Workshop
V850 Software Development Tools

Green Hills MULTI for V850

Renesas Electronics Corporation (China)
Automotive Product Center

2011年11月3日星期四 Rev. 1.00

Workshop Agenda

Workshop Agenda (1/3)

- Introduction
- GHS MULTI IDE and project mangagement
  - MULTI IDE concept
  - GHS MULTI project manager
  - How to setup a project with the GHS MULTI project manager
  - How to set project options
- GHS C / C++ compiler
  - CPU register usage
  - Calling conventions
  - Memory models
  - Macros, Pragma directives and Intrinsic functions
  - Compiler settings
    - Recommended settings
    - Useful settings
  - MISRA-C checker
  - GHS Startup file and runtime initilization

Workshop Agenda (2/3)

- GHS Linker EXLR
  - GHS linker directive file (.ld)
  - Standard memory sections
  - Map file generation and how to read the map file
- GHS Librarian AX
  - Creating and modifying Libraries
  - Using Pre-Built Libraries
- GHS MULTI Debugger / 850eserv2 target server
  - MULTI Debugger, 850eserv2 target server, EXEC and Device-File - How does it fit together?
  - 850eserv2, Simulator and other target servers
  - Supported emulators and On-Chip debuggers
  - Integration of the new Renesas E1/E20 On-Chip Debugger
  - Connection manager and connection methods
  - Scripting

Workshop Agenda (3/3)

- User Manuals and further reading
  - Where to find the required information?

Introduction

V850 Software Development Tools Lineup

Green Hills SOFTWARE

CubeSuite+

IAR SYSTEMS

RENESAS V850

eclipse

Applilet®

Application
Leading
Tool

V850 3rd-Party Software Development Tools

- **Green Hills MULTI for V850**
  current version: **V5.1.7D**
  Very powerful toolchain, standard toolchain at all major european V850 automotive customers

- **IAR Embedded Workbench for V850**
  current version: **V3.80**
  Code-size limited kickstart version and unlimited 30-day eval. version available at
  http://www.iar.com/downloads

- **Renesas Eclipse for V850**
  under development, release planned for 01/2012
  free of charge GNU Toolchain
  Includes GNU C-compiler and GDB debugger
  Build and debug plugins for GHS and IAR will be included in the final release. Further information:
  http://www.renesaseclipse.com

Green Hills Multi® C/C++ Compiler for V850

- Integrated development environment with project management tools and editor
- Highly optimizing C/C++ compiler for V850/ES/E1/E2 cores
- Debugging system support
  - Green hills Software Simulator
  - Renesas E1 integration **(NEW Feature)**
  - Renesas V850MINIL/MINICUBE2 integration
  - Renesas IECUBE/IECUBE2 integration
  - TimeMachine Debugging Suite (Optional Add-On)
- MISRA C checker
- Eclipse build phase plug-in
- Supports Elf/Dwarf debug format
- Different license options available
  - Node-Locked, Dongle, Floating
- Current version: V5.17D*
- Order Code: CPDW9X/NT-CDR-V85X

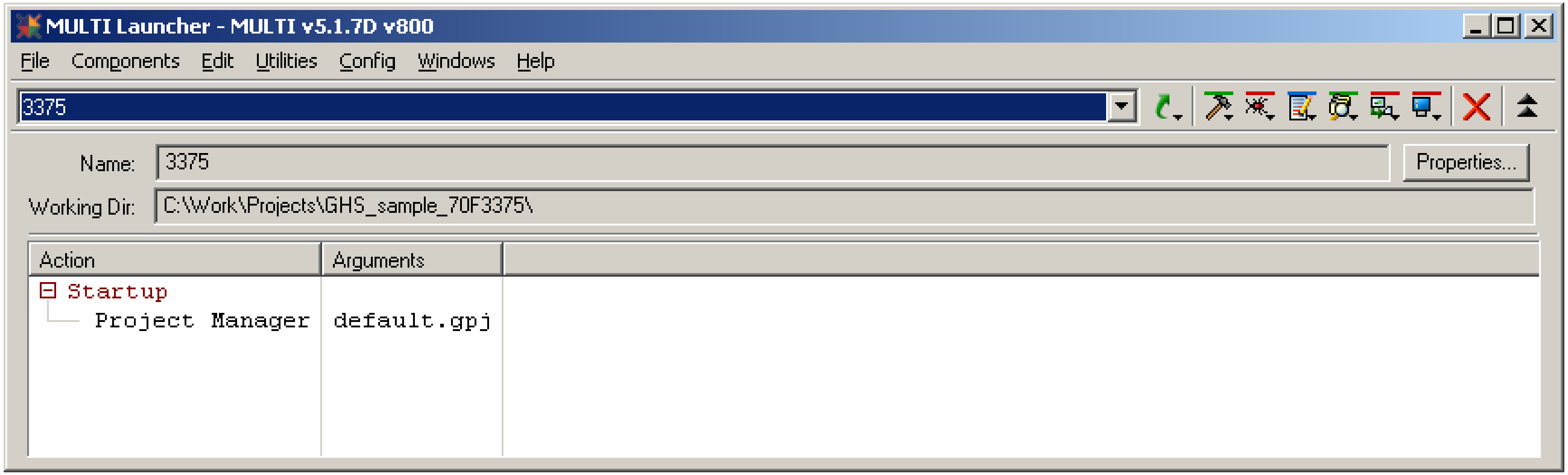
*: April, 2011

GHS MULTI IDE and project mangement

Overview GHS MULTI IDE

- **MULTI Launcher:** The gateway to the MULTI IDE, which allows you to quickly launch any of the primary MULTI tools, access open windows,and manage MULTI workspaces.
- **MULTI Editor:** A graphical editor for modifying text files. Next to the built-in editor MULTI allows to integrate any other favored editor.
- **MULTI Project Manager:** A graphical interface for managing and building projects.
- **MULTI Debugger:** A graphical source-level debugger.
- **Instruction Set Simulator:** A tool that simulates your embedded processor on your development host.
- **MULTI License Administrator:** A graphical utility for managing Green Hills tools licenses.

GHS MULTI Launcher



MULTI Launcher - MULTI v5.1.7D v800
File Components Edit Utilities Config Windows Help
Properties...

3375

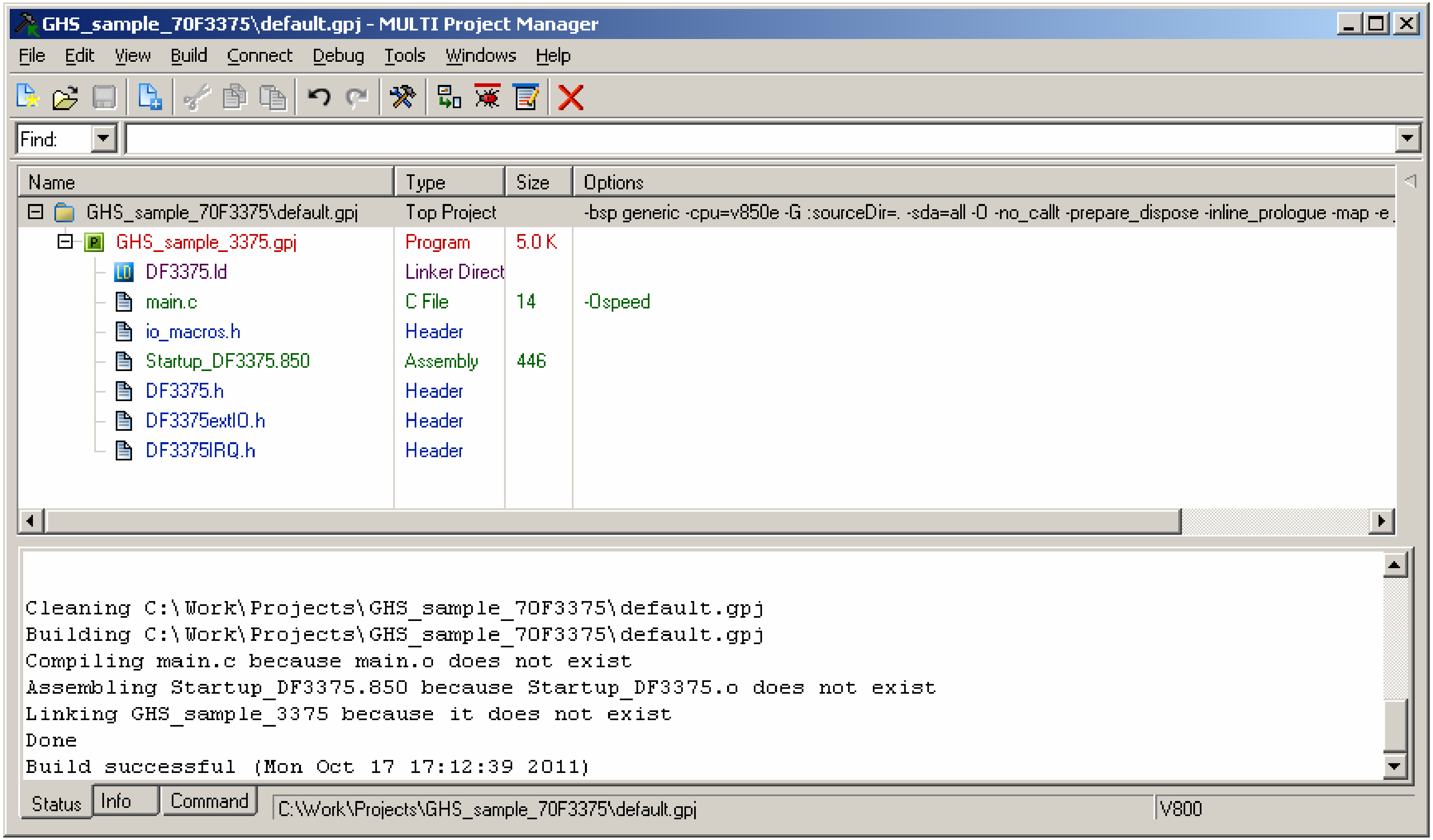
Name: 3375

Working Dir: C:\Work\Projects\GHS_sample_70F3375\

Action	Arguments
⊟ Startup	
Project Manager	default.gpj

- The gateway to the MULTI IDE, which allows you to quickly launch any of the primary MULTI tools, access open windows, and create and manage MULTI workspaces
- The launcher also allows to setup „Shortcuts“ and „Action sequences“. They execute in the working directory of the currently selected workspace

GHS MULTI Project Manager


GHS_sample_70F3375\default.gpj - MULTI Project Manager
File Edit View Build Connect Debug Tools Windows Help
Find:

Name	Type	Size	Options
GHS_sample_70F3375\default.gpj	Top Project		-bsp generic -cpu=v850e -G :sourceDir=. -sda=all -O -no_callt -prepare_dispose -inline_prologue -map -e
GHS_sample_3375.gpj	Program	5.0 K	
DF3375.ld	Linker Direct		
main.c	C File	14	-Ospeed
io_macros.h	Header		
Startup_DF3375.850	Assembly	446	
DF3375.h	Header		
DF3375extIO.h	Header		
DF3375IRQ.h	Header		

```
Cleaning C:\Work\Projects\GHS_sample_70F3375\default.gpj
Building C:\Work\Projects\GHS_sample_70F3375\default.gpj
Compiling main.c because main.o does not exist
Assembling Startup_DF3375.850 because Startup_DF3375.o does not exist
Linking GHS_sample_3375 because it does not exist
Done
Build successful (Mon Oct 17 17:12:39 2011)
```

C:\Work\Projects\GHS_sample_70F3375\default.gpj | V800

GHS MULTI project manager concept

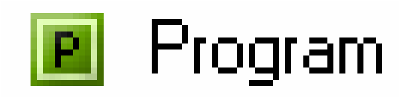
- The MULTI Project Manager is a graphical tool that organizes your source and other input files, and controls the tools needed to compile your software project.

- The term *project* is used to encompass all of the files that are used to build your application.

- Projects are defined in Green Hills *project files* (**.gpj**), which are similar to makefiles. The Project Manager maintains file dependencies, and sets the options used in building.

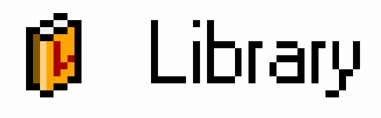
- Projects are organized in a tree structure, where the root of the tree is a Top Project (usually called **default.gpj**).

The different project types



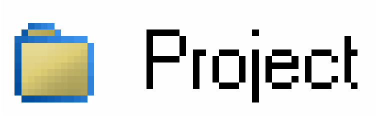
Program

- A framework for creating your own program. Only projects of types „program“ will generate an executable file



Library

- A framework for creating your own V850 library file (**.a**)

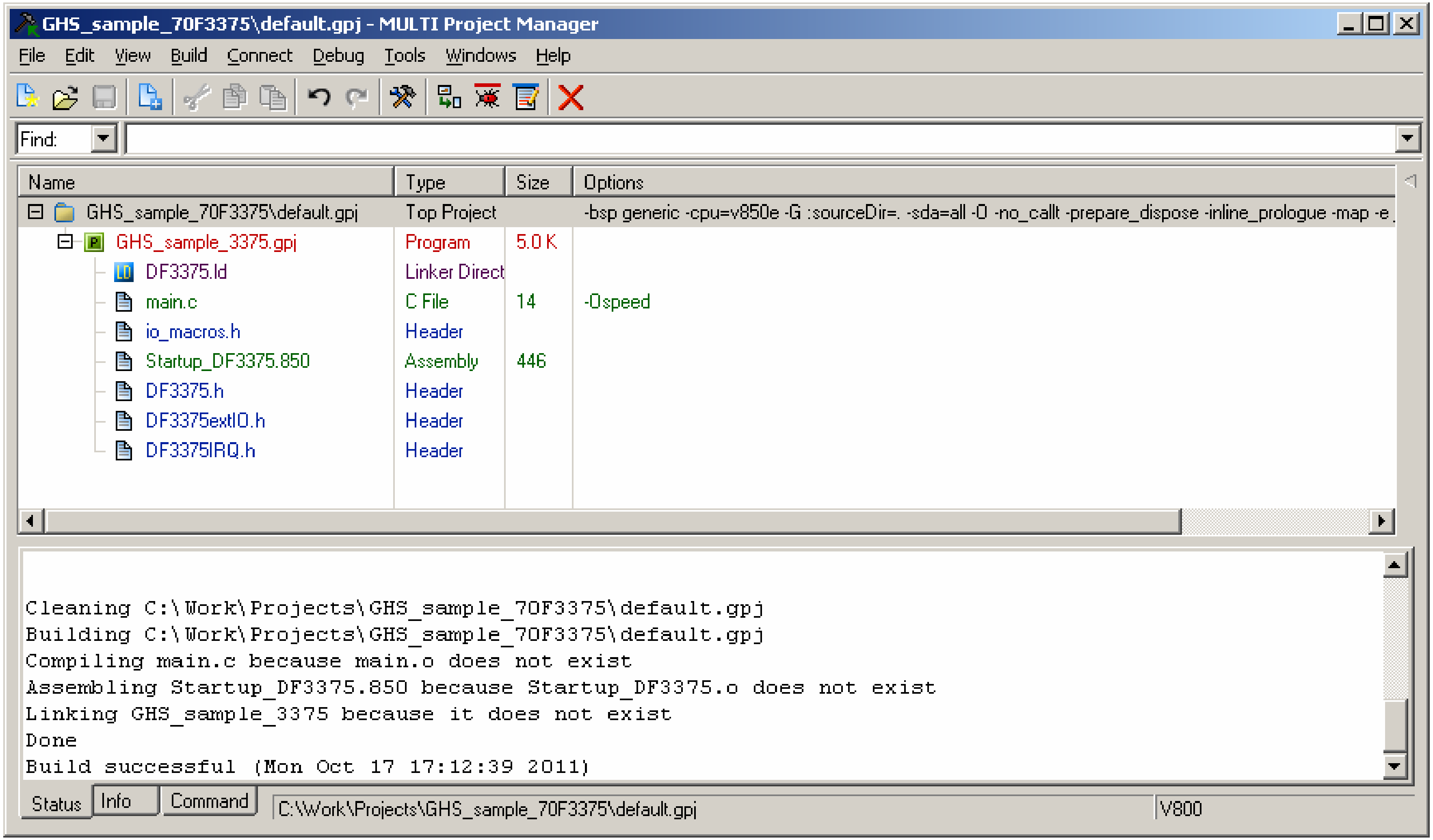


Project

- A general purpose container used to group programs, libraries, source code, and other projects. It can also be used to structure the project tree

