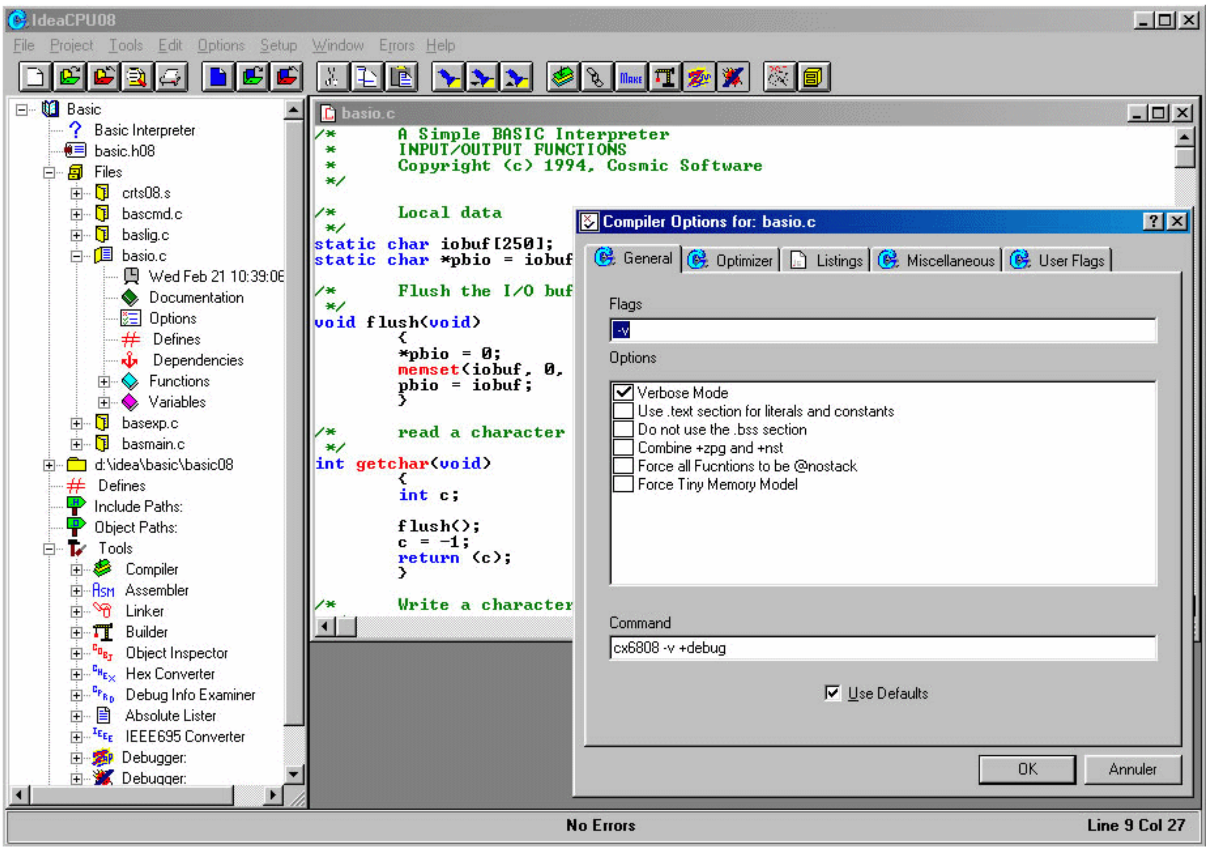
Cosmic Software C Compilers



```
/*      A Simple BASIC Interpreter
 *      INPUT/OUTPUT FUNCTIONS
 *      Copyright (c) 1994, Cosmic Software
 */

/*      Local data
 */
static char iobuf[250];
static char *pbio = iobuf

/*      Flush the I/O buf
 */
void flush(void)
        {
        *pbio = 0;
        memset(iobuf, 0,
        pbio = iobuf;
        }

/*      read a character
 */
int getchar(void)
        {
        int c;

        flush();
        c = -1;
        return (c);
        }

/*      Write a character
```

Compiler Options for: basio.c

Flags: -v

Options

☑ Verbose Mode
☐ Use .text section for literals and constants
☐ Do not use the .bss section
☐ Combine +zpg and +nst
☐ Force all Fucntions to be @nostack
☐ Force Tiny Memory Model

