are normally outputs to be read.

If you have just run Example 1, make sure you remove the

```
#define _ENABLE_APP1_
```

previously added otherwise Example 2 will never be reached.

```
void IO_ReadTest(void)
{

      /* Declare Locals */

   /* Copy the values of the port pins to XDATA */
   uiPx_Image[0] = P0_DATA;
   uiPx_Image[1] = P1_DATA;
   uiPx_Image[2] = P2_DATA;
   uiPx_Image[3] = P3_DATA;

   /* XDATA pin images can now be read via secure READ command */

}
```

The applet exits immediately back to secure SPI mode.

7.5 Example 2 Tricore Driver

This driver uses a different approach to Example 1. Here the applet that reads the IO port pins into the XDATA uiPx_Image[] runs once and then exits. The uiCIC_ReadSecureAddress() library function is then used to read the port images into global variables "uiCIC61508_Px" in the Tricore application. These can be displayed in the debugger real time update window.

However as the port-capture applet is now loaded into the CIC61508 at 0xF100, we can call it repeatedly to update the port pin status in the Tricore global variables. If the ADC pots on P2 are adjusted, the uiCIC61508_P2 value in the debugger watch window will change.

```
void vExecute_App2(void)
{

   /* Declare locals */
   uint32 uiTimer;

   /* Load a CIC61508 applet */
   CIC_status = uiLoad_CIC61508Applet(CIC61508_AppletCodeApp2,
                                     uiCIC61508_AppletSizeApp2,
                                     CIC61508_APPLET_CODE_BASE);
   if(CIC_status != CIC_No_Error)
   {

      /* User can decide how to handle errors */
      while(1) { ; }

   }

   /* Run this test for about 30 secs */
   for(uiTimer = 0x00 ; uiTimer < 20000U ; uiTimer++)
   {

      /* Read IO Port images in XDATA */
      uiCIC61508_P0 = (uint8)uiCIC_ReadSecureAddress(0xF080, XDATA) ;
      uiCIC61508_P1 = (uint8)uiCIC_ReadSecureAddress(0xF081, XDATA) ;
      uiCIC61508_P2 = (uint8)uiCIC_ReadSecureAddress(0xF082, XDATA) ;
      uiCIC61508_P3 = (uint8)uiCIC_ReadSecureAddress(0xF083, XDATA) ;

      /* Jump to the applet again to re-read IO ports */
      CIC_status = uiCIC_JumpAddress(0x00U, CIC61508_APPLET_CODE_BASE ,
                                    Jump_Address);
   }

}
```

Note: as this example destroys the

7.6 Applet Example 3

The standard secure SPI mode read and write functions do not allow the state of CIC61508 SYSDIS pins to be directly controlled. During early board testing, These pins can be very important as they often have a big influence on other hardware. This example along gives a way to control them.

The example is supplied in 2 forms. The first one is a simple demonstration of toggling each SYSDIS pin in sequence. The second one uses the same applet to allow individual SYSDIS pin control via an interface function.

The principle of operation is similar to Example 2 except that the XDATA array of port images is now written to by the secure mode write command and the array itself then written to the actual port pins by the applet. The writes to the port pins are triggered by the repeated calling of the applet.

```
void IO_SYSDISx_WriteTest(void)
{

   /* Declare Locals */
   uint8 Data;

   /* Update SYSDIS A & B on Port3 */
   Data  = (P3_DATA & SPC_P3_DATA_MASK);
   Data |= uiPx_Image[3];
   P3_DATA = Data & (uint8)~SPC_P3_DATA_MASK;

   /* Update SYSDIS C on Port0 */
   Data  = (P0_DATA & SPC_P0_DATA_MASK);
   Data |= uiPx_Image[0];
   P0_DATA = Data & (uint8)~SPC_P0_DATA_MASK;

}
```