All GHS project files are ASCII files and can be edited with a simple text editor

GHS MULTI Project Manager



Name	Type	Size	Options
GHS_sample_70F3375\default.gpj	Top Project		-bsp generic -cpu=v850e -G :sourceDir=. -sda=all -O -no_callt -prepare_dispose -inline_prologue -map -e
GHS_sample_3375.gpj	Program	5.0 K	
DF3375.ld	Linker Direct		
main.c	C File	14	-Ospeed
io_macros.h	Header		
Startup_DF3375.850	Assembly	446	
DF3375.h	Header		
DF3375extIO.h	Header		
DF3375IRQ.h	Header		

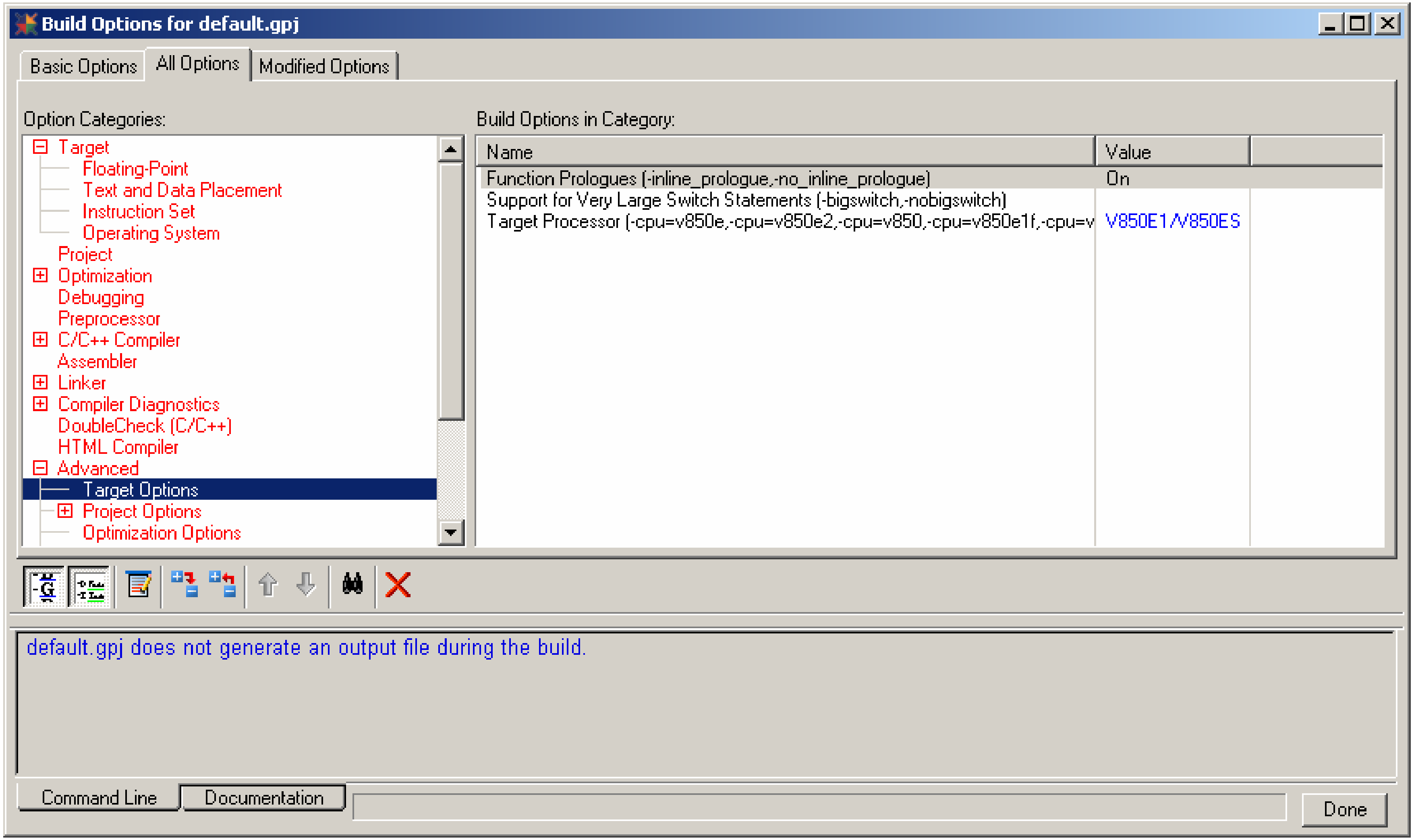
```
Cleaning C:\Work\Projects\GHS_sample_70F3375\default.gpj
Building C:\Work\Projects\GHS_sample_70F3375\default.gpj
Compiling main.c because main.o does not exist
Assembling Startup_DF3375.850 because Startup_DF3375.o does not exist
Linking GHS_sample_3375 because it does not exist
Done
Build successful (Mon Oct 17 17:12:39 2011)
```

C:\Work\Projects\GHS_sample_70F3375\default.gpj   V800

How to set project options

- Each Top-project, Sub-project or source file can have its individual project options.
- If one option is specified e.g. for Top-project and Sub-project, the option setting is overriden by the sub settings The same rule is also valid for source file options
- You can open the project settings window either by marking the file and then select „Edit -> Set build options...“ or by double click on the „Options“ column in the MULTI project manager main window
- Search function in the options window helps a lot to find a particular option setting mentioned in the GHS manuals
- The option settings in the GUI matches with the arguments used by the command line interface

GHS MULTI project options window



Name	Value
Function Prologues (-inline_prologue,-no_inline_prologue)	On
Support for Very Large Switch Statements (-bigswitch,-nobigswitch)	
Target Processor (-cpu=v850e,-cpu=v850e2,-cpu=v850,-cpu=v850e1f,-cpu=v	V850E1/V850ES

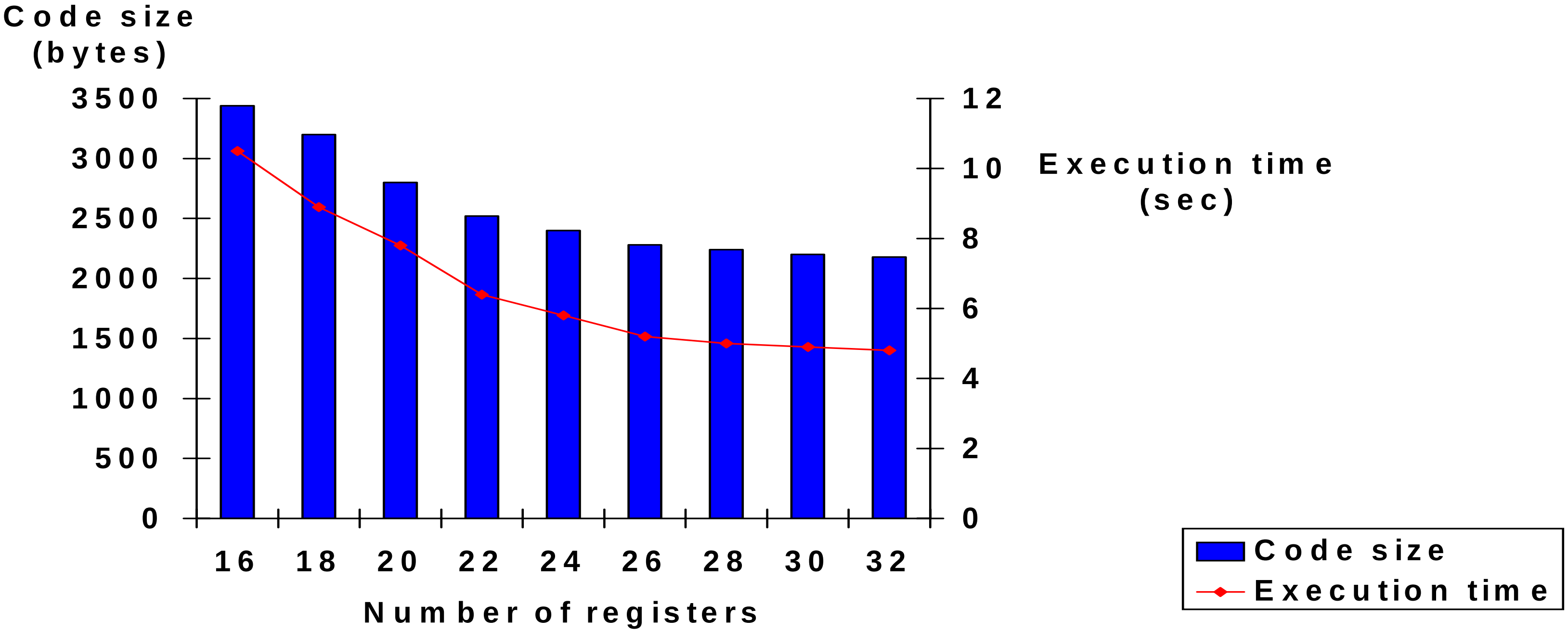
default.gpj does not generate an output file during the build.

GHS C / C++ compiler

Register usage and binary interface

Why 32 General Purpose Registers?

- Because of
  - Code size optimization and execution speed enhancement.
  - See comparison of performance/object code efficiency vs. number of registers of a servo control example program (compiled in C language):



Register Usage

Register	Alternate Name	Usage	Notes
R0	zero	zero	hardware
R1		address generation	
R2		temporary	can be reserved for user
R3	SP	stack pointer	
R4	GP	global pointer	used for .sdata & .sbss
R5	TP	text pointer	can be reserved for user / used for .rosdata
R6 - R9		parameters	
R10		function return value / temporary	
R11		high part of 64-bit return value / temporary	
R12 - R14		temporary	in all register modes
R15 - R16		temporary	reserved for user in 22 register mode
R17 - R19		temporary	reserved for user in 22 and 26 registers mode
R20 - R21		permanent / mask registers	reserved for user in 22 and 26 registers mode / 0xff 0xffff mask registers if selected
R22		permanent	reserved for user in 22 and 26 registers mode
R23 - R24		permanent	reserved for user in 22 register mode
R25 - R27		permanent	
R28		permanent / frame pointer	
R29		permanent / PIC base register	
R30	EP	temporary / tiny data base	EP is the only register for 16 bit load/store
R31	LP	link pointer	function return address

Register usage settings

- Reserve R2 for user
  - e.g. ISR stack pointer for an RTOS
- Reserve R5 for user
  - no separate TP for small data constants
- 22 and 26 register mode
  - registers can be used in assembly parts
  - less registers have to be saved by ISR
- mask registers for 0xff and 0xffff
  - 8 and 16 bit calculations can be done more effective

Register Usage considerations

- Do not mix up 32, 26 and 22 register mode in same program
  - unpredictable effects may occur especially with ISR

- Do not mix normal code with code reserving R2 or R5
  - Some RTOS use these registers and user program must not change them

- If you use mask registers for one module, you must also use them in all other modules
  - Compiler assumes R20==0xff R21==0xffff

Calling Conventions 1 / 3

- MCU Registers R6 - R9 are used to pass parameters of atomic type
- structures are stored on stack
- Any function can use temporary registers w/o restoring original values (except ISR)
  - functions can work even w/o stack
- The function’s return address is stored in LP (R31)
- The function’s return value is stored in R10 / R11

Calling Conventions 2 / 3

- C - source

```
int foo ( int p)
  {
        int i,j;

        i = bar (13, p) ;
        j = cit () ;
        wyk () ;
        return I+j ;
  }
```

- Assembly - code

```
        .text
        .align 2
        .globl _foo
_foo:
        add  -12,sp
        st.w  lp,8[sp]         -- save the return address
        st.w  r27,4[sp]        -- save permanent register r27
        st.w  r26,0[sp]        -- save permanent register r26

        mov  r6,r7             -- move p to r7
        mov  13,r6

        jarl   _bar, lp        -- call bar()
        mov  r10,r27           -- assign result of bar() to I

        jarl   _cit, lp        -- call cit()
        mov  r10,r26           -- assign result of cit() to j

        jarl   _wyk, lp        -- call wyk()
        mov  r26,r10           -- move j to r10

        add  r27,r10           -- r10 now contains i+j
        ld.w  0[sp],r26        -- restore r26
        ld.w  4[sp],r27        -- restore r27
        ld.w  8[sp],lp         -- restore lp
        add  12,sp
        jmp  [lp]              -- return from subroutine
```

Stack :

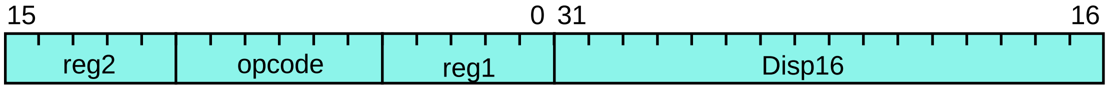
High Address	. . .	sp0
	lp	sp + 8
	r27	sp + 4
	r26	sp (=sp0 - 12)
Low Address	. . .	

