
DSP STAR Code Builder User's Manual

Revision 4.2

Literature Number: NDT041201

December 2004



LIST OF CONTENTS

1.	CODE BUILDER.....	1-1
1.1.	General Description	1-1
1.2.	Installation	1-3
1.3.	Menu and Tool Bar Icon.....	1-4
1.4.	Project Management	1-4
1.4.1.	PROJECT SETTING	1-5
1.4.2.	PROJECT INITIALIZATION	1-7
1.5.	Program Build.....	1-8
1.6.	DspStar Menu	1-8
1.6.1.	COMMUNICATION SETUP AND CONNECT/DISCONNECT	1-9
1.6.2.	PROGRAM LOAD	1-9
1.6.3.	FLASH READ/WRITE.....	1-10
1.7.	Debug Tools	1-12
1.7.1.	RESET	1-12
1.7.2.	RUN	1-12
1.7.3.	MEMORY EDIT	1-12
1.7.4.	MEMORY COPY/FILL	1-12
1.7.5.	BREAKPOINT	1-13
1.7.6.	BREAK	1-13
1.7.7.	CONTINUE.....	1-13
1.8.	View related Tools.....	1-13
1.8.1.	MEMORY VIEW	1-14
1.8.2.	DISASSEMBLY VIEW	1-15
1.8.3.	SYMBOL VIEW	1-16
1.8.4.	CORE REGISTER	1-16
1.8.5.	PERIPHERAL REGISTER.....	1-17
1.8.6.	GRAPH TIME	1-18
1.8.7.	GRAPH DFT	1-19
1.8.8.	IMAGE VIEW	1-19
1.9.	Window.....	1-20
1.10.	Help.....	1-22
1.11.	Real Time Data exchange (RTDX)	1-22

LIST OF FIGURES

FIGURE 1.	WINDOW EXAMPLE OF THE CODE BUILDER	1-2
FIGURE 2.	THE CODE BUILDER MENU	1-4
FIGURE 3.	PROJECT MENU	1-5
FIGURE 4.	PROJECT SETTING DIALOG BOX	1-6
FIGURE 5.	INITIALIZATION DIALOG BOX	1-7
FIGURE 6.	BUILD MENU	1-8
FIGURE 7.	DSPSTAR MENU	1-8
FIGURE 8.	WHEN COMMUNICATION SETUP SUCCEEDS (IN CASE OF DSP6711)	1-9
FIGURE 9.	WHEN COMMUNICATION SETUP FAILS	1-9
FIGURE 10.	COMMUNICATION SETUP DIALOG BOX	1-9
FIGURE 11.	FLASH MEMORY READ/WRITE	1-11
FIGURE 12.	DEBUG MENU	1-12
FIGURE 13.	MEMORY VIEW WINDOW	1-14
FIGURE 14.	POP-UP MENU IN MEMORY VIEW WINDOW	1-15
FIGURE 15.	DISASSEMBLY VIEW	1-15
FIGURE 16.	SYMBOLS VIEW	1-16
FIGURE 17.	REGISTER WINDOWS	1-17
FIGURE 18.	GRAPH TIME VIEW	1-18
FIGURE 19.	POPUP MENU IN GRAPH VIEWS	1-18
FIGURE 20.	GRAPH DFT VIEW	1-19
FIGURE 21.	IMAGE VIEW	1-20
FIGURE 22.	WINDOW MENU	1-20
FIGURE 23.	OUTPUT WINDOW	1-22
FIGURE 24.	BLOCK STRUCTURE OF RTDX IN CODE BUILDER	1-23
FIGURE 25.	DECLARATIONS OF THE RTDX FUNCTIONS	1-23

LIST OF TABLES

TABLE 1.	CODE BUILDER ICONS.....	1-4
TABLE 2.	ASSEMBLER AND COMPILER OPTIONS.....	1-6

1. Code Builder

1.1. General Description

Code Builder is a Windows based application that allows the user to write DSP code, compile / link, debug, execute on DSP and control the hardware. The main features of the Code Builder are as follows:

- **Project Management and Editing**
Provides source code editing tools and project management tools.
- **Code Generation Environment**
Provides code generation tools such as assembling, C compiling and linking.
- **Debugging**
Provides debugging tools such as breakpoint, watch window, disassembly view, register view, symbol view and memory view.
- **Signal and Data Processing**
Provides graph plotting for signal and data processing.
- **Hardware Control**
Provides hardware control tools such as ROM Read/Write/Erase.

The Code Builder application is shown in Figure 1. Tools can be selected either from the pull down menu or from the various toolbars. The user's can manage various DSP applications using the "project" directory under which all the relevant files are arranged in a tree structure. Source code (assembly or C language files including header files) can be edited in the source window. The first two tabs in the output window show informative messages created during building or debugging and the third tab shows a watch window. The fourth tab opens a console screen, which displays the results of standard I/O

functions. Two register windows, core register window and peripheral register window show the register contents, while the memory window displays memory contents for a specified address range. In the memory view, memory contents can be edited by double clicking on the address. Memory values of the memory block are plotted in the graph view with specified plotting parameters. Also, memory values are taken to get a plot in frequency domain providing DFT results. Image Data in DSP memory can be shown in the screen through Image View Menu.