Command: cx6808 -v +debug

☑ Use Defaults

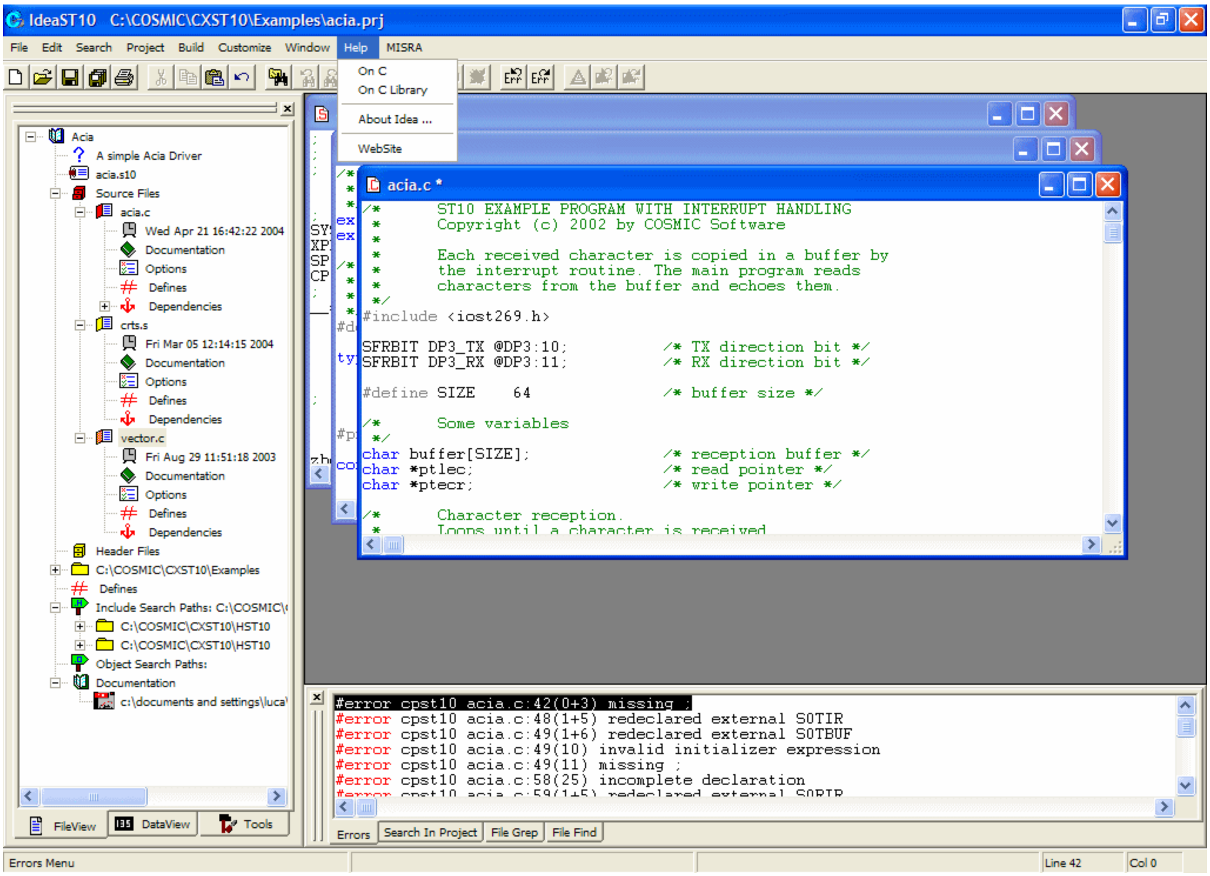
No Errors   Line 9 Col 27

Cosmic's C cross compilers are part of a complete and uniform compiler product line incorporating over 20 years of innovative design and development. Cosmic's familiar and easy to use software interface is common through all supported microcontrollers to simplify the learning curve and the migration between targets. All C compilers include:

- IDEA
  Cosmic's own integrated development environment for Windows is preconfigured for the compiler it comes with. (IDEA is not available on UNIX)
- ANSI and ISO C Compiler
  Cosmic compilers follow ANSI and ISO rules and conventions. C Syntax extensions provide efficent use of processor specific features such as zero page, peripherals, eeprom and paged memory.
- Reentrant and Recursive
  Most Cosmic Compilers are fully reentrant and recursive using standard ANSI stack frame conventions.
- Library source code
  Cosmic Compilers come with source code for all libraries provided.
- In-line Assembly
  The compilers support three convenient methods for adding assembly code inside a C program including an argument passing mechanism.
- Absolute Listings
  Optionally produce relocatable and/or absolute C interspersed with the corresponding Assembly listings.
- Windows, UNIX and Linux
  Cosmic compilers are available on PCs running Windows 2000/XP and UNIX systems running PC-Linux, SUN Solaris and HP-UX.
- Host Independent Formats
  The Cosmic relocatable and absolute object formats are host independent. This allows user's on PC Windows, Linux, SUN and HP to share objects for linking and debugging.

- IEEE-695 and ELF/DWARF
  The Cosmic compilers support the IEEE-695 and ELF/DWARF 2.0 standard debug formats used by many popular emulators and logic analyzers.
- C Interrupt functions and vector table
  Interrupts can be managed entirely in C. The compiler takes care to save and restore the necessary registers.
- Macro assembler
  Supports C #defines and #includes so that C and assembly language modules may share common constants and macros.
- Linker
  Place the code in memory and take care of initializing global variables. Can also generate automated checksums for most targets.
- Utilities
  Librarian, hex file generator, object format converters, debugging support utilities, multi-pass compiler command driver.

Integrated Development Environment



```
/*      ST10 EXAMPLE PROGRAM WITH INTERRUPT HANDLING
 *      Copyright (c) 2002 by COSMIC Software
 *
 *      Each received character is copied in a buffer by
 *      the interrupt routine. The main program reads
 *      characters from the buffer and echoes them.
 */
#include <iost269.h>

SFRBIT DP3_TX @DP3:10;          /* TX direction bit */
SFRBIT DP3_RX @DP3:11;          /* RX direction bit */

#define SIZE    64              /* buffer size */

/*      Some variables
 */
char buffer[SIZE];              /* reception buffer */
char *ptlec;                    /* read pointer */
char *ptecr;                    /* write pointer */

/*      Character reception.
 *      Loops until a character is received
```

```
#error cpst10 acia.c:42(0+3) missing ;
#error cpst10 acia.c:48(1+5) redeclared external S0TIR
#error cpst10 acia.c:49(1+6) redeclared external S0TBUF
#error cpst10 acia.c:49(10) invalid initializer expression
#error cpst10 acia.c:49(11) missing ;
#error cpst10 acia.c:58(25) incomplete declaration
#error cpst10 acia.c:59(1+5) redeclared external S0RIR
```