V850 Memory Models

Why memory models

- V850 supports 32 bit address range
- Instructions for V850 are 16 or 32 bit wide
- loading data from any address costs 3 instructions
- normal load instructions have a signed 16 bit offset included
  - memory models take advantage of this offset

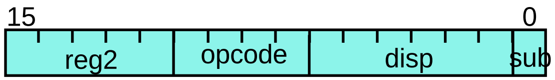
ld.x disp16[reg1], reg2



15 0 31 16

reg2 opcode reg1 Disp16

sld.x disp7/8[ep], reg2



15 0

reg2 opcode disp sub

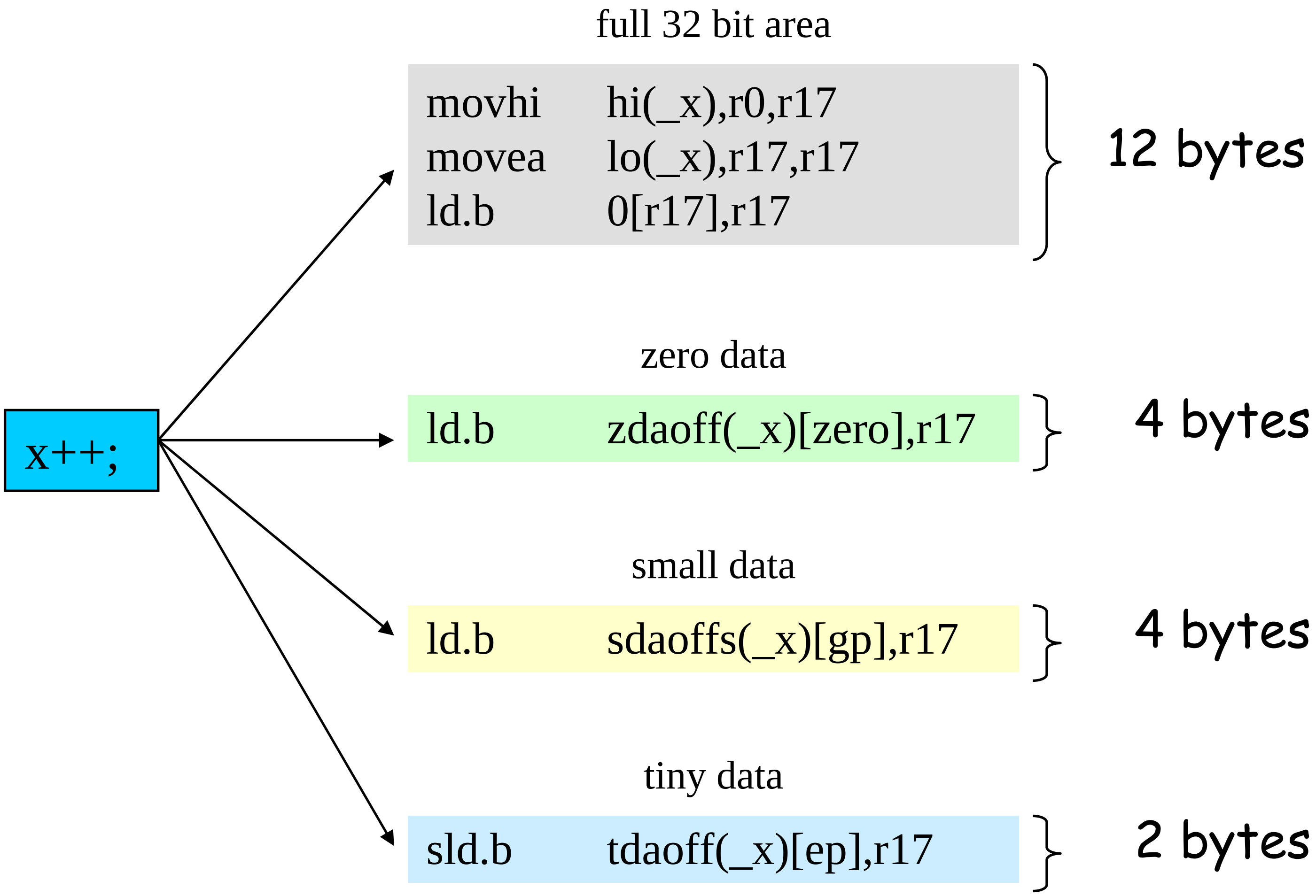
Memory models

- Small data optimization
  - two registers GP (R4) and TP (R5) are reserved to hold base address of .sbss/.sdata and .rosdata
  - 128 KB can be accessed with a single load instruction

- Zero data optimization
  - hardwired register R0 is used as a base address for .zbss/.zdata/.rozdata
  - 64 KB can be accessed with a single load instruction

- Tiny data optimization
  - EP is loaded with base address of .tdata and data within this section is accessed with a single 2 byte load instruction
  - 128/256 bytes can be accessed with a 2 byte load instruction
  - multiple tiny data support is possible, but loading EP could eliminate short load advantage

Memory models



x++;

full 32 bit area

```
movhi     hi(_x),r0,r17
movea     lo(_x),r17,r17
ld.b      0[r17],r17
```

12 bytes

zero data

```
ld.b      zdaoff(_x)[zero],r17
```

4 bytes

small data

```
ld.b      sdaoffs(_x)[gp],r17
```

4 bytes

tiny data

```
sld.b     tdaoff(_x)[ep],r17
```

2 bytes

Memory models

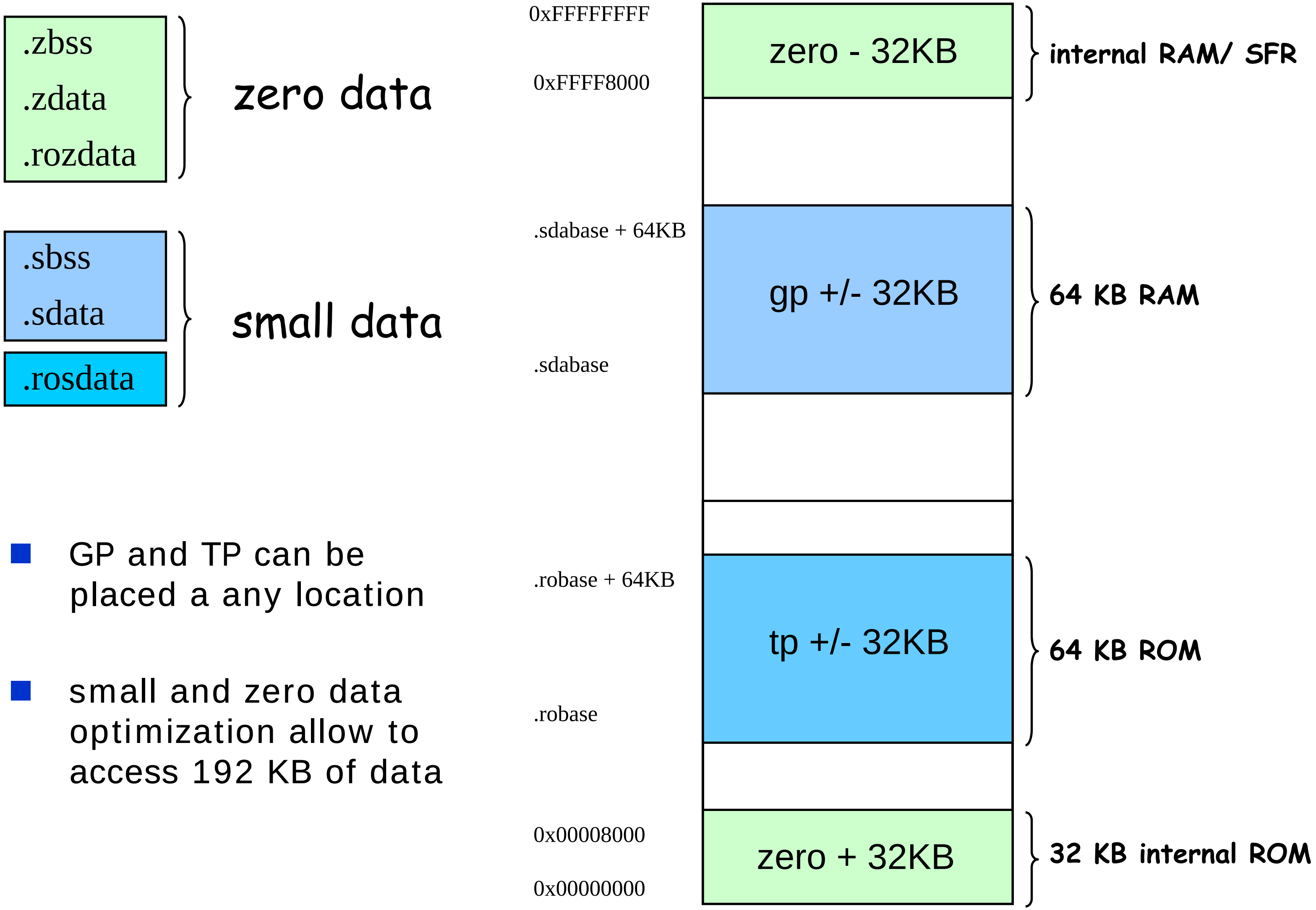
- SDA and ZDA addressing modes allowed
  - In the same module
  - Across modules in the same project

- zero and small data optimization allow to access 192 KB of data

- #pragma instructions or command line switches tell the compiler which memory model it should use for data

```
#pragma ghs startsda     #pragma ghs startzda
#pragma ghs endsda       #pragma ghs endzda
```

Memory models 3 / 3



- GP and TP can be placed a any location
- small and zero data optimization allow to access 192 KB of data

GHS Macros, Pragma directives and Intrinsic functions

Macros

- The preprocessor defines a huge number of macro names, depending on the compiler and language you use and the builder and driver options you specify.
- Useful for conditional compilation with #ifndef

Examples:

```
__V850__
__V850E__
__V850E2V3__

__LANGUAGE_ASM__
__LANGUAGE_C__
__LANGUAGE_CXX__
__STRICT_ANSI__

__cplusplus
__DATE__
__FILE__
__FULL_DIR__
__FULL_FILE__
__LINE__
__STDC__
__STDC_VERSION__
__TIME__

__BASE__
__FUNCTION__

__ghs_sda
__ghs_sda_threshold=n
__ghs_zda
__ghs_zda_threshold=n
__ghs_tda
```

Pragma directives

- #pragma directives allow individual compiler implementations to add special features to C programs without changing the C language.
- Programs that use #pragma directives stay relatively portable, although they make use of features not available in all ANSI C implementations.
- According to the ANSI standard, the #pragma directives that are not recognized by the compiler should be ignored.
- The majority of Green Hills proprietary #pragma directives begin with the keyword ghs to differentiate them from other implementations.
- The compiler considers any #pragma beginning with the ghs keyword to be recognized.

Pragma directives

- Following some of the most common used GHS pragma directives

```
#pragma alignfunc (n)       #pragma ghs interrupt
#pragma alignvar (n)
                            #pragma ghs OL
#pragma asm                 #pragma ghs OM
#pragma endasm              #pragma ghs OS
                            #pragma ghs ZO
#pragma pack (n)
                            #pragma ghs startsda
                            #pragma ghs endsda


#pragma intvect intfunc integer_constant

#pragma ghs section secttype="sectname"
```

Intrinsic functions

- Compiler included intrinsic functions perform certain tasks which are difficult or inefficient to write in a high-level language.
- Intrinsics do not affect the performance of the optimizer, like writing code using inline assembler
- Prototypes for the functions can be found in directory <*install_dir*>/include/v800/v800_ghs.h

Common used intrinsics are e.g.:

```
void __EI()                               int __SCH0R(int);
void __DI()                               int __SCH1R(int);
                                          int __SCH0L(int);
unsigned int __STSR(unsigned int)         int __SCH1L(int);
void __LDSR(unsigned int, unsigned int)

extern unsigned int __MULUH(unsigned int a, unsigned int b);
extern signed int __MULSH(signed int a, signed int b);
```

Compiler Settings

Recommended compiler settings

- -O
  - General optimization to generate minimized code size with improved code execution speed.

- -Ospeed for runtime critical modules
  - compile to optimize execution speed
  - turns on all the -O optimizations and the loop optimizations

- -Osize –inline_prologue –no_callt –prepare_dispose for general use
  - reduce code size (similar to driver options: -Osize and -OS)
  - turns on all the -O optimizations except those which increase the code size
  - Also please use ‘-O’ only!

Recommended compiler settings

- -inline_prologue
  - Places the functions prologue and epilogue in the function itself, instead of calling runtime library calls to prepare stack frames.
- --no_callt
  - Generates bigger, but faster code
- -prepare_dispose
  - Enables prepare and dispose instructions for function pro and epilogues
  - Very efficient for interrupt service routines

Recommended compiler settings

- -cpu=v850e, -cpu=v850e2v3
  - Sets the instruction set for the particular V850 core architecture which will be used by the compiler

- -sda=all
  - allows to access 64KB RAM and 64KB ROM data
  - should be sufficient for single chip applications
  - very effective code

- -entry=_RESET
  - If not specified, debugger starts program at “__start.
  - This entry is required in case you use the Renesas startup code provided with DFxxxx_startup.850 assembly file.

Recommended compiler settings

- -list -passsource
  - generate assembler list file with C/C++ source code
- -map
  - This is default setting. Will generate a linker map file.
- -delete (Linker ELXR)
  - Deletion of unreferenced/unused functions
- -g
  - generate debug information
  - -G does also generate debug information, but also include additional code for debugging
  - using –g instead of –G saves approx. 1000 Bytes .bss-memory

Programming IO

Programming IO

- Renesas publishes always the latest available device specific GHS packages on

  http://www.renesas.eu/update

- The package consists of
  - Device File (.800)
    - needed to establish a debugger connection
  - io_macros.h
    - predefined macros for SFR programming
  - DFxxx.h and DFxxx_EXTIO.h
    - SFR definitions for both, assembler and C/C++
  - DFxxx_IRQ.h
    - System wide defines for available interrupt sources
  - DFxxx_Startup.850
    - The device Reset routine and ISR jump tables

Programming IO

- Also included in the device file package
  - DFxxxx.ld
    - Linker directive file template, containing memory map and section declarations for the particular device
  - DFxxxx.grd
    - GHS register definitions file, can be loaded into the MULTI debuggers register view to show all SFR‘s of the particular device

Programming IO

- The IO or SFR registers can be accessed by
  - Zero pointers, as implemented in the device header files

```
// clock select
   TP0CTL0 = 0x03 ; // fxx/8
   TP1CTL0 = 0x03 ; // fxx/8
   TP2CTL0 = 0x03 ; // fxx/8
```

  - Application, respectively user defined pointers to structures
    - This method may be more secure than the previous one, but will produce more code to access an SFR.

Programming IO

- Startup code provides a skeleton for basic device initialization.
  - All possible interrupts are defined
  - RESET is the standard entry point
  - Basic Initialization of stack and global pointers
  - Jumps directly to __start(), the main entry point for the compiler.

- Possible Modifications include the bus tuning of the V850 devices to improve boot speed. The function __lowinit can be used for this purpose.