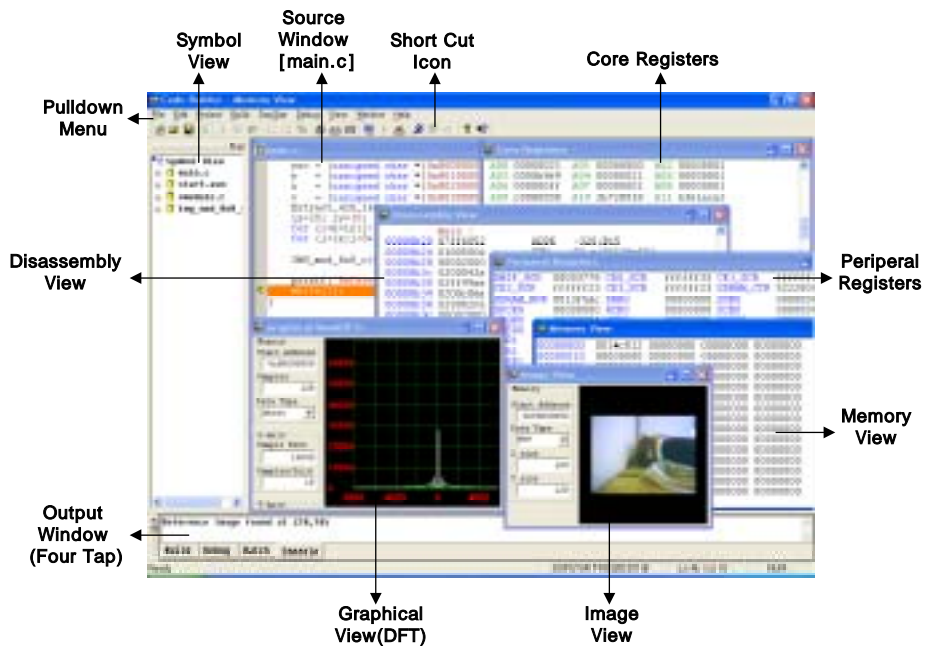
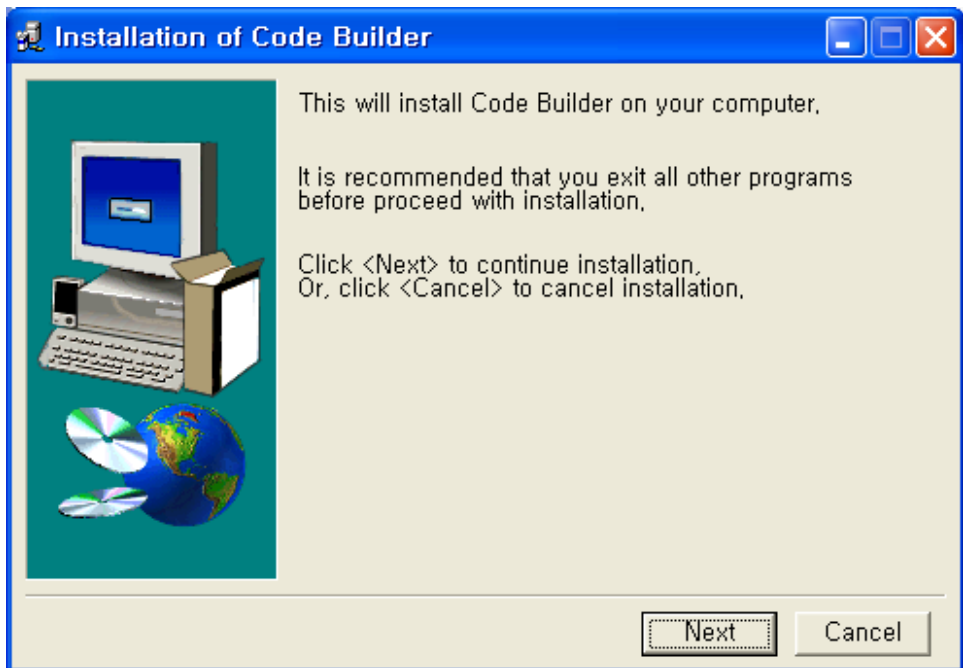


Figure 1. Window Example of the Code Builder

1.2. Installation

The Code Builder installation can be done in two ways. The first is to use Installshield, which follows the following procedures:

- (1) Run the install file on Code Builder CD ROM.
- (2) Follow the onscreen instructions. The first step is shown below:



- (3) If you experience problems then please call NDTech Technical support or your local distributor.

The second installation can also be performed by simply copying the installed directory as Code Builder does not need to set up environment registry variables.






















1.3. Menu and Tool Bar Icon

The Code Builder Menu is shown in Figure 2 along with short con icons used on the toolbars. A detailed description of the short con icons is listed in Table 1.