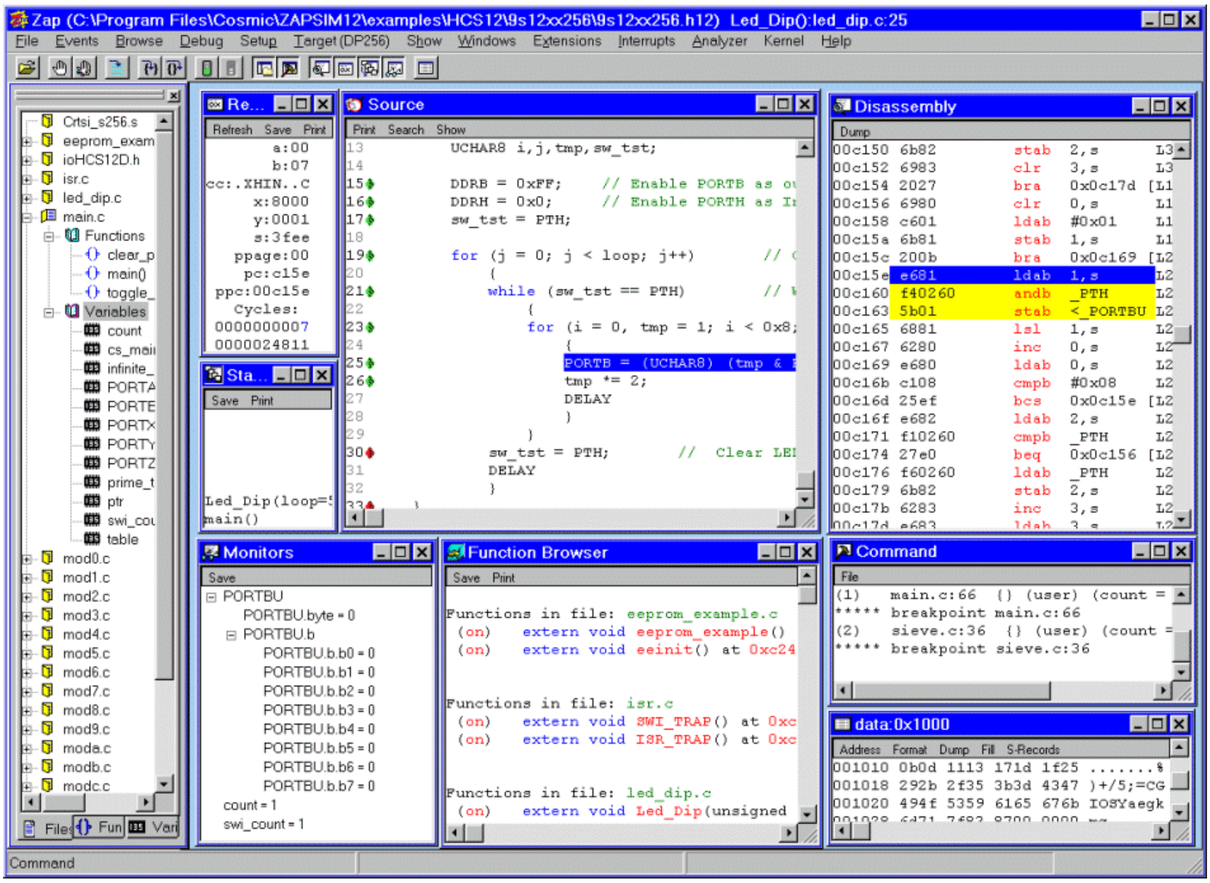
All Cosmic C Cross Compilers for Windows include IDEA - Cosmic's own integrated development environment which is preconfigured for the the compiler it comes with. IDEA is designed specifically for developing embedded applications with Cosmic compilers. IDEA integrates an editor, project manager, graphical smart build/make facility, program analyzer, link file generator, documentation manager and ZAP debugger into one easy to use environment running under Windows 2000, XP and Vista. IDEA includes the following features:

- Integrated Windows Editor
  Windows 32 bit MS style editor with syntax highlights for both C and assembly source.
- Project Manager
  Convenient Project window provides easy access to the most frequently used functions

such as make, build, touch, mark, compile, link and debug. The project manager also supports Drag and Drop so you can select and drag files from a Windows Explorer to the project manager for easy setup.

- Program Analyzer
  Intuitive Explorer style source file display - shows compiler options, include file dependencies, file build status, time and date of last edit, function prototypes, command line defines, global and static variables and documentation.
- Link File Generator
  Flexible Link builder offers point and click configuration of memory map.
- Graphical Smart Build
  Configure Compiler and Linker to build only files that need to be rebuilt and run various compiler or user defined utilities automatically. No need for makefile.
- Point and Click Options
  IDEA provides an intuitive graphical setup for compiler and assembler options.
- Automatic Error Handling
  When an error is found in a file just double click on the error and an editor window will open on that file, at the line with the error.
- Global and local options
  IDEA allows to set compiler and assembler options both at project and file level.
- Multifile functions
  Search in the project files or other group of files or directories.
- Documentation Manager
  Attach any documents or notes to your project or to individual source files.