Interrupt Support

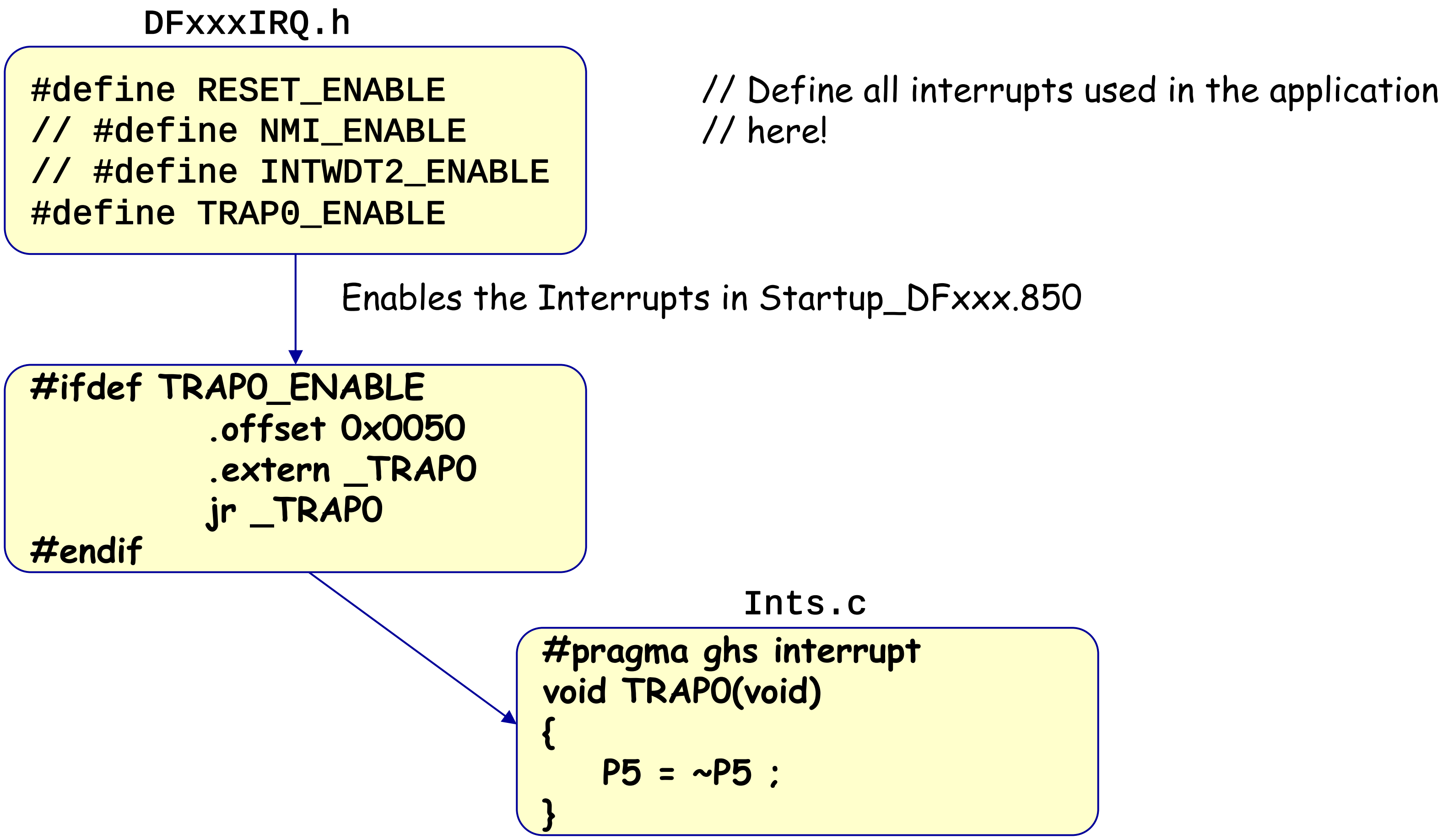
Interrupt Support

- Green Hills provides sophisticated support for ISR design in C.
  - All Interrupts are reentrant, if __EI() is specified!
- Two ways to implement an interrupt
  - #pragma ghs interrupt
    - The function below is an interrupt enable in a header file and the name must match the corresponding label name in startup.850
  - Keyword "#pragma intvect" creates ISRs 'On-The-Fly'.
    - No activation necessary in header file and startup.850.

```
#pragma ghs interrupt
void TRAP0(void)
{
    P5 = ~P5 ;
}
```

```
__interrupt void trap1(void);
#pragma intvect trap1 0x050

__interrupt void trap1()
{
    P5 = ~P5 ;
}
```

Interrupt Support



DFxxxIRQ.h

```
#define RESET_ENABLE
// #define NMI_ENABLE
// #define INTWDT2_ENABLE
#define TRAP0_ENABLE
```

// Define all interrupts used in the application
// here!

Enables the Interrupts in Startup_DFxxx.850

```
#ifdef TRAP0_ENABLE
            .offset 0x0050
            .extern _TRAP0
            jr _TRAP0
#endif
```

Ints.c

```
#pragma ghs interrupt
void TRAP0(void)
{
    P5 = ~P5 ;
}
```

A save, O/S independent implementation!

MISRA-C

MISRA-C

MISRA The Motor Industry Software Reliability Association

- Specification of a C programming language subset
- Targets are:
  - suitable for embedded automotive systems
  - increase safety level
- Details are described in:
  - MISRA-C:2004
  - MISRA-C:1998 (obsolete)

MISRA-C:2004

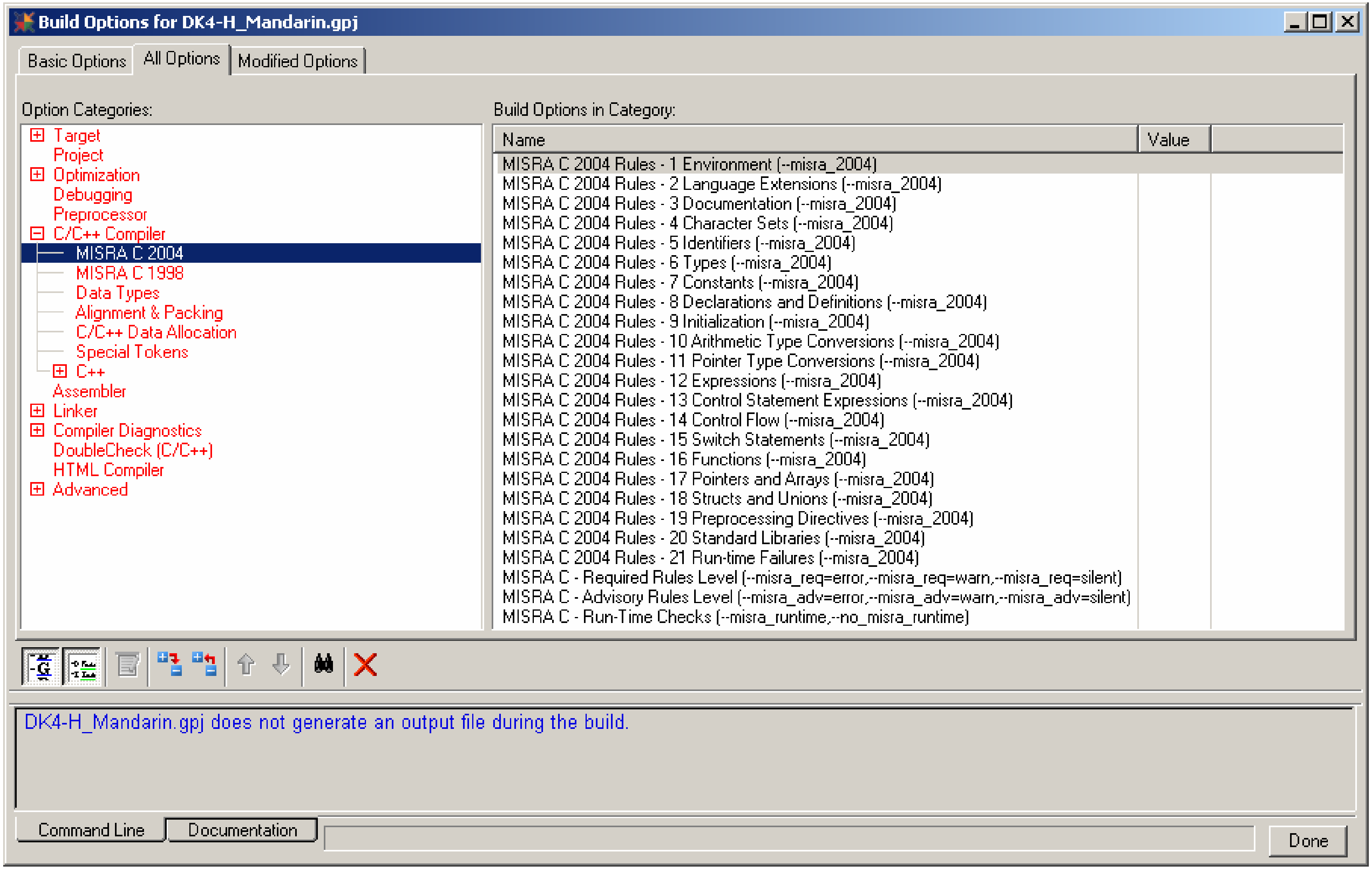
Guidelines for the use of the C language in critical systems

GHS MISRA-C checker

- GHS contains options which allow to control compliance with the MISRA-C rule set. Both specifications, MISRA-C:1998 and MISRA-C:2004 are covered by the rule checker
- GHS allows to set or ignore all or individual MISRA-C rules
- MISRA rules can be treated either as error, warning or silent
- Examples:

--misra_2004=all	Enables checking of all the rules
--misra_2004=none	Disables checking of all the rules
--misra_2004=all,-2.3	Enables all rules except rule 2.3

GHS MISRA-C checker



Build Options for DK4-H_Mandarin.gpj

Option Categories:

- ⊞ Target
- Project
- ⊞ Optimization
- Debugging
- Preprocessor
- ⊟ C/C++ Compiler
  - MISRA C 2004
  - MISRA C 1998
  - Data Types
  - Alignment & Packing
  - C/C++ Data Allocation
  - Special Tokens
  - ⊞ C++
- Assembler
- ⊞ Linker
- ⊞ Compiler Diagnostics
- DoubleCheck (C/C++)
- HTML Compiler
- ⊞ Advanced

Build Options in Category:

Name	Value
MISRA C 2004 Rules - 1 Environment (--misra_2004)	
MISRA C 2004 Rules - 2 Language Extensions (--misra_2004)	
MISRA C 2004 Rules - 3 Documentation (--misra_2004)	
MISRA C 2004 Rules - 4 Character Sets (--misra_2004)	
MISRA C 2004 Rules - 5 Identifiers (--misra_2004)	
MISRA C 2004 Rules - 6 Types (--misra_2004)	
MISRA C 2004 Rules - 7 Constants (--misra_2004)	
MISRA C 2004 Rules - 8 Declarations and Definitions (--misra_2004)	
MISRA C 2004 Rules - 9 Initialization (--misra_2004)	
MISRA C 2004 Rules - 10 Arithmetic Type Conversions (--misra_2004)	
MISRA C 2004 Rules - 11 Pointer Type Conversions (--misra_2004)	
MISRA C 2004 Rules - 12 Expressions (--misra_2004)	
MISRA C 2004 Rules - 13 Control Statement Expressions (--misra_2004)	
MISRA C 2004 Rules - 14 Control Flow (--misra_2004)	
MISRA C 2004 Rules - 15 Switch Statements (--misra_2004)	
MISRA C 2004 Rules - 16 Functions (--misra_2004)	
MISRA C 2004 Rules - 17 Pointers and Arrays (--misra_2004)	
MISRA C 2004 Rules - 18 Structs and Unions (--misra_2004)	
MISRA C 2004 Rules - 19 Preprocessing Directives (--misra_2004)	
MISRA C 2004 Rules - 20 Standard Libraries (--misra_2004)	
MISRA C 2004 Rules - 21 Run-time Failures (--misra_2004)	
MISRA C - Required Rules Level (--misra_req=error,--misra_req=warn,--misra_req=silent)	
MISRA C - Advisory Rules Level (--misra_adv=error,--misra_adv=warn,--misra_adv=silent)	
MISRA C - Run-Time Checks (--misra_runtime,--no_misra_runtime)	

DK4-H_Mandarin.gpj does not generate an output file during the build.

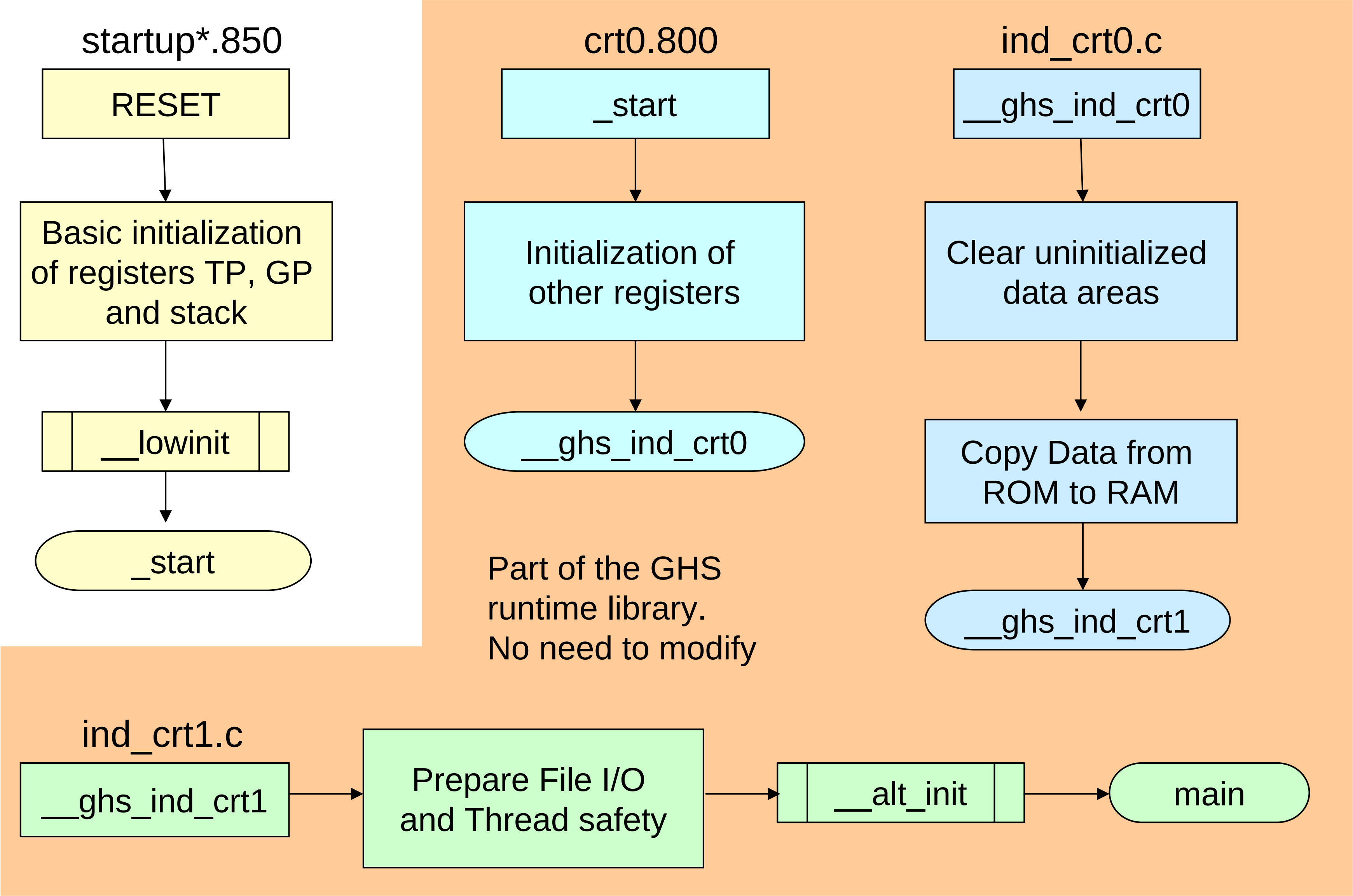
GHS Startup files and Runtime initialization

GHS Startup files and Runtime initialization

- **Note:** It is not required to add Green Hills-provided target libraries manually to your project. The compiler will link in the correct target libraries for your configuration when the appropriate Builder options are set.

- The below case only applies if you want to optimize the GHS startup libraries on your own

Reset Procedure


startup*.850
RESET
Basic initialization of registers TP, GP and stack
__lowinit
_start
crt0.800
_start
Initialization of other registers
__ghs_ind_crt0
ind_crt0.c
__ghs_ind_crt0
Clear uninitialized data areas
Copy Data from ROM to RAM
__ghs_ind_crt1
ind_crt1.c
__ghs_ind_crt1
Prepare File I/O and Thread safety
__alt_init
main

Part of the GHS runtime library.
No need to modify

Renesas Startup Files

- Please use Renesas provided startup code just after RESET is issued.
  - This initialized the most important registers, such as
    - Global pointer (GP)
    - Text pointer (TP)
    - Stack pointer (SP)
  - RESET() calls a subprocess named __lowinit().
    - If lowinit is not existing, no warning is issued and execution is just continued to _start.
  - The __lowinit routine maybe implemented in C,
  - But it may neither
    - use/ corrupt the stack, nor
    - make use of special address modes, such as zda or sda
  - Use __lowinit to initialize clock tree or other important SFRs

The GHS Standard Startup Module crt0.850

- Main entry point for GHS programs is the routine _start in module <ghs-dir>\src\libstartup\crt0.800
  - _start() is
    - assembler based
    - CPU core oriented
    - Set default register values
    - Calculates basic pointers important for PIC/PID programming

- As _start is written in assembler, there is not much room to improve code size or execution speed.