Figure 2. The Code Builder Menu

Table 1. Code Builder Icons

	New File	Create a new file		Find	Finds the specified text
	Open	Open an existing file		Compile	Compile
	Save	Save the document		Build	Incremental Build
	Print	Print an existing File		Rebuild	Build all
	Cut	Cuts the selection and moves it to the clipboard		Load	Loads program into DSP memory
	Copy	Copies the selection into the clipboard		Run	Program Execute
	Paste	Inserts the clipboard contents at the insertion point		Reset	Reset DSP
	Undo	Undoes the last action		Break	Break
	Redo	Redoes the previously undone action		Continue	Continue
	Break point	Set Break point		Help Topics	Help Topics
	About Code builder	About Code builder			

1.4. Project Management

In order to develop DSP application program, the tools in "Project" menu as shown in Figure 3 can be used.

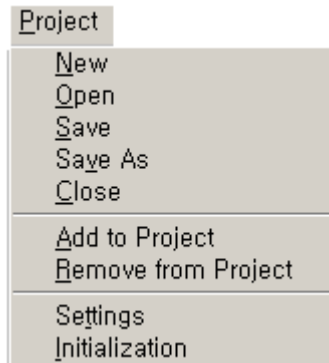


Figure 3. Project Menu

Use the *Add to Project* to add the files to the current project. The file is deleted from the current project by using the *Remove from Project* menu or the delete key.

1.4.1. Project Setting

Select *Project->Project Setting* to specify the build options. The project setting window is displayed in Figure 4. You can specify options for assembler, linker, or compiler. The directory for compiler, linker and library must be specified. A list of the build options is shown in Figure 4. The compiler option *-g* is needed to get Symbol View in Figure 16 at program load.

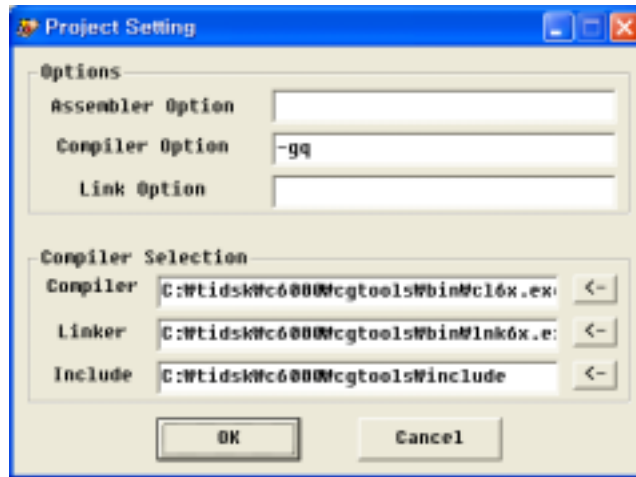


Figure 4. Project Setting Dialog Box

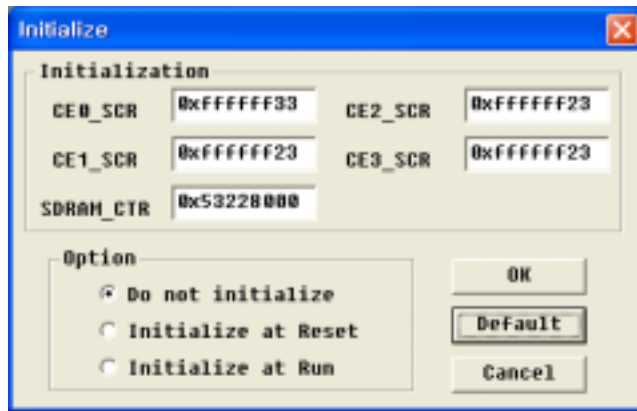
Table 2. Assembler and Compiler Options

Assembler Options	
-adNAME	Pre-defined NAME
-ahc<f>	.copy file f
-ahi<f>	.include file f
-al	Generates an assembly listing file
-as	Puts labels in the symbol table
-auNAME	undefined NAME
-ax	Generates the cross-reference file
Compiler Options	
-@<fn>	Appends the contents of a file to the command line.
-b	Disables merge of symbolic debugging information.
-c	Autoinitializes variables at run-time
-dNAME	Predefines the constant <i>name</i> for the preprocessor.
-g	Creates debugging information
-i<dir>	Defines library search path
-n	Compiles or assembly optimizes only.
-q	Quiet
-k	Retains the assembly language output from the compiler or assembly optimizer.
Optimizations	
-o0	Optimizes register usage
-o1	Uses -o0 optimizations and optimizes locally
-o2	Uses -o1 optimizations and optimizes globally
-o3	Uses -o2 optimizations and optimizes the file
-os	Interlists optimizer comments with assembly statements
Linker Options	
-a	Generates absolute executable output
-i<dir>	Defines library search path
-l<dir>	Supplies library or command filename
-o<file>	Names the output file