ZAP Debugger



```
     a:00
     b:07
cc:..XHIN..C
   x:8000
   y:0001
   s:3fee
 ppage:00
 pc:c15e
ppc:00c15e
 Cycles:
0000000007
0000024811
```

```
Led_Dip(loop=5
main()
```

```
        UCHAR8 i,j,tmp,sw_tst;

        DDRB = 0xFF;     // Enable PORTB as ou
        DDRH = 0x0;      // Enable PORTH as In
        sw_tst = PTH;

        for (j = 0; j < loop; j++)        // C
            {
            while (sw_tst == PTH)         // W
                {
                for (i = 0, tmp = 1; i < 0x8;
                    {
                    PORTB = (UCHAR8) (tmp & P
                    tmp *= 2;
                    DELAY
                    }
                }
            sw_tst = PTH;          //  Clear LED
            DELAY
            }
    }
```

```
00c150 6b82      stab  2,s       L3
00c152 6983      clr   3,s       L3
00c154 2027      bra   0x0c17d  [L1
00c156 6980      clr   0,s       L1
00c158 c601      ldab  #0x01     L1
00c15a 6b81      stab  1,s       L1
00c15c 200b      bra   0x0c169  [L2
00c15e e681      ldab  1,s       L2
00c160 f40260    andb  _PTH      L2
00c163 5b01      stab  <_PORTBU  L2
00c165 6881      lsl   1,s       L2
00c167 6280      inc   0,s       L2
00c169 e680      ldab  0,s       L2
00c16b c108      cmpb  #0x08     L2
00c16d 25ef      bcs   0x0c15e  [L2
00c16f e682      ldab  2,s       L2
00c171 f10260    cmpb  _PTH      L2
00c174 27e0      beq   0x0c156  [L2
00c176 f60260    ldab  _PTH      L2
00c179 6b82      stab  2,s       L2
00c17b 6283      inc   3,s       L2
00c17d e683      ldab  3,s       L2
```

```
PORTBU
    PORTBU.byte = 0
    PORTBU.b
        PORTBU.b.b0 = 0
        PORTBU.b.b1 = 0
        PORTBU.b.b2 = 0
        PORTBU.b.b3 = 0
        PORTBU.b.b4 = 0
        PORTBU.b.b5 = 0
        PORTBU.b.b6 = 0
        PORTBU.b.b7 = 0
count = 1
swi_count = 1
```

```
Functions in file: eeprom_example.c
 (on)   extern void eeprom_example()
 (on)   extern void eeinit() at 0xc24

Functions in file: isr.c
 (on)   extern void SWI_TRAP() at 0xc
 (on)   extern void ISR_TRAP() at 0xc

Functions in file: led_dip.c
 (on)   extern void Led_Dip(unsigned
```

```
(1)    main.c:66  {} (user) (count =
***** breakpoint main.c:66
(2)    sieve.c:36  {} (user) (count =
***** breakpoint sieve.c:36
```

```
001010 0b0d 1113 171d 1f25 .......%
001018 292b 2f35 3b3d 4347 )+/5;=CG
001020 494f 5359 6165 676b IOSYaegk
001028 6d71 7f83 8700 0000 mq
```

Cosmic's ZAP debugger is a full featured C and Assembly language source-level debugger for embedded applications. ZAP's intuitive graphical interface is uniform for all targets and execution environments. ZAP typical features include:

- ANSI C Debugging
  Provides easy access to any C object including Enums, Bit Fields, Structs, Floats, Strings etc.
- Assembly Source Debugging
  Debug mixed C and assembly applications at the C or Assembly source level including coordinated source and disassembly displays.
- Nonintrusive "Optimizer On" Debugging
  ZAP does not modify or augment the user code in any way. The code used by ZAP is the same optimized code that will be used in the final product. All debug symbols are stored in a separate section on the host (e.g. PC). Debug symbols are never stored on the target.
- Automated Testing
  In addition to the graphical interface, ZAP offers a robust command and scripting language which can be used to create automated test scripts including:
  Record and Playback - Save a debugging session and play it back later
  Multiple File Simulated I/O - Interactively open, read and write to multiple input and output files on the host system.
- Source Browsing
  Browse and set breakpoints in any source windows.
- On-line Help
  Includes Using ZAP, C Language Syntax and C library Functions to provide a complete debugging environment on the host.
- OS Support plug-in
  Some versions of ZAP are Kernel-aware via a plugin provided by Cosmic (example: OSEK for the HC12)

ZAP Simulator

Cosmic provides a simulator for virtually all the architectures supported. ZAP simulator features include:

- MCU Cycle Counter
  ZAP accurately counts MCU cycles to provide valuable timing information.
- Interrupt Simulation
  ZAP provides a configurable mechanism to simulate MCU interrupts.
- Graphical Performance Analysis
  Displays code coverage and timing information on a file by file or function by function basis.
- Code Coverage
  Generate reports for code executed or not executed.
- Chronographs
  Displays a time-line of function calls to track program execution.

ZAP for emulators and boards

ZAP is available for the most common development boards and emulators for every architecture. Check the microcontroller-specific pages for more information.