Architecture Independent Initialization

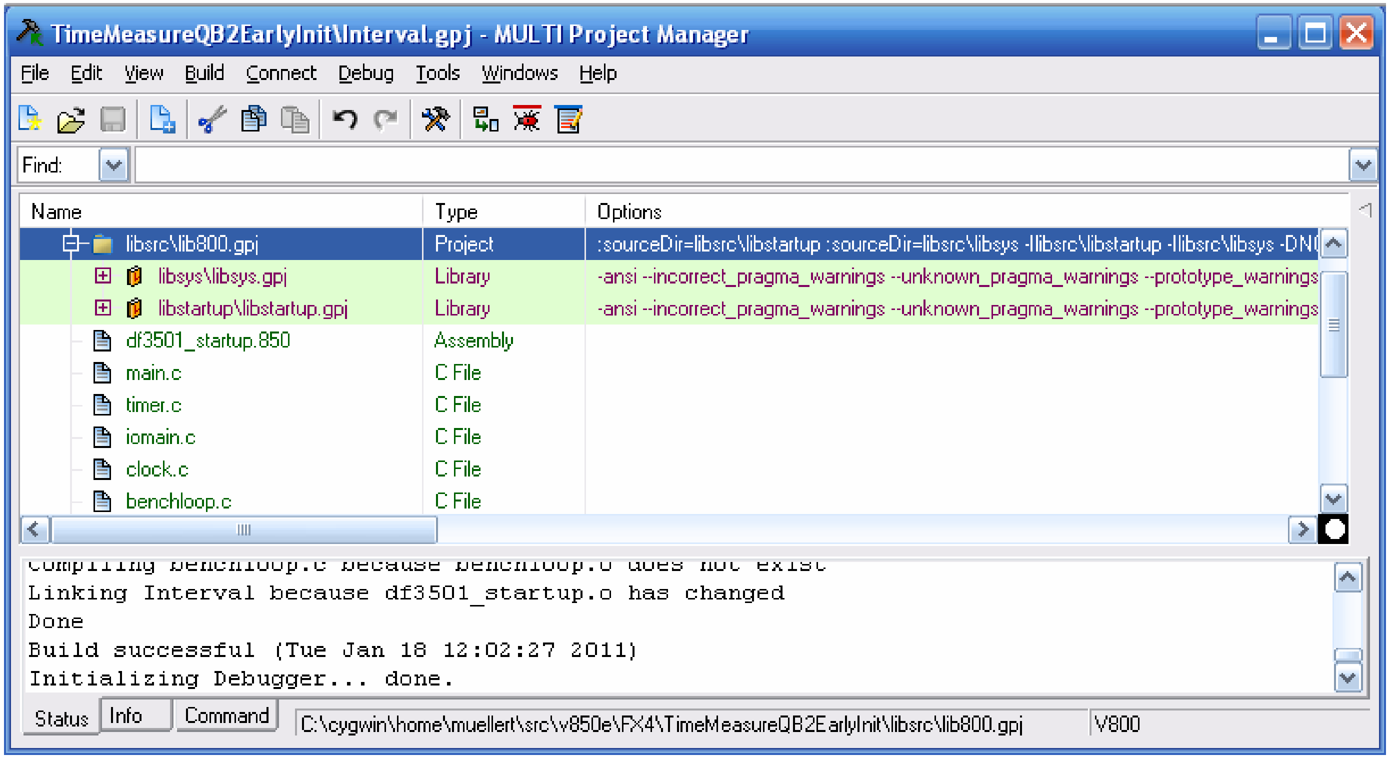
- GHS provides architecture independent initialization with the routines
  - ind_crt0.c and
  - ind_crt1.c

- ‘ind_crt0’ provides the skeleton to initialize the applications memory areas.
  - RAM is cleared
  - Constant data and initial values are copied into RAM
  - Each section is handled separately.
  - The linker directive file determines the action taken in this module.

- ‘ind_crt1’ is located under <ghs-dir>\src\libsys and provides
  - iobuffer and lock functionality (thread safety)

Rebuilding Sources

- Create a directory in you project path, such as libsrc
- Copy the directories of <ghs-dir>\src\*. To that target directory
- Create a subproject to incorporate both projects
  - ‘libstartup’ and ‘libsys’



Name	Type	Options
libsrc\lib800.gpj	Project	:sourceDir=libsrc\libstartup :sourceDir=libsrc\libsys -Ilibsrc\libstartup -Ilibsrc\libsys -DNI
libsys\libsys.gpj	Library	-ansi --incorrect_pragma_warnings --unknown_pragma_warnings --prototype_warnings
libstartup\libstartup.gpj	Library	-ansi --incorrect_pragma_warnings --unknown_pragma_warnings --prototype_warnings
df3501_startup.850	Assembly	
main.c	C File	
timer.c	C File	
iomain.c	C File	
clock.c	C File	
benchloop.c	C File	

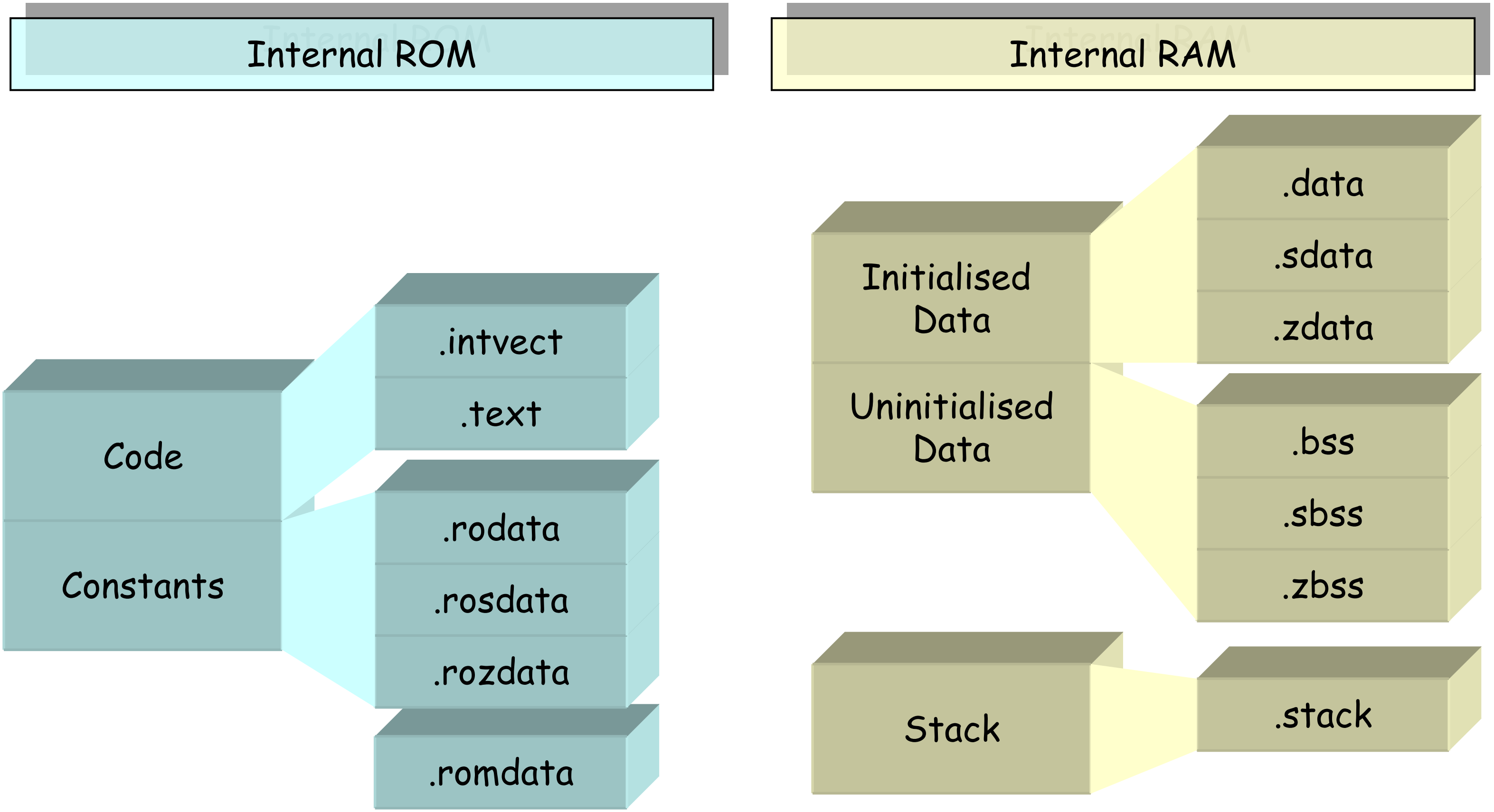
```
Compiling benchloop.c because benchloop.o does not exist
Linking Interval because df3501_startup.o has changed
Done
Build successful (Tue Jan 18 12:02:27 2011)
Initializing Debugger... done.
```

C:\cygwin\home\muellert\src\v850e\FX4\TimeMeasureQB2EarlyInit\libsrc\lib800.gpj V800

GHS Linker EXLR

Sections

- 4 different basic section types are known
  - rodata, text, bss, data
    - 2 subtypes ca be determined



Internal ROM

Code

Constants

.intvect

.text

.rodata

.rosdata

.rozdata

.romdata

Internal RAM

Initialised Data

Uninitialised Data

Stack

.data

.sdata

.zdata

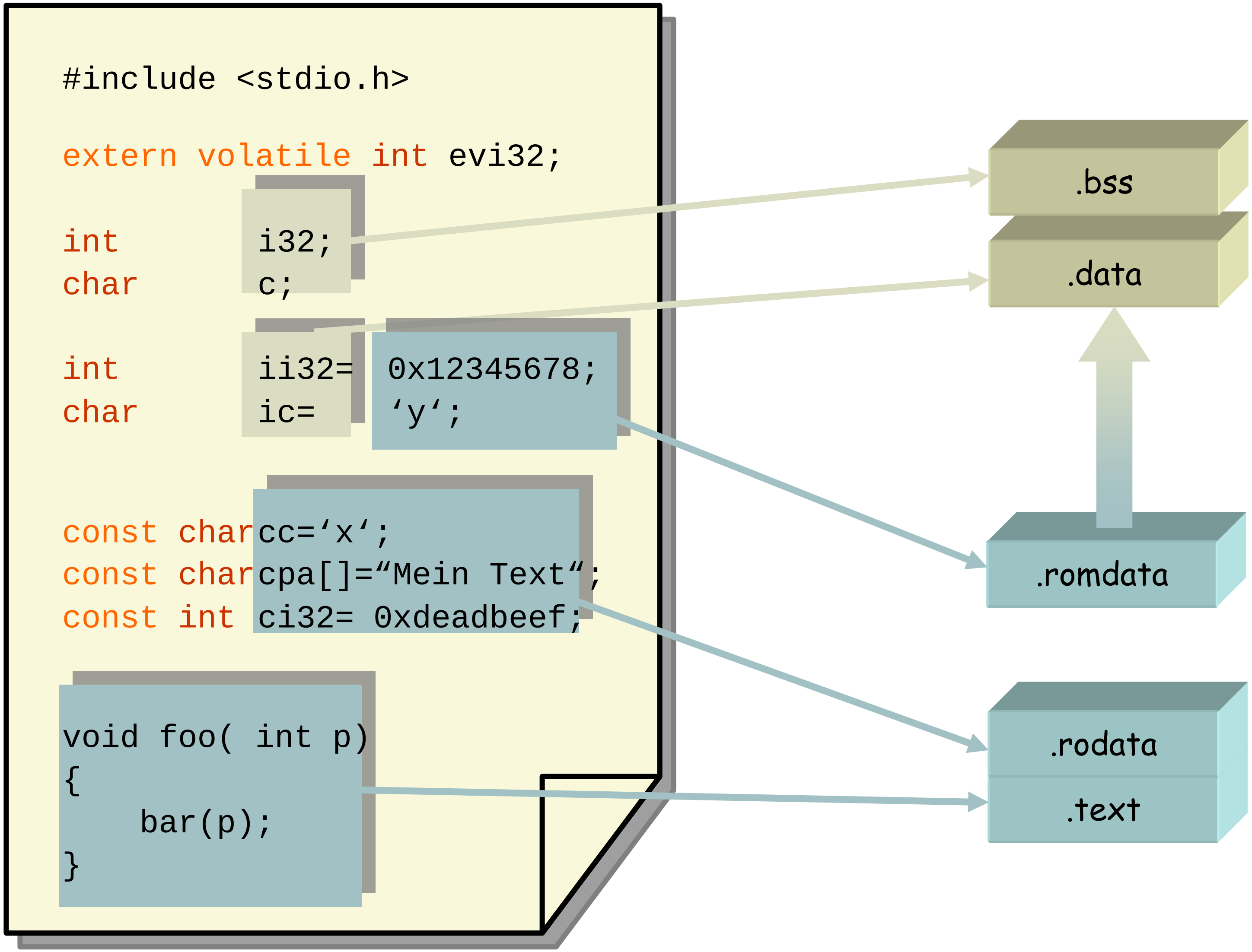
.bss

.sbss

.zbss

.stack

Sections



```
#include <stdio.h>

extern volatile int evi32;

int        i32;
char       c;

int        ii32= 0x12345678;
char       ic=  ‘y‘;


const char cc=‘x‘;
const char cpa[]=“Mein Text“;
const int  ci32= 0xdeadbeef;

void foo( int p)
{
    bar(p);
}
```

Linker Directive File

```
MEMORY
 {
  iROM : ORIGIN = 0x00000000, LENGTH = 640k
  iRAM : ORIGIN = 0xFFFF3000, LENGTH = 48k
 }
```

Memory Map

SECTIONS {	Section Declaration	
/* Start of internal ROM area (iROM) */ .intvect :>iROM .intvect_end 0x0430 :>.	ISR Vector Table	.section “.intvect”,.text .offset 0x00 jr _reset
.rozdata :>.	zero data constants	#pragma ghs startzda // or -zda=all const int x=5;
.robase align(4) :>. .rosdata align(4) :>.	small data constants	#pragma ghs startsda // or -sda=all const int x=5;
.rodata align(4) :>.	normal data constants	#pragma ghs startdata const int x=5;
.text align(4) :>.	normal code	void(main(void) { ...

Linker Directive File

. # [..] .fixaddr align(4) :>. .fixtype align(4) :>. .secinfo align(4) :>.	section information for startup code	Sections to clear or to copy
.syscall align(4) :>.	Host based f file I/O	Sections to clear or to copy
.romdata ROM(.data) :>. .romzdata ROM(.zdata) :>. .romsdata ROM(.sdata) :>. .romtdata ROM(.tdata) :>.	ROM copies of initialized data	memcpy(data,. romdata, sizeof(.data));

.data align(4) :> iRAM	initialized data	#pragma ghs startdata int x=5;
.bss align(4) :>.	Un-initialized data	#pragma ghs startdata int y;
.sdabase align(4) :>.	Small data base	

.sdata align(4) :>.	small data	#ragma ghs startsda
		int x=5;
		int y;

.sbss align(4) :>.		