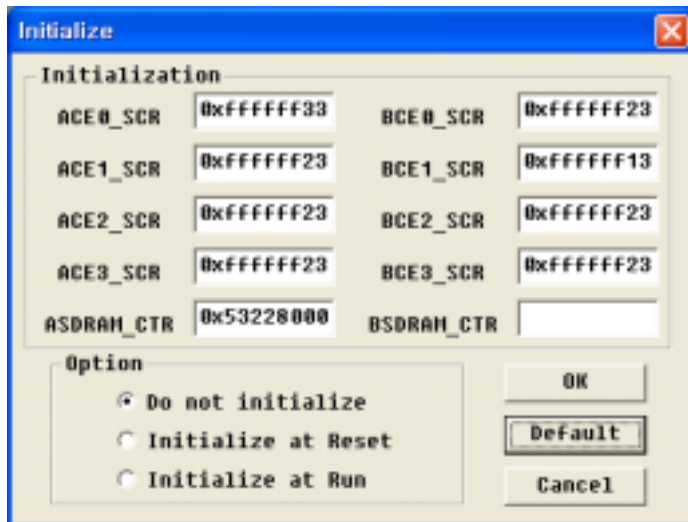
-q	Suppresses progress messages (quiet)
-s	Strips symbol table information and line number entries from the output module
-heap 0x<size>	Sets heap size (bytes)
-stack 0x<size>	Sets stack size (bytes)

1.4.2. Project Initialization

Use the *Initialization* menu to specify memory initialization options as in Figure 5. The *Default* button displays default EMIF setting values.



(a) C6000 DSP (except C6416)



(b) 6416 DSP

Figure 5. Initialization Dialog Box

1.5. Program Build

The Build menu of the Code Builder contains three sub menus as shown in Figure 6

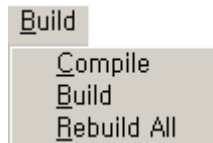





Figure 6. Build Menu

Select *Project->Build* or click  in the toolbar to create an executable file. The output (coff) file is created in the directory where the current project is located. The output file name is specified in the linker command file.

Select *Project->Compile* or press  to check a file for syntax error. Select *Project->Build* for incremental building and select *Project->Rebuild All* or press  for building all the related files.

1.6. DspStar Menu

The *DspStar* menu shown in Figure 7 provides the board related tools such as Communication Setup, Connect/Disconnect, Program Load, Flash Read and Flash Write. The particular DSP can be selected through the *Chip Select* menu.

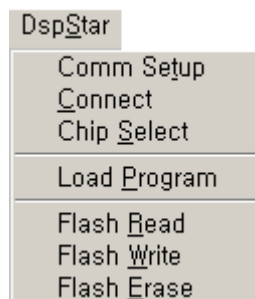


Figure 7. DspStar Menu

1.6.1. Communication Setup and Connect/Disconnect

When the Code Builder is started, automatic communication setup is performed. When communication setup is complete, the status bar appears as in Figure 8. When it fails, the status bar in Figure 9 indicates the failure. In case of failure, check the board power and then select *DspStar->Comm Setup*. In Figure 10, you can specify a port and a baud rate. After pressing *OK* button, select *DspStar->Connect*. *Connect* and *Disconnect* menu are interchangeably toggled according to as-of-the-time situation. The status bar in the Code Builder window shows the connection result.



Figure 8. When communication setup succeeds (In case of DSP6711)




Figure 9. When communication setup fails



Figure 10. Communication Setup Dialog Box

1.6.2. Program Load

The Program Load menu is to load an executable output file obtained from the build process into the DSP STAR memory. Select *DspStar->Program Load* or press  to open program load window. Upon completion, the selected output file is now loaded in the DSP program memory, ready to be executed.

1.6.3. Flash Read/Write

Flash memory can be used to hold boot codes. Code Builder provides the tools to allow the user to execute the code stored in the flash memory and change the user's own boot code. Flash memory is divided into 16 pages ranging from page 0 to page 15.

Use *DspStar->Flash Memory->Read* to read the flash memory or *DspStar->Flash Memory->Write* to write a code to the flash memory, as shown in Figure 11. Then, the address setting window in Figure 11 appears. Once the Flash code is brought to DSP memory, it can be executed.

Flash Memory Write is to write the boot code or the application program into Flash memory area. Use *Program Load* to load a boot code from the host PC and select page number like *Page 0* in the page setting window and press *OK* button. You can use all 16 pages. For other page number, follow the same procedure.

What follows is a set of instructions to develop a DSP program and burn it into Flash memory. For programs larger than 1K bytes in size, the file "start.asm" as in Romboot directory of example programs can be used. The following is a list of steps to burn code into Flash Rom.

1. Make sure that the reset vector in the file "start.asm" should be open for flash boot as follows:

```
;reset:          B.S2    _c_int00  ; open for load & run
reset:          B.S2    boot      ; open for flash boot
```

2. Build the project.
3. Load the out file into DSP memory using *DspStar->Program Load* Menu.
4. Use *DspStar->Flash Write* to burn the codes into FLASH memory. **It is very important not to execute the program before Flash Write.**

5. Turn off the DSP board power and then turn on the power. Then the program works as programmed.

Flash Memory Read loads an application program stored in flash memory area into the DSP program memory area. As in Flash Memory Write, choose a page number and type in the address and the length and then press *OK* button. Then the program code from the selected page is loaded into the specified address of DSP program memory. With the *DspStar->Flash Erase* menu, the data in Flash ROM can be erased, resulting in 0xffff value.



Figure 11. Flash Memory Read/Write

1.7. Debug Tools

Figure 12 shows *Debug* Menu.