7.6.1 Example 3, Part2 Tricore Driver

This is an example how applets are used in a real situation. A simple interface function "vSYSDIS_State()", allows the user to specify the value of each SYSDIS pin individually:

```
void vSYSDIS_State(boolean SYSDIS_A_state, boolean SYSDIS_B_state,
                   boolean SYSDIS_C_state)
{

   /* Declare locals */
   uint8 SYSDISAB_State ;

   /* Load a CIC61508 applet */
   CIC_status = uiLoad_CIC61508Applet(CIC61508_AppletCodeApp3, uiCIC61508_AppletSizeApp3,
CIC61508_APPLET_CODE_BASE);
   if(CIC_status != CIC_No_Error)
   {

      /* User can decide how to handle errors */
      while(1) { ; }

   }

   /* Combine SYSDISA and SYSDISB states into a single variable */
   SYSDISAB_State = (uint8)SYSDIS_A_state | ((uint8)SYSDIS_B_state * 2);

   /* Update the SYSDISx pins from the Port images in XDATA */
   CIC_status = uiCIC_WriteSecureAddress(SYSDISAB_State, Port3_Image, XDATA);
   CIC_status = uiCIC_WriteSecureAddress(((uint8)SYSDIS_C_state * 0x04U), Port0_Image,
XDATA);

   /* Jump to the applet again to write the SYSDISx pins */
   CIC_status = uiCIC_JumpAddress(0x00U, CIC61508_APPLET_CODE_BASE , Jump_Address);

}
```

It is called from the main() demonstration as per:

```
/* Set state of SYSDIS Pins */
vSYSDIS_State(0,0,0);   /* SYSDISA = 0, SYSDISB = 0, SYSDISC = 0 */

vSYSDIS_State(0,0,1);   /* SYSDISA = 0, SYSDISB = 0, SYSDISC = 1 */

vSYSDIS_State(0,1,0);   /* SYSDISA = 0, SYSDISB = 1, SYSDISC = 0 */
```

On each invocation, the applet App3 loaded by vExecute_App3() is re-called to write the new SYSDIS pin pattern to the CIC61508 pins.

8 CIC61508 Secure SPI Mode Driver Library

This library has been design to run on any Tricore variant and requires just a pre-initialised SSC port and GPTA compare register interrupt. The functions contained with it are listed below:

```
extern void vCIC_SPI_Driver(void) ;
extern void vInit_CIC_SPI_Driver(void) ;
extern uint16 uiCIC_SPI_SendMessage(uint16 uiTx_data) ;
extern CIC_ErrorType uiCIC_SPI_ReadSFR(uint8 uiSFR_addr, uint8 *uiSFR_data) ;
extern CIC_ErrorType uiCIC_SPI_WriteSFR(uint8 uSFR_addr, uint8 uiSFR_data) ;
extern CIC_ErrorType uiCIC_SPI_EnterSecureMode(void) ;
extern uint16 uiCIC_ReadSecureAddress(uint16 uiAddress, uint16 uiMspace) ;
extern CIC_ErrorType uiCIC_EraseDFLASH(void) ;
extern CIC_ErrorType uiCIC_WriteEntireDFLASH(uint8 *uiInputData, uint16 uiLength) ;
extern uint16 uiCIC_PollDFLASH_Busy(void) ;
extern  CIC_ErrorType  vVerify_DFLASH(uint8  *uiVerifyArray,  uint16  uiDFLASH_length,  uint16
uiDFLASH_base_addr) ;
extern CIC_ErrorType uiReset_CIC_legacy(void) ;
extern CIC_ErrorType vGet_CIC61508_SFRs(void) ;
extern void vCIC_Format_DFLASH(uint8 *uiDFLASH_ptr, uint16 uiDFLASH_length) ;
extern void CIC_wait(void) ;
extern void vDeInit_CIC_SPI_Driver(void) ;
extern void vRead_DFLASH_Tune_ID(int8 *iBuffer) ;
extern uint16 uiCIC_SPI_FastWriteSFR(uint8 uiSFR_addr, uint8 uiSFR_data) ;
extern CIC_ErrorType uiCIC_SPI_FastReadSFR(uint8 uiSFR_addr, uint8 *uiSFR_data) ;
extern CIC_ErrorType uiReset_CIC(void);
extern   CIC_ErrorType   uiCIC_WriteSecureAddress(uint8   uiData,   uint16   uiAddress,   uint16
uiMspace);
extern CIC_ErrorType uiReset_CIC_Immediate(void);
extern CIC_ErrorType uiLoad_CIC61508Applet(uint8 *uiAppletCode, uint16 uiAppletSize, uint16
uiAppletCodeBaseAddr);
extern CIC_ErrorType uiCIC_JumpAddress(uint8 uiData, uint16 uiAddress, uint16 uiMspace);
```

The function prototypes for the functions in the library are accessed via the CIC_SecureMode.h header file.

A full version of the SPI driver library is available under a specific licence for use in customers' own applications.

w w w . h i t e x . c o m

Published by Hitex (U.K.) Limited.

CIC61508, V0.9