Linker Directive File

.zdata align(4) :>.	zero data	#ragma ghs startzda
		int x=5;
		Int y;

.zbss align(4) :>.		
.tdata align(4) MAX_SIZE(256) :>.	tiny data	#pragma ghs starttda int y;
.stack align(4) PAD(0x200) :>. }	stack	

User defined Sections

- The linker support user-defined sections of all types!
  - Renamed sections, as they are declared in C-modules have to be declared in the linker script.

C-Module

```
#pragma ghs section text=“.mytext”
void bootme( void)
{
      …
}
#pragma ghs section text=default
```

Linker Script File

```
SECTIONS
{
      …
      .text align(4)              :>.
      .mytext align(4)            :>.
      …
}
```

User defined Sections

- It is also possible to allocate entire libraries or user defined modules to specific sections!
  - The example below demonstrates how the sections of modules crt0.o and all library routines are placed in absolute located section “.mytext”

Linker Script File

```
SECTIONS
{
        ...
        .text align(4)       :>.
        .mytext align(4)   : {crt0.o(.text) libsys.a(*(.text))}>.
        ...
}
```

Linker Output Analysis

- The linker generates by default a map listing containing
  - Image Summary
  - Module Summary
  - Global Symbols sorted by address
- Optional listings include
  - Cross Reference
  - Global Symbols sorted by name
- Two additional files may be generated, if a callgraph is required or a global size analysis per module.

GHS Librarian AX

Creating and Using Libraries with GHS

- The librarian combines object files created by the assembler or linker into a library (or archive file). By convention, libraries have the **.a** extension. At link time, the linker can search through libraries for object files, and pull them in to provide definitions for undefined symbols.

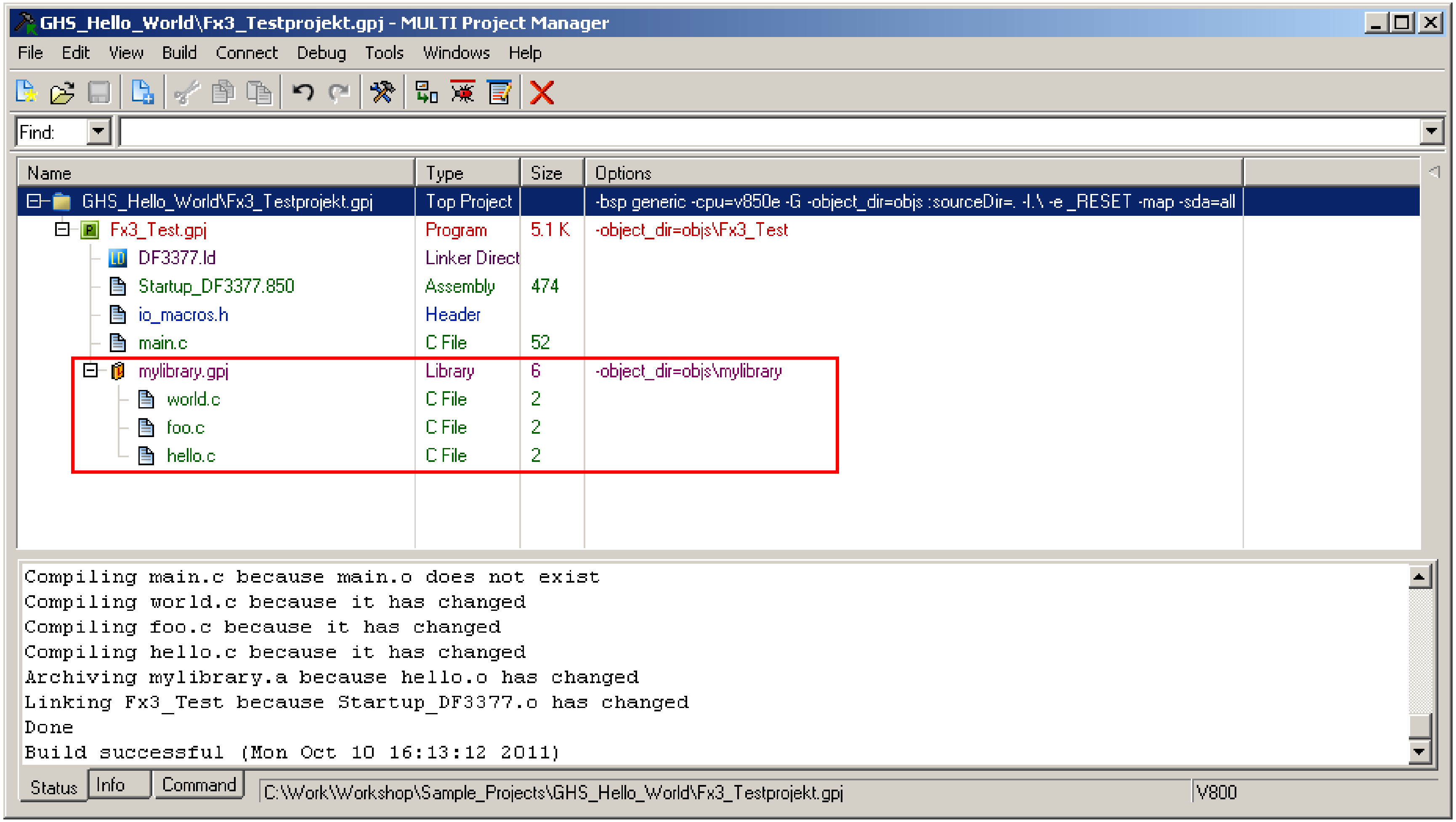
- Libraries can be created either by using the compiler driver or by using the GHS MULTI project manager and the GUI

  Example, using the compiler driver:

```
ccv850 foo.c -archive -o libfoo.a

ccv850 hello.o world.o -archive -o libhello.a
```

Creating and Using Libraries with GHS

- Creating a library using the MULTI project manager



Name	Type	Size	Options
GHS_Hello_World\Fx3_Testprojekt.gpj	Top Project		-bsp generic -cpu=v850e -G -object_dir=objs :sourceDir=. -I.\ -e _RESET -map -sda=all
Fx3_Test.gpj	Program	5.1 K	-object_dir=objs\Fx3_Test
DF3377.ld	Linker Direct		
Startup_DF3377.850	Assembly	474	
io_macros.h	Header		
main.c	C File	52	
mylibrary.gpj	Library	6	-object_dir=objs\mylibrary
world.c	C File	2	
foo.c	C File	2	
hello.c	C File	2	

```
Compiling main.c because main.o does not exist
Compiling world.c because it has changed
Compiling foo.c because it has changed
Compiling hello.c because it has changed
Archiving mylibrary.a because hello.o has changed
Linking Fx3_Test because Startup_DF3377.o has changed
Done
Build successful (Mon Oct 10 16:13:12 2011)
```

C:\Work\Workshop\Sample_Projects\GHS_Hello_World\Fx3_Testprojekt.gpj

V800

Creating and Using Libraries with GHS

- Using a Pre-Built Library in an existing project is quiet simple. Just add the library module to the program project file.

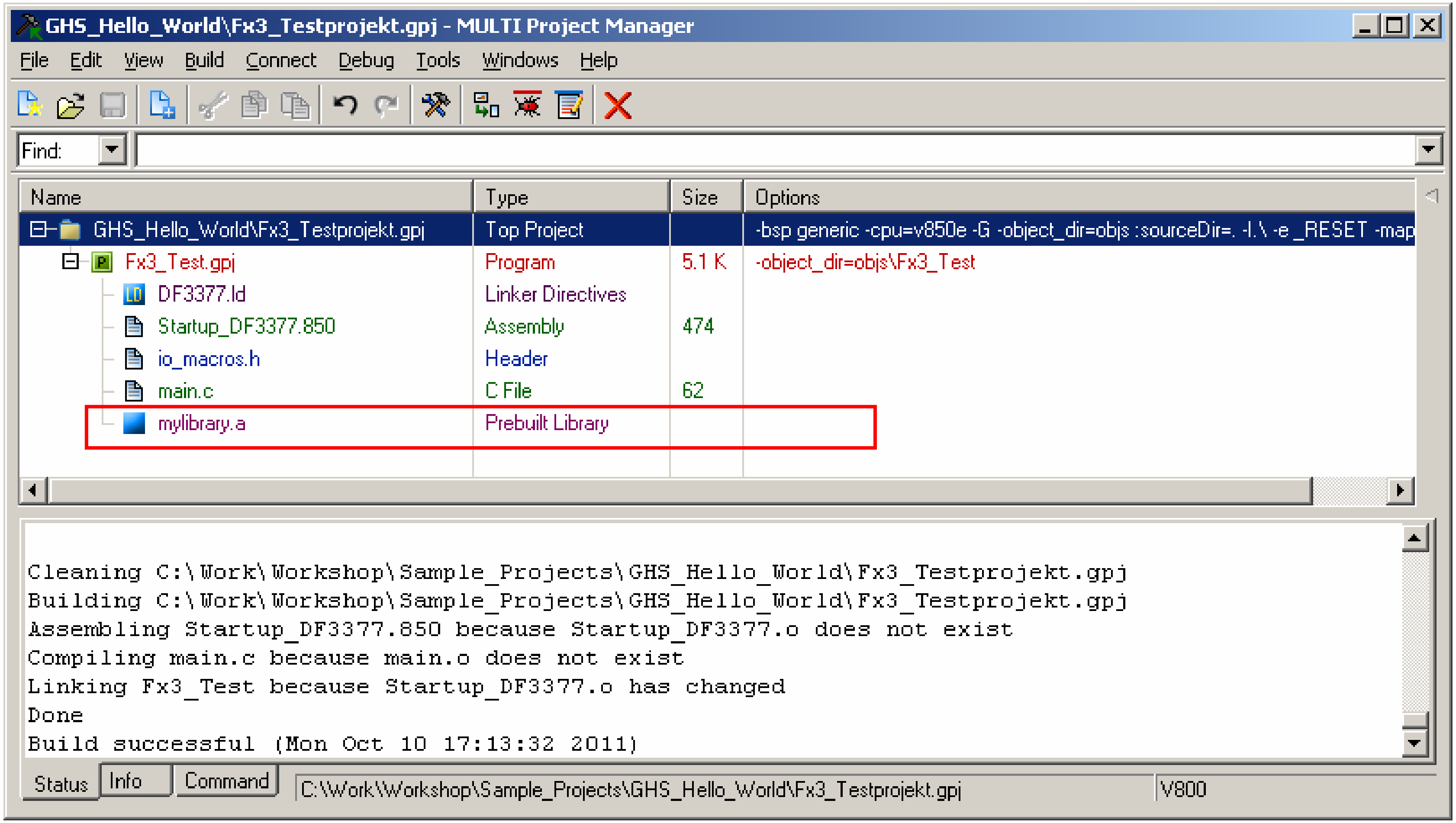
- **Note:** Use this method when the source code for the pre-built library is unavailable. If you do have the source code for the library, use a **Library** project instead, because MULTI will rebuild the source files as necessary.

- **Warning:** Never manually add Green Hills-provided target libraries to your project (for example, libind.a). The compiler will link in the correct target libraries for your configuration when the appropriate Builder options are set.

- For further details see also GHS manual **build_v800.pdf** (Chapter 12, The ax Librarian)

Creating and Using Libraries with GHS

- Using a Pre-Built Library using the MULTI project manager



GHS_Hello_World\Fx3_Testprojekt.gpj - MULTI Project Manager
File Edit View Build Connect Debug Tools Windows Help
Find:

Name	Type	Size	Options
GHS_Hello_World\Fx3_Testprojekt.gpj	Top Project		-bsp generic -cpu=v850e -G -object_dir=objs :sourceDir=. -I.\ -e _RESET -map
Fx3_Test.gpj	Program	5.1 K	-object_dir=objs\Fx3_Test
DF3377.ld	Linker Directives		
Startup_DF3377.850	Assembly	474	
io_macros.h	Header		
main.c	C File	62	
mylibrary.a	Prebuilt Library		

```
Cleaning C:\Work\Workshop\Sample_Projects\GHS_Hello_World\Fx3_Testprojekt.gpj
Building C:\Work\Workshop\Sample_Projects\GHS_Hello_World\Fx3_Testprojekt.gpj
Assembling Startup_DF3377.850 because Startup_DF3377.o does not exist
Compiling main.c because main.o does not exist
Linking Fx3_Test because Startup_DF3377.o has changed
Done
Build successful (Mon Oct 10 17:13:32 2011)
```

C:\Work\Workshop\Sample_Projects\GHS_Hello_World\Fx3_Testprojekt.gpj V800

GHS Utility programs

GHS Utility programs

- The GHS MULTI installation includes many useful command line utility programs, including functional replacements for the standard UNIX utilities **dump**, **file**, **hide**, **nm**, **size**, **strip**, and **what**. All utility programs work with files generated by any Green Hills development tools.

  - **gasmlist**: Generates interlaced assembly and source output (object files only).
  - **gbin2c**: Converts binary files into C array definitions.
  - **gbincmp**: Compares two binary files (single object files or executables only).
  - **gbldconvert**: Converts old-style build files (**.bld**) to new-style Green Hills project files (**.gpj**).
  - **gbuild**: A command line interface to the MULTI Builder.
  - **gcolor**: Takes text input and colors it using ANSI escape sequences.
  - **gcompare**: Compares space or time performance.

GHS Utility programs

- **gdump**: Similar to the UNIX **dump** program. Dumps or disassembles a file.
- **gfile**: Similar to the UNIX **file** program. Describes the file type.
- **gfunsize**: Returns the code size of a function.
- **ghide**: Similar to the UNIX **hide** program. Hides global symbols in an object file.
- **gmemfile**: Converts an executable file to a binary image suitable for loading.
- **gnm**: Similar to the UNIX **nm** program. Displays file information.
- **gpjmodify**: Performs command line editing of new-style **.gpj** project files.
- **grun**: (for executable files) Runs a server in batch mode.
- **gsize**: Similar to the UNIX **size** program. Displays section sizes.
- **gsrec**: Converts an executable file to Motorola S-Record format, Intel hexadecimal, or Tektronix hexadecimal format file.
- **gstack**: (for executable files) Computes stack size for each task.

GHS Utility programs

- **gstrip**: Similar to the UNIX **strip** program. Removes symbol or debugging information from an executable.
- **gversion**: (for executable files) Returns version date and time information.
- **gwhat**: Similar to the UNIX **what** program. Reports or updates version information.

- The Utilites are located in the root-directory of the GHS MULTI installtion and can be started either from command-line or via MULTI Launcher (Utilities -> Launch Utility Programs) or MULTI Project Manager (Tools -> Use Utilities)

- Detailed information regarding usage and options of every utility can be found in the GHS Manual **build_v800.pdf** (Chapter 13, Utility programs)

GHS MULTI debugger and 850eserv2 target server

GHS MULTI Debugger Concept

- The GHS MULTI debugger is a generic debugger supporting different CPU architectures (e.g. Renesas V850 + SH, and competitors, etc...)