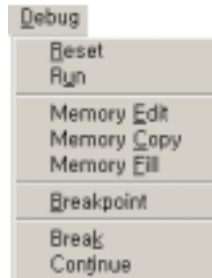




Figure 12. Debug Menu

1.7.1. Reset

Reset is to reset the DSP processor. It is the same as pressing the reset icon () in the toolbar.

1.7.2. Run

Run is to execute the code loaded in the DSP program memory. It is the same as pressing the execute icon () in the toolbar.

1.7.3. Memory Edit

Select *Debug->Memory Edit* to change the content of a memory. One can write an memory address in decimal or in hexadecimal or in symbol in the address box.



1.7.4. Memory Copy/Fill

Memory Copy is to copy from one memory area to another memory area. Select *Debug->Memory->Copy* to initiate memory copy. Fill the start address (decimal or hexadecimal or


symbol), the memory length in words and the destination address.

Memory Fill is to fill memory area with a value. Fill the start address, the memory length in words and the memory value. Click *OK* button to fill the memory block with the given data value.


1.7.5. Breakpoint

Code Builder allows the user to set breakpoints in C source file. The user can set breakpoints wherever he/she would like. Move to the line where you would like to set a breakpoint, then select *Debug->Breakpoint* or double click the mouse left button or click the breakpoint icon . The breakpoint can also be released by selecting *Debug->Breakpoint* or double-clicking the mouse left button or clicking the breakpoint icon. The stopped execution can resume with *Continue*, described in section 1.7.7. Its short cut icon is .

1.7.6. Break

Break is to break current execution temporarily. The halted execution can resume with *Continue*, described in section 1.7.7. Its short cut icon is .

1.7.7. Continue

Continue is to restart the execution. It is the same as pressing the continue icon ()

1.8. View related Tools

This section describes the tools provided to allow the user to view memory contents in data format, in graphs, or in disassembly format, symbols created as global/local variables

in the program source and further DSP registers.

1.8.1. Memory View

When *View->Memory* is selected, the memory contents in the memory window are displayed as in Figure 13. The hexadecimal numbers displayed in blue are the memory address, while the numbers displayed in red or black are the memory content values. Memory values displayed in red are the ones changed, while those in black are the ones unchanged from previous display.

Click the mouse right button to get the pop-up menu as in Figure 14. Once the memory contents appear, use the arrow keys (↑,↓) or page keys (page up, page down) or mouse wheel to move beyond or below the memory area in the display.

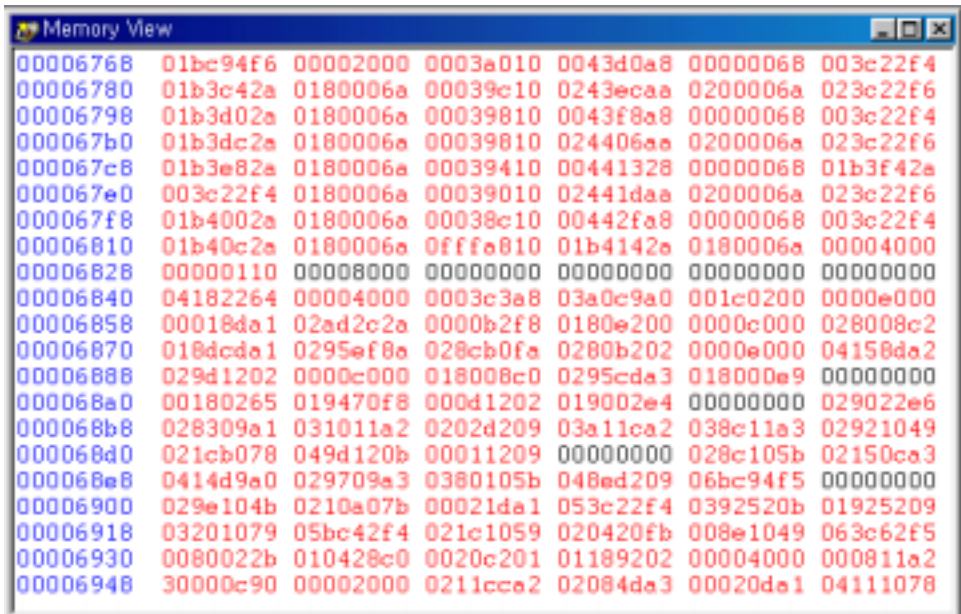


Figure 13. Memory View Window

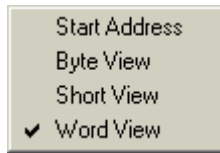


Figure 14. Pop-up menu in Memory View Window

Memory values can be edited by double clicking on the specific addresses of the memory window.

1.8.2. Disassembly View

Disassembly View is to show the memory contents in an assembly language format as in Figure 15. Type an address in decimal or in hexadecimal or in symbol in the edit box. Arrow keys (↑,↓) move the display up or down by one memory word. But, page keys (page up and page down) moves the display by memory block.

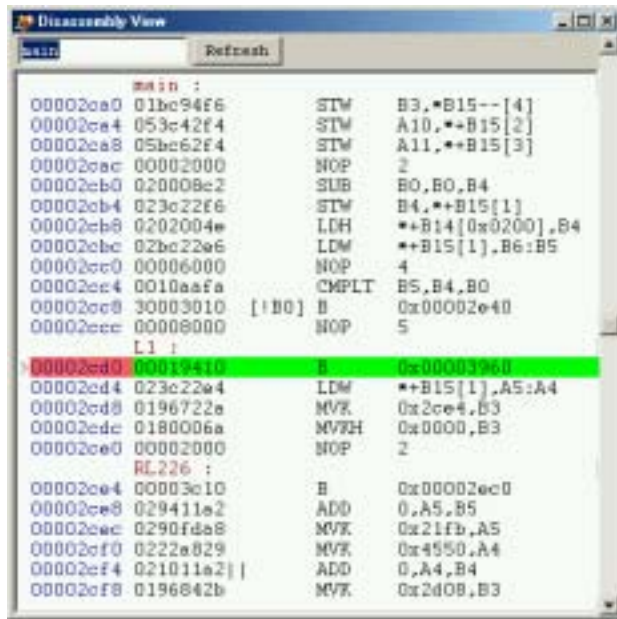


Figure 15. Disassembly View

In the above figure, the red and green stripes are the results from breaking and setting a breakpoint, respectively. It is possible to set a breakpoint in the C source window rather than the disassembly window, but when new breakpoint is set in the C file, the green stripe appears both in C source window and in Disassembly View window. Similarly, when program execution breaks, the red stripe appears both in the C source window and in the Disassembly window.

1.8.3. Symbol View

When an executable file is loaded, a symbol view automatically appears as in Figure 16. The symbol view displays the source files, the functions and the global/local variables in tree structure. You can open the source edit window by double clicking a mouse left button on the source file, and add variables to the watch window also by double clicking the mouse left button on the variable name. The symbol view can be used to add the variable (local or global) to the watch window.

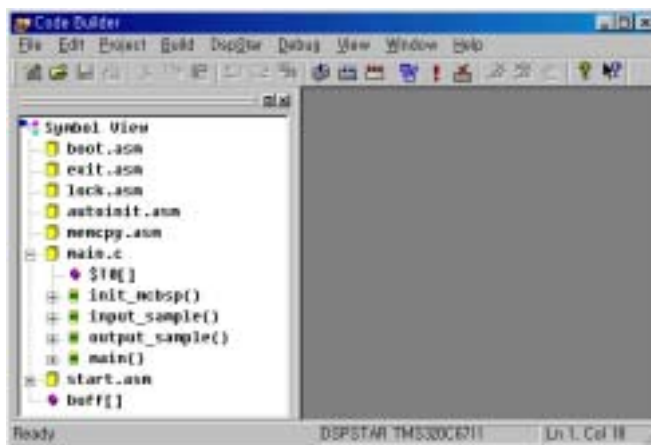


Figure 16. Symbols View

1.8.4. Core Register

The Core Register View displays the contents of DSP core registers. In the display

window (Figure 17), double-click the mouse left button on the specific register, then Register Edit Dialog Box in Figure 17 appears. Select the register and type the value and press *OK* button to change the register value.

1.8.5. Peripheral Register

The Peripheral Register displays the contents of DSP peripheral registers. In the display window (Figure 17), double-click the mouse left button on the specific register, then Register Edit Dialog Box shown in Figure 17 appears. Select a register and type a value and press *OK* button to change the register value. Be cautious when changing the register value. It may result in halting the DSP processor.