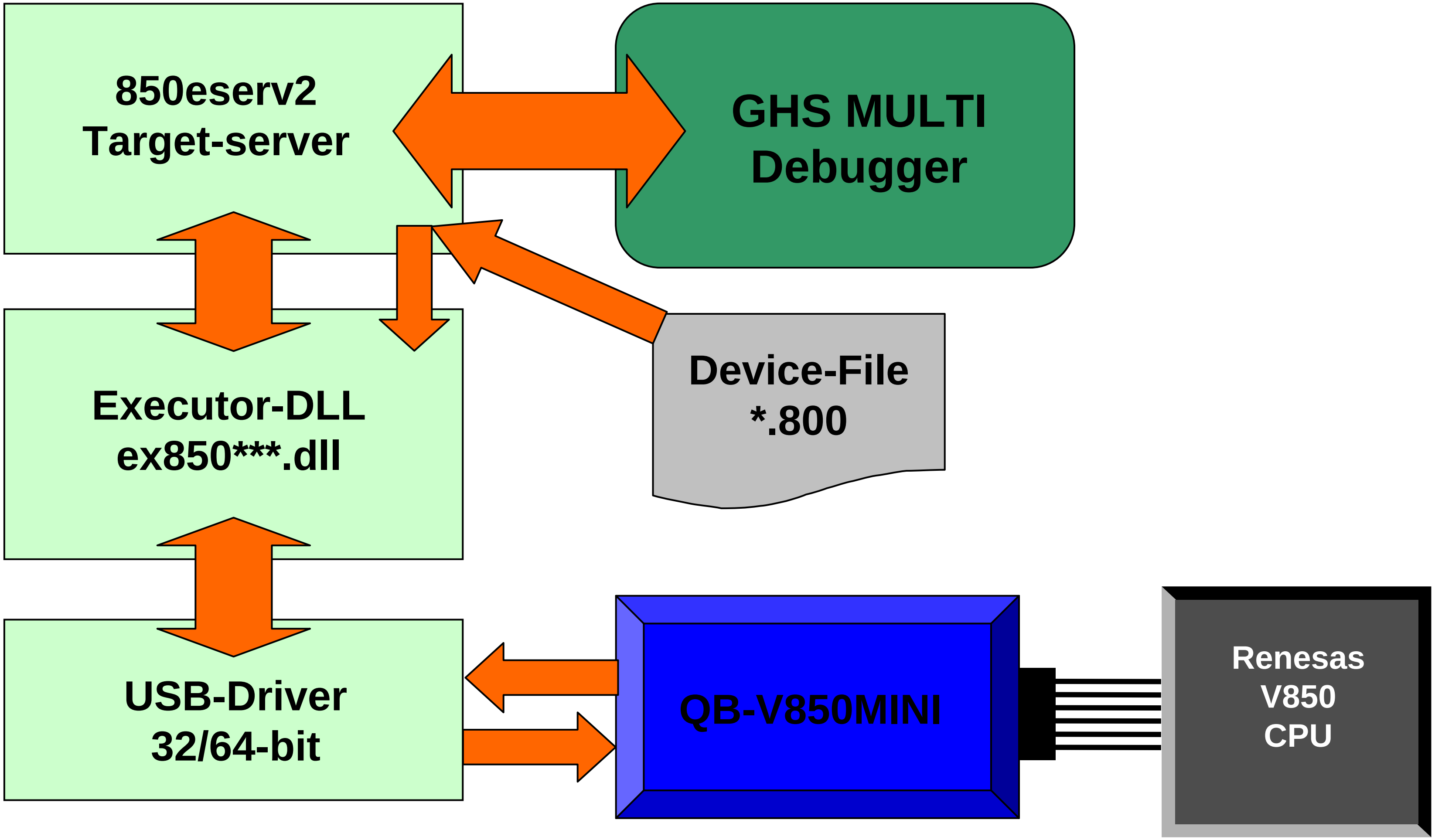
- The connection to the target hardware will be handled via a a separate target server. For V850 and all Renensas in-house debugging products (e.g. MINICUBE, IECUBE, E1/E20) this server is called 850eserv.

- The 850eserv fully controls the communication between the GHS MULTI debugger and target hardware. It also provides additional features which are not covered by the debugger, e.g. access to Dataflash of some V850 derivates, setting of hardware breakpoints and events etc...

- How does everything fit together?
  Let‘s see on the next slide.......

GHS MULTI Debugger Concept

- How does everything fit together???



850eserv2 target server requirements and features

- The 850eserv2 target server is the successor of the former 850eserv target server, adding new features and better integaration into the GHS MULTI GUI

- 850eserv2 requires GHS MULTI version **4.2.4** or later

- V850 specific debugging features supported by 850eserv2:
  - Events
  - Hardware breakpoints
  - Dataflash access and mofication
  - Supports GHS SuperTrace Probe
  - Supports RealTime RAM monitoring feature of QB-V850MINI
  - Hot-Plugin support (V850E2 only)

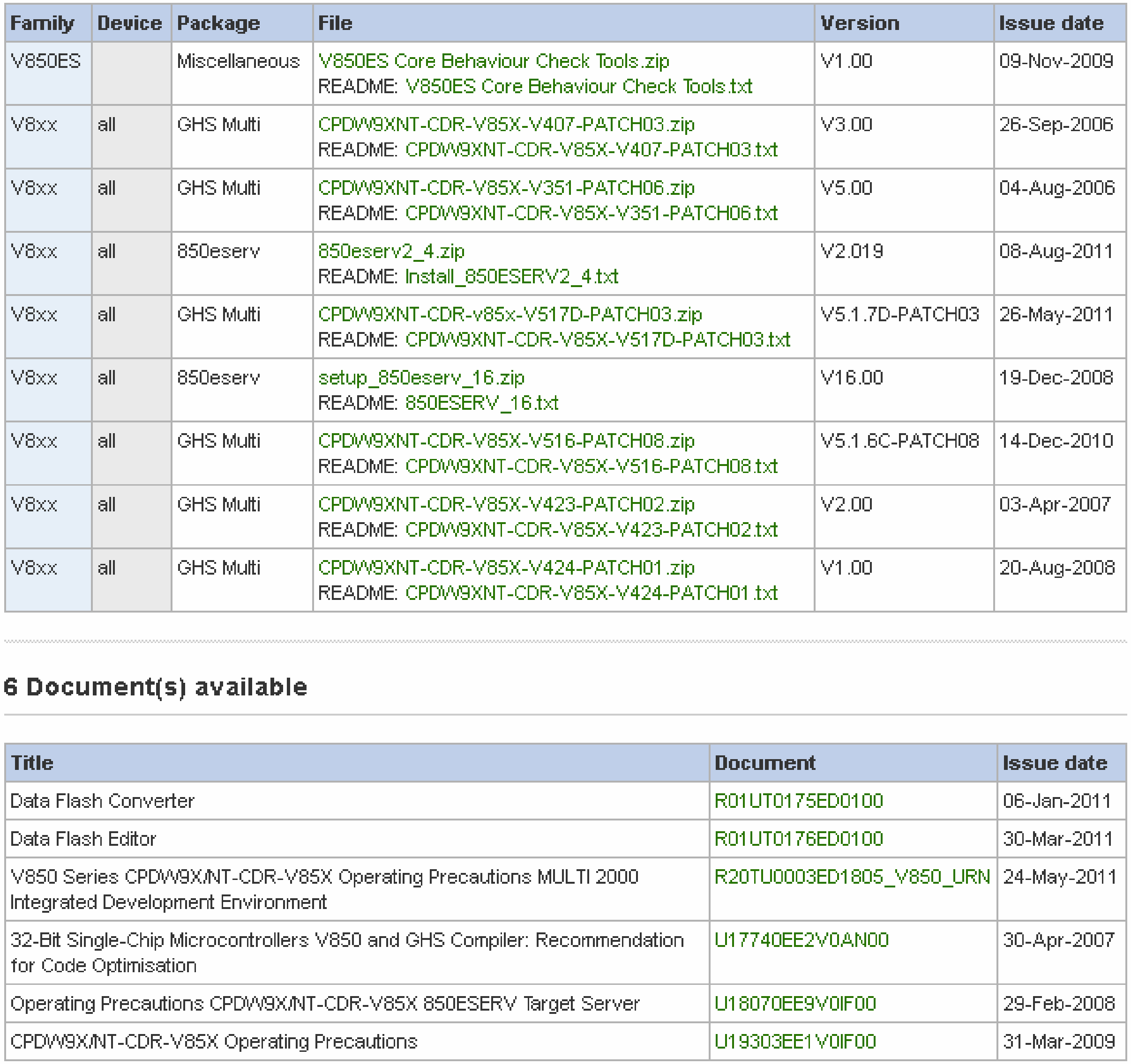
850eserv2 target server package download

- The 850eserv2 target server package should be always updated to the latest version. If there is already any existing in the GHS installation, it should be updated as well

- Latest REE 850eserv2 update package can be downloaded from REE Toolweb download section
  - Contact Renesas for update http://www.renesas.eu/updates Select CPDW9x/NT-CDR-V85X and load latest available 850eserv2 package
  - Direct link http://www2.renesas.eu/updates?id=26

- The package consists of
  - current 850eserv2 target server
  - latest EXEC-DLL's
  - latest USB-drivers (32/64-bit) for Renesas debug tools

850eserv2 target server package download



Family	Device	Package	File	Version	Issue date
V850ES		Miscellaneous	V850ES Core Behaviour Check Tools.zip README: V850ES Core Behaviour Check Tools.txt	V1.00	09-Nov-2009
V8xx	all	GHS Multi	CPDW9XNT-CDR-V85X-V407-PATCH03.zip README: CPDW9XNT-CDR-V85X-V407-PATCH03.txt	V3.00	26-Sep-2006
V8xx	all	GHS Multi	CPDW9XNT-CDR-V85X-V351-PATCH06.zip README: CPDW9XNT-CDR-V85X-V351-PATCH06.txt	V5.00	04-Aug-2006
V8xx	all	850eserv	850eserv2_4.zip README: Install_850ESERV2_4.txt	V2.019	08-Aug-2011
V8xx	all	GHS Multi	CPDW9XNT-CDR-v85x-V517D-PATCH03.zip README: CPDW9XNT-CDR-V85X-V517D-PATCH03.txt	V5.1.7D-PATCH03	26-May-2011
V8xx	all	850eserv	setup_850eserv_16.zip README: 850ESERV_16.txt	V16.00	19-Dec-2008
V8xx	all	GHS Multi	CPDW9XNT-CDR-V85X-V516-PATCH08.zip README: CPDW9XNT-CDR-V85X-V516-PATCH08.txt	V5.1.6C-PATCH08	14-Dec-2010
V8xx	all	GHS Multi	CPDW9XNT-CDR-V85X-V423-PATCH02.zip README: CPDW9XNT-CDR-V85X-V423-PATCH02.txt	V2.00	03-Apr-2007
V8xx	all	GHS Multi	CPDW9XNT-CDR-V85X-V424-PATCH01.zip README: CPDW9XNT-CDR-V85X-V424-PATCH01.txt	V1.00	20-Aug-2008

6 Document(s) available

Title	Document	Issue date
Data Flash Converter	R01UT0175ED0100	06-Jan-2011
Data Flash Editor	R01UT0176ED0100	30-Mar-2011
V850 Series CPDW9X/NT-CDR-V85X Operating Precautions MULTI 2000 Integrated Development Environment	R20TU0003ED1805_V850_URN	24-May-2011
32-Bit Single-Chip Microcontrollers V850 and GHS Compiler: Recommendation for Code Optimisation	U17740EE2V0AN00	30-Apr-2007
Operating Precautions CPDW9X/NT-CDR-V85X 850ESERV Target Server	U18070EE9V0IF00	29-Feb-2008
CPDW9X/NT-CDR-V85X Operating Precautions	U19303EE1V0IF00	31-Mar-2009

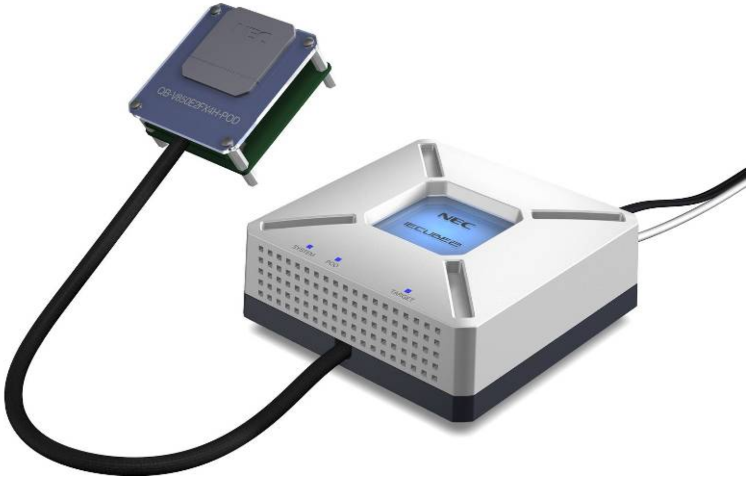
V850 device file packages

- The device-files for a particular device are part of the REE device file package which also includes IO-Headers, Startup-Files, Linker-Directive Files, GRD-Files, etc…

- Renesas publishes always the latest available device specific packages on http://www.renesas.eu/update
  Package for a specific device or family can be found by the Device/Parameter file finder at the bottom of the package

- The device file (extension .*800) is a binary file created by the responsible device developers and contains all neccessary information e.g. memory information, SFR information, parameters for flash download, etc…

Supported Emulators and On-Chip Debuggers



Integration of the new Renesas E1/E20 OCD

- From version **V2.019** onwards 850eserv2 also officialy supports the Renesas E1 and E20 On-Chip debuggers. For this reason some new server starting options were introduced (see also sv-v850e2-us-86.pdf, Chapter 6)

New Options:

-e1jtag	Connects to E1 Emulator JTAG
-e20jtag	Connects to E20 Emulator JTAG
-e1serial	Connects to E1 Emulator Serial
-e20serial	Connects to E20 Emulator Serial

Example:

```
connect 850eserv2 –e1jtag –e2 -id ffffffffffffffffffff ………
```

V850 On-Chip Debugging Emulator function comparison

		MINICUBE2	V850MINIL	E1
Debug support		V850, 78K	V850	V850, **78K, RL78**
On-board Flash programming function		Yes	Yes (via eFLASHLOAD)	**Yes**
Required resources on target MCU	Pins	Serial Interface:	JTAG: 5 I/O pins	JTAG: 5 I/O pins
		via UART - 2 I/O pins		
		or via CSI - 4 I/O pins		
	ROM	2 KBytes + 16 Bytes	-	-
	RAM	16 Bytes	-	-
	Others	Modification of Reset-Vector	-	-
NEXUS compliant debug interface		No	Yes (V850E2 only)	Yes (V850E2 only)
Breakpoints	HW	2	2	4 (V850E2 only)
	SW	4	4	8 (V850E2 only)
Execution time measurement function	MIN	100 us	100 ns	100 ns
	MAX	approx. 100 hours	3.5 minutes	3.5 minutes
Real-time Trace		No	No	No
Real-time RAM monitoring function		Yes	Yes	Yes
Debugging of selfprogramming		No	Possible	Possible
Hot plug-in function		No	No	**Yes**
Power supply function		Yes (3.1V or 5V)	No	**Yes (3.3V or 5V)**
	MAX	100 mA max.	-	**200 mA max.**
User System Interface Conector		16-pin 2.54mm pitch	20-pin 2.54mm pitch	14-pin 2.54mm pitch
Sales Price		Low (75€)	High (400€)	**Low (100€)**

Connection methods

Connecting To A Target

- Manually via Multi command line
  - The connection is manually typed in
    Example of connection to the instruction set simulator

    ```
    MULTI> connect sim850 –cpu=v850e –rom
    ```

  - Example of connection to standard FX3 Minicube

    ```
    MULTI> connect 850eserv2 -minicube -id ffffffffffffffffffff
    -noiop    -df=DF3371.800 -ip=C:\ghs\v800517d\v850e
    ```

- Target Connections shall be automatically established, either via
  - Startup Script
  - Connection Manager

Connecting To A Target

- Using a startup Script, then the script shall have the same name as the application built by Multi with extension ’.rc’.