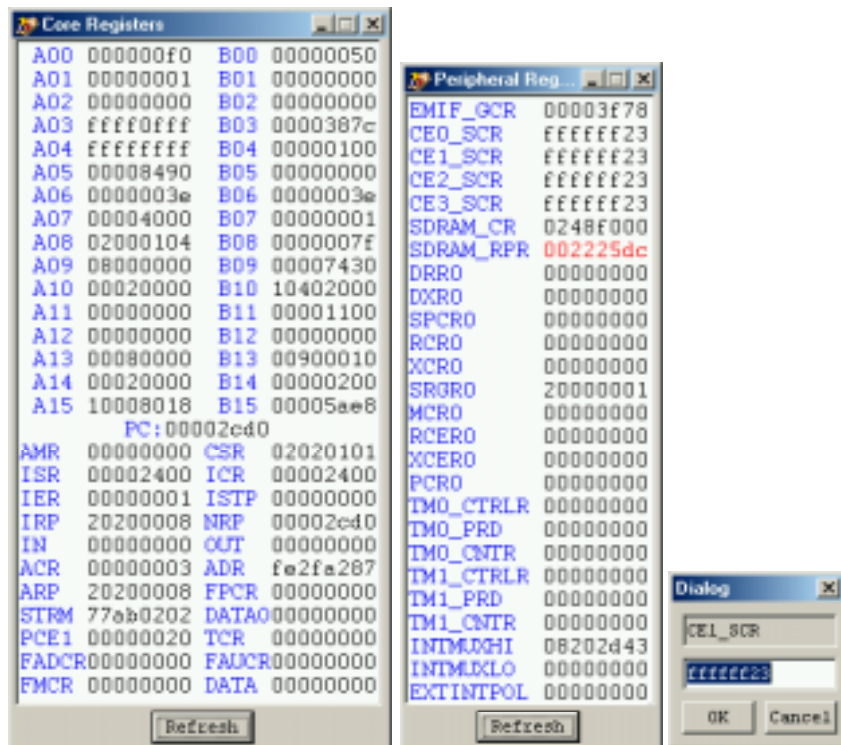


Figure 17. Register Windows

1.8.6. Graph Time

When selecting *View->Graph Time*, the graph dialog box in Figure 18 appears. In the graph plot, *Samples* numbers of data values from *Start* address (in decimal, in hexadecimal, or in symbol) are taken in the plot. Reading memories depends on *Data Type* specified in the graph dialog box. *Samples/Grid* represents the number of samples plotted in one horizontal grid.

Value/Grid represents y value corresponding to one vertical grid. When the mouse right button is clicked, a pop-up menu appears as in Figure 19. *Auto Refresh* keeps updating graph display.

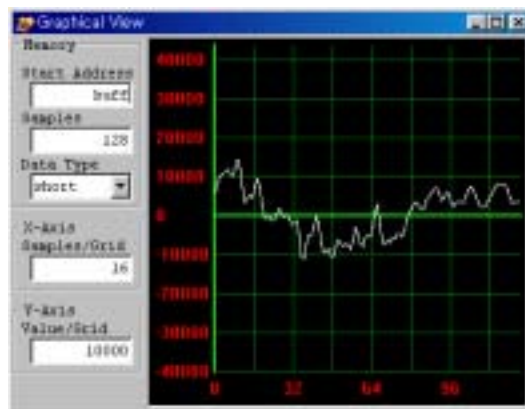


Figure 18. Graph Time View

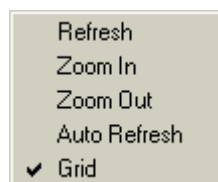


Figure 19. Popup Menu in Graph Views

1.8.7. Graph DFT

Graph DFT plots data points in frequency domain (in terms of DFT) as in Figure 20. In the graph plot, *Samples* numbers of data values from *Start* address (in decimal, in hexadecimal, or in symbol) are taken in the plot. Reading memories depends on *Data Type* specified in the graph dialog box. *Samples/Grid* represents the number of samples plotted in one horizontal grid, and *Sample Rate* represents a sampling rate of the data. One can select a *Plot Type* among Magnitude, Phase, Imaginary, and Real.

Similar to *Graph Time*, when the right mouse button is clicked, a pop-up menu appears as in Figure 19. *Auto Refresh* keeps updating graph display.

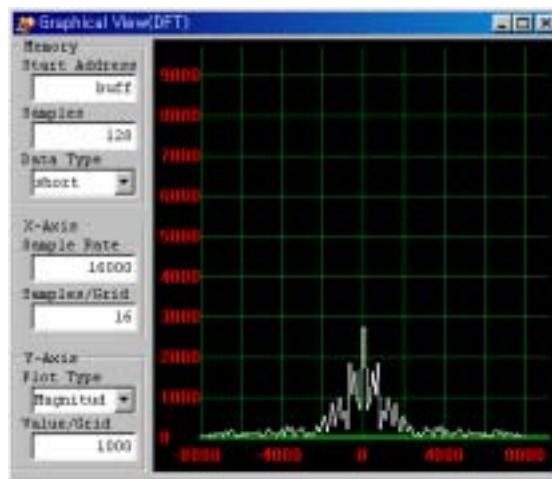


Figure 20. Graph DFT View

1.8.8. Image View

The View->Image View menu can be used to get image window as in Figure 21. The data from *Start Address* is taken to get image display in right image window. The Data Type combo box provides several image format such as YUV, RGB, B/W and X and Y size represent horizontal and vertical image sizes (pixel numbers). After entering four

parameter values, hit the enter key to start reading in memory. During this process, progressive bar located in lower left part of the Code Builder window display reading process.

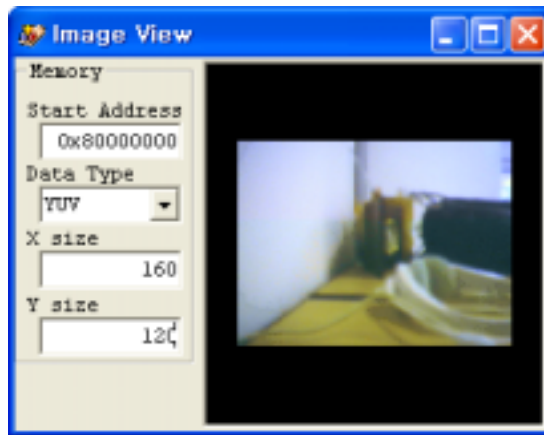


Figure 21. Image View

1.9. Window

The Window menu in the Code Builder looks as in Figure 22. By selecting *Window->Project*, *Window->Symbol*, or *Window->Output*, the project, the symbol, or the output window respectively is displayed.

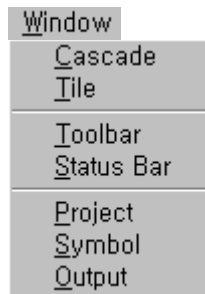
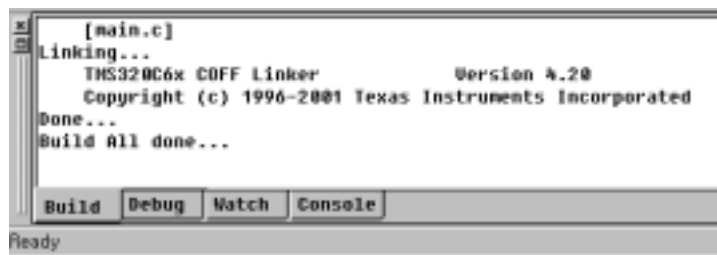


Figure 22. Window Menu

The output window in Figure 23 has four taps: build tap, debug tap, watch tap and console

tap. The build tap shows build-related messages and the debug tap shows debug-related messages, while the watch tap is a watch window of symbols, which can be added from the Symbol Tree by double clicking the mouse left button, or deleted using the delete key. When the mouse left button is double-clicked on the specific watch symbol name (or the other symbol area), then Memory View Window (Memory Edit Window) appears. When the program includes console I/O functions such as printf, getchar, and scanf, the console tab is used as input and output windows.



(a) Build Tab



(b) Console Tap



(c) Debug Tap

Name	Address	C	Value(Hex)	Type	Value(Type)
first	0x00008654	G	0x031be2a2	short []	
second	0x0000865c	G	0x0003fb08	short []	
i	0x00000014	L	0x00000000	int	0

(d) Watch Tap

Figure 23. Output Window

1.10. Help

The Code Builder Help provides assistance in using Code Builder and developing with the DSP Star products.

1.11. Real Time Data exchange (RTDX)

This section describes a real time data exchange between DSP hardware and host PC computer. RTDX permits developers to exchange data in real time between a host and a target DSP without stopping their DSP applications. Data exchange speed can be upto 1Mbps, but is now 115200 bps due to current limitations on RS232 implementation. As shown in Figure 24, there is no software resident in DSP target hardware. The PC RTDX functions communicate to DSP through RS serial connection. The serial standard RS232 is implemented using serial HPI. In host PC, the developers can develop their MFC application using the RTDX functions of the Code Builder.

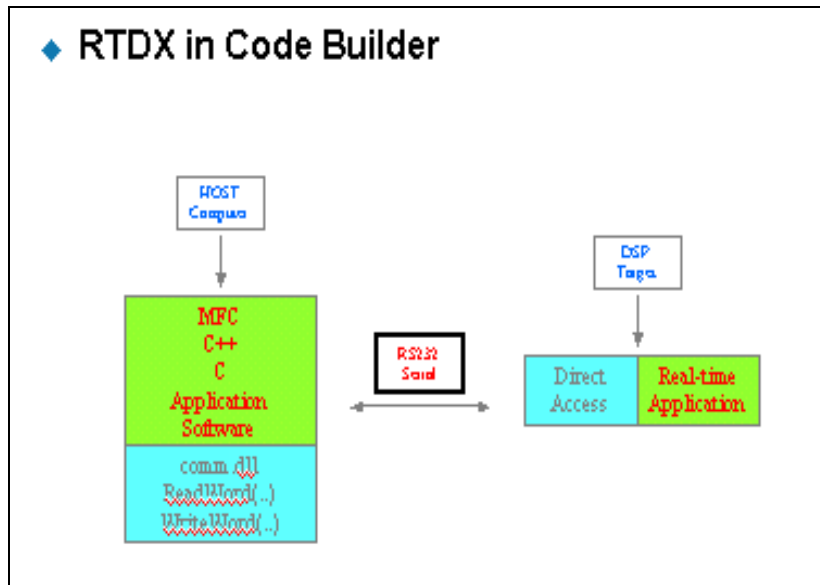


Figure 24. Block Structure of RTDX in Code Builder

The Code Builder provides three files related to RTDX: comm.dll, comm.lib, and comm.h. Include the provided header file “comm.h” in the MFC program and use comm.lib for building process and comm.dll for executing. Figure 25 lists the RTDX functions declared in the comm.h file, which can be used for the development of the user's PC application program.

```
extern "C" _declspec(dllexport) BOOL    GetConnected(void);
extern "C" _declspec(dllexport) int    OpenPort(CString sPortName, DWORD dwBaud);
extern "C" _declspec(dllexport) void   ClosePort();
extern "C" _declspec(dllexport) void   Reset();
extern "C" _declspec(dllexport) int    ReadBuffer(DWORD add, DWORD *data, DWORD m);
extern "C" _declspec(dllexport) int    WriteBuffer(DWORD add, DWORD *data, DWORD m);
extern "C" _declspec(dllexport) DWORD  ReadWord(DWORD add);
extern "C" _declspec(dllexport) BOOL   WriteWord(DWORD add, DWORD data);
extern "C" _declspec(dllexport) int    ConnectBD(CString port, DWORD baud);
```

Figure 25. Declarations of the RTDX functions