- Example 1:

Project is:	DriveMe.gpj
Output file is:	DriveMe
RC Filename:	DriveMe.rc

- Example 2:

Project is:	DriveMe.gpj
Output file is:	DriveMe.out
RC Filename:	DriveMe.out.rc

- Example 2:

Project is:	DriveMe.gpj
Output file is:	DriveMe.abs
RC Filename:	DriveMe.abs.rc

Connecting To A Target

- Connection examples

  - Example 1 (straight forward Minicube)

```
connect 850eserv2 -minicube -id ffffffffffffffffffff –noiop -df=DF3371.800
-ip=C:\ghs\v800517d\v850e
target dclock 6000,0,swoff
```

  - Example 2 (Single Line IECUBE)

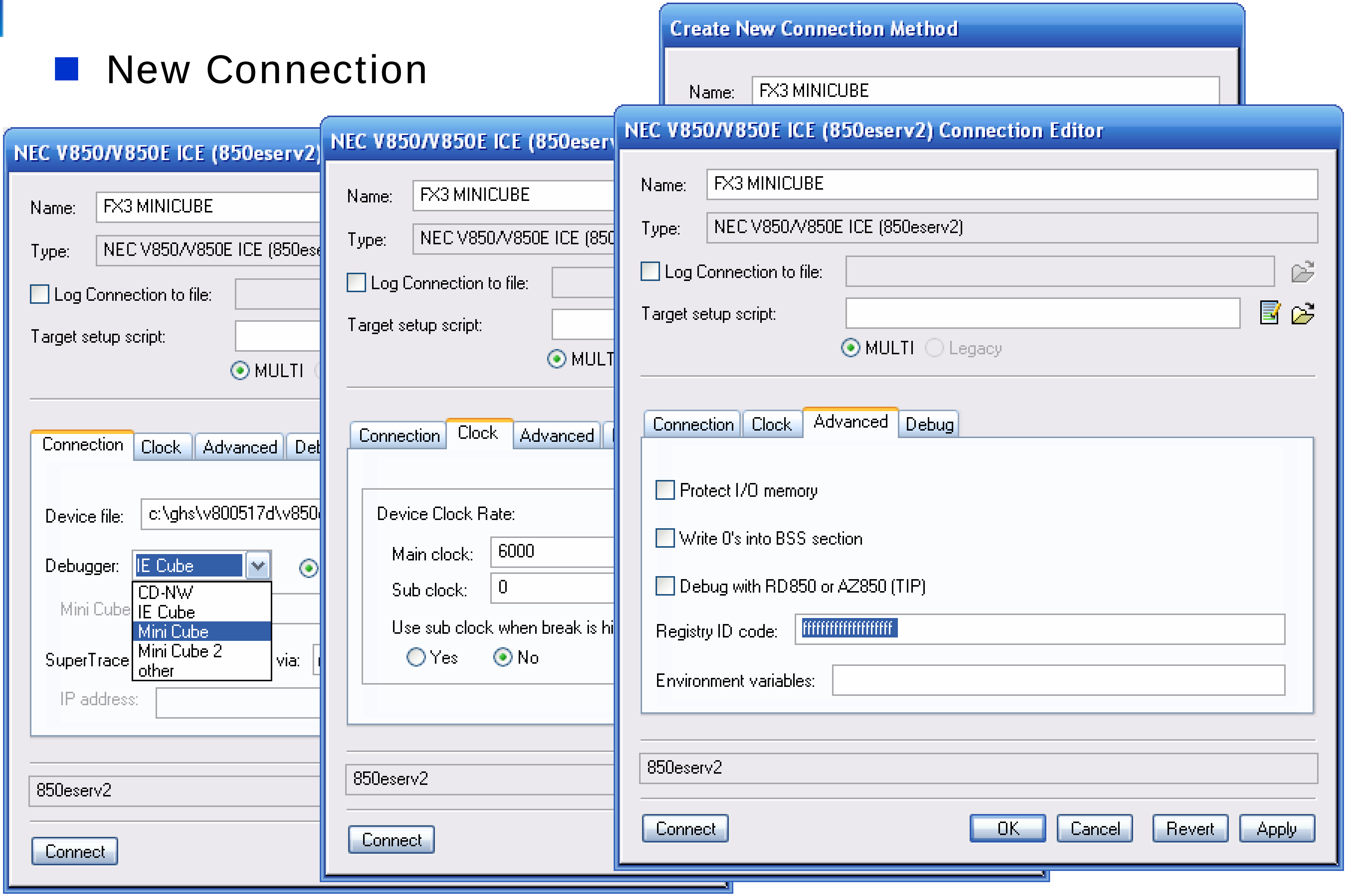
```
connect 850eserv2 -iecube -id ffffffffffffffffffff –noiop -df=DF3371.800 -
ip=C:\ghs\v800517d\v850e –dclock=6000,0,swoff
```

  - Example 2 (Conditional connection)

```
if( _REMOTE_CONNECTED==0) {
connect 850eserv2 -minicube -id ffffffffffffffffffff –noiop -df=DF3371.800
-ip=C:\ghs\v800517d\v850e –dclock=6000,0,swoff
}
target rst
```

Connection Manager

- New Connection



Create New Connection Method

Name: FX3 MINICUBE

NEC V850/V850E ICE (850eserv2)

Name: FX3 MINICUBE

Type: NEC V850/V850E ICE (850ese

□ Log Connection to file:

Target setup script:

● MULTI

Connection | Clock | Advanced | Deb

Device file: c:\ghs\v800517d\v850

Debugger: IE Cube

- CD-NW
- IE Cube
- Mini Cube
- Mini Cube 2
- other

Mini Cube

SuperTrace

via:

IP address:

850eserv2

Connect

NEC V850/V850E ICE (850eserv

Name: FX3 MINICUBE

Type: NEC V850/V850E ICE (850

□ Log Connection to file:

Target setup script:

● MULT

Connection | Clock | Advanced

Device Clock Rate:

Main clock: 6000

Sub clock: 0

Use sub clock when break is hi

○ Yes ● No

850eserv2

Connect

NEC V850/V850E ICE (850eserv2) Connection Editor

Name: FX3 MINICUBE

Type: NEC V850/V850E ICE (850eserv2)

□ Log Connection to file:

Target setup script:

● MULTI ○ Legacy

Connection | Clock | Advanced | Debug

□ Protect I/O memory

□ Write 0's into BSS section

□ Debug with RD850 or AZ850 (TIP)

Registry ID code: ffffffffffffffffffff

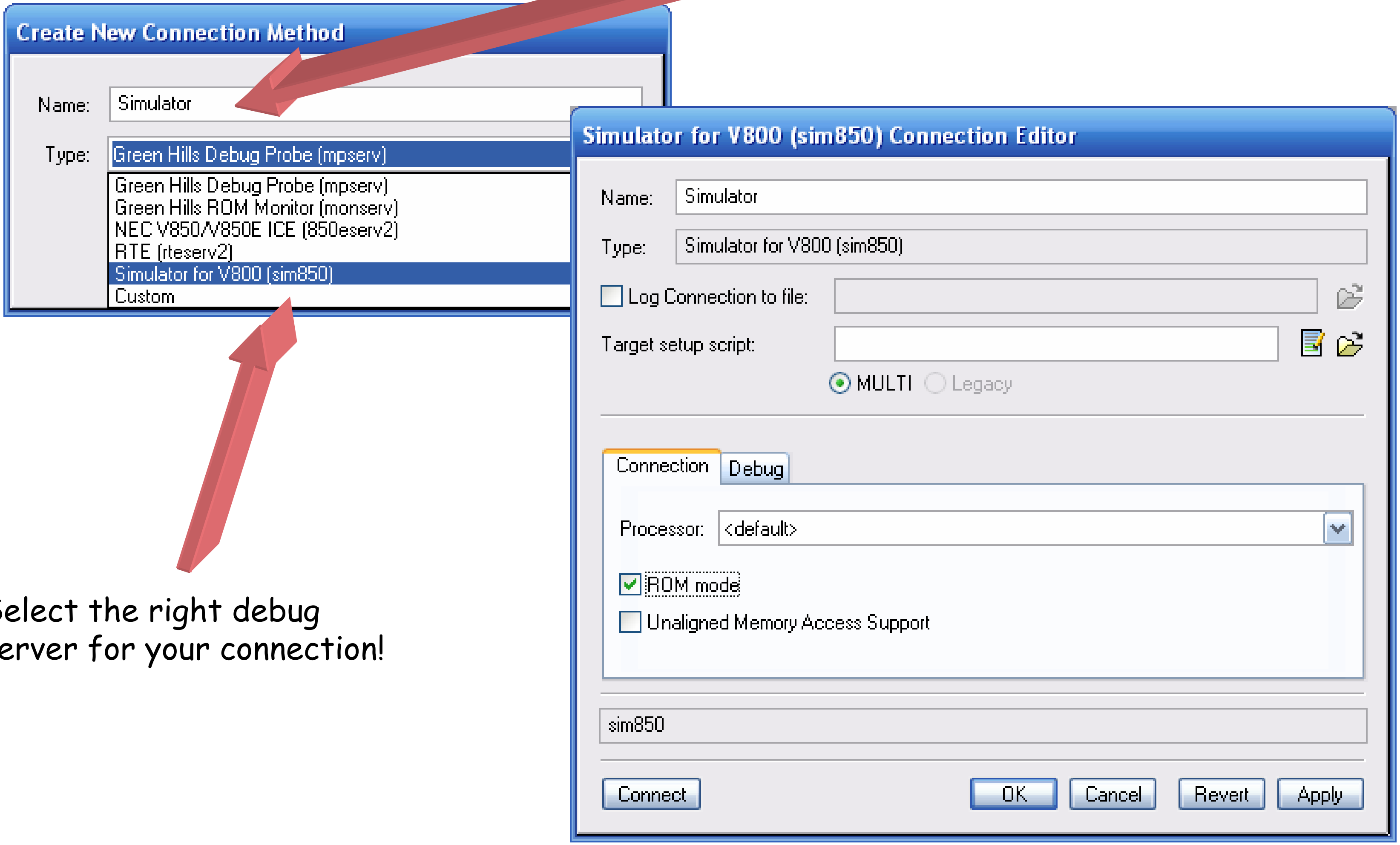
Environment variables:

850eserv2

Connect | OK | Cancel | Revert | Apply

Connection Manager

Connection List



Create New Connection Method

Name: Simulator

Type: Green Hills Debug Probe (mpserv)

- Green Hills Debug Probe (mpserv)
- Green Hills ROM Monitor (monserv)
- NEC V850/V850E ICE (850eserv2)
- RTE (rteserv2)
- Simulator for V800 (sim850)
- Custom

Simulator for V800 (sim850) Connection Editor

Name: Simulator

Type: Simulator for V800 (sim850)

□ Log Connection to file:

Target setup script:

● MULTI ○ Legacy

Connection | Debug

Processor: <default>

☑ ROM mode

□ Unaligned Memory Access Support

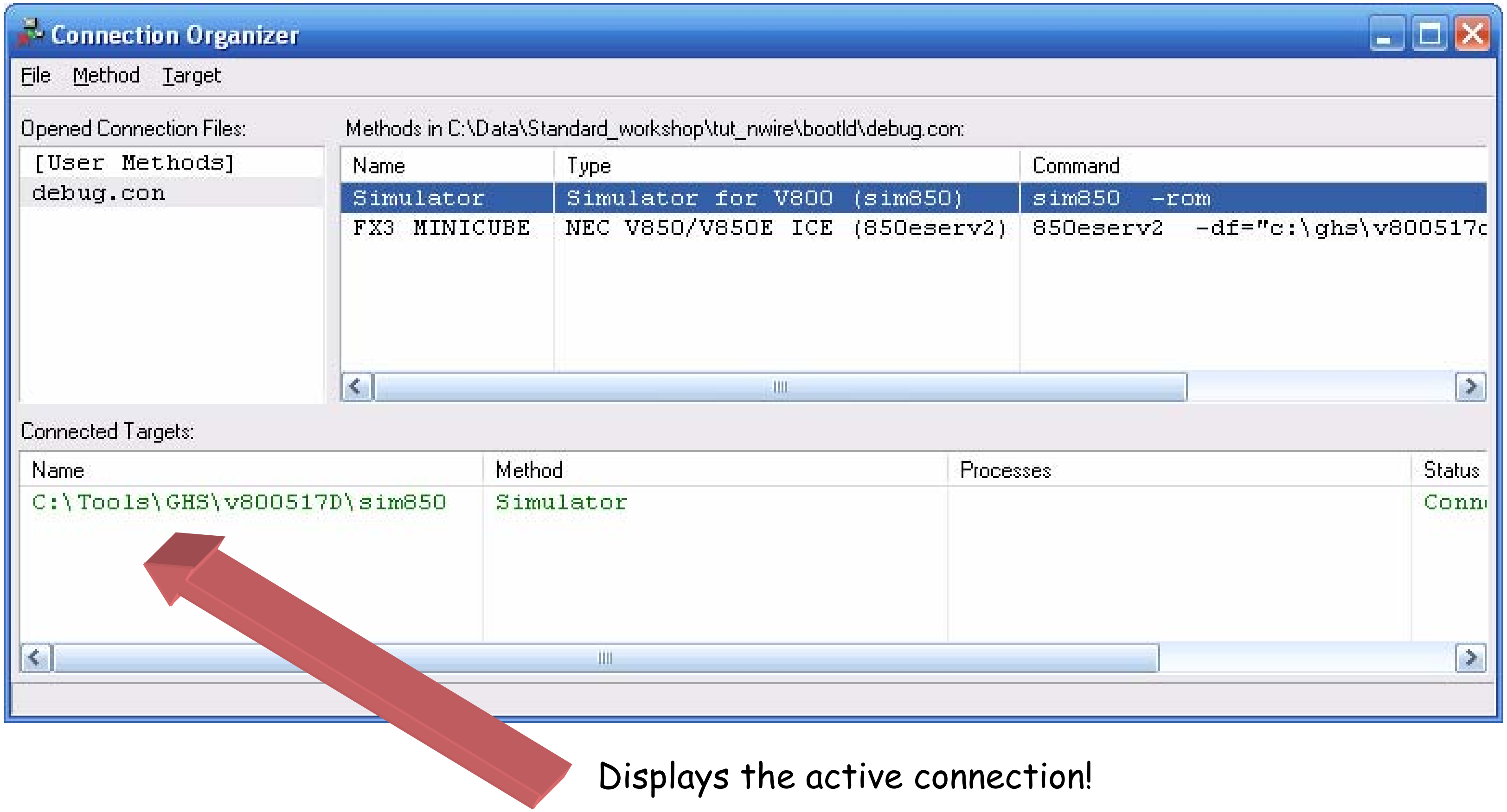
sim850

Select a name for your connection!

Select the right debug server for your connection!

Connection Manager

- Connection Manager with multiple choices



Opened Connection Files:

[User Methods]
debug.con

Methods in C:\Data\Standard_workshop\tut_nwire\bootld\debug.con:

Name	Type	Command
Simulator	Simulator for V800 (sim850)	sim850 -rom
FX3 MINICUBE	NEC V850/V850E ICE (850eserv2)	850eserv2 -df="c:\ghs\v800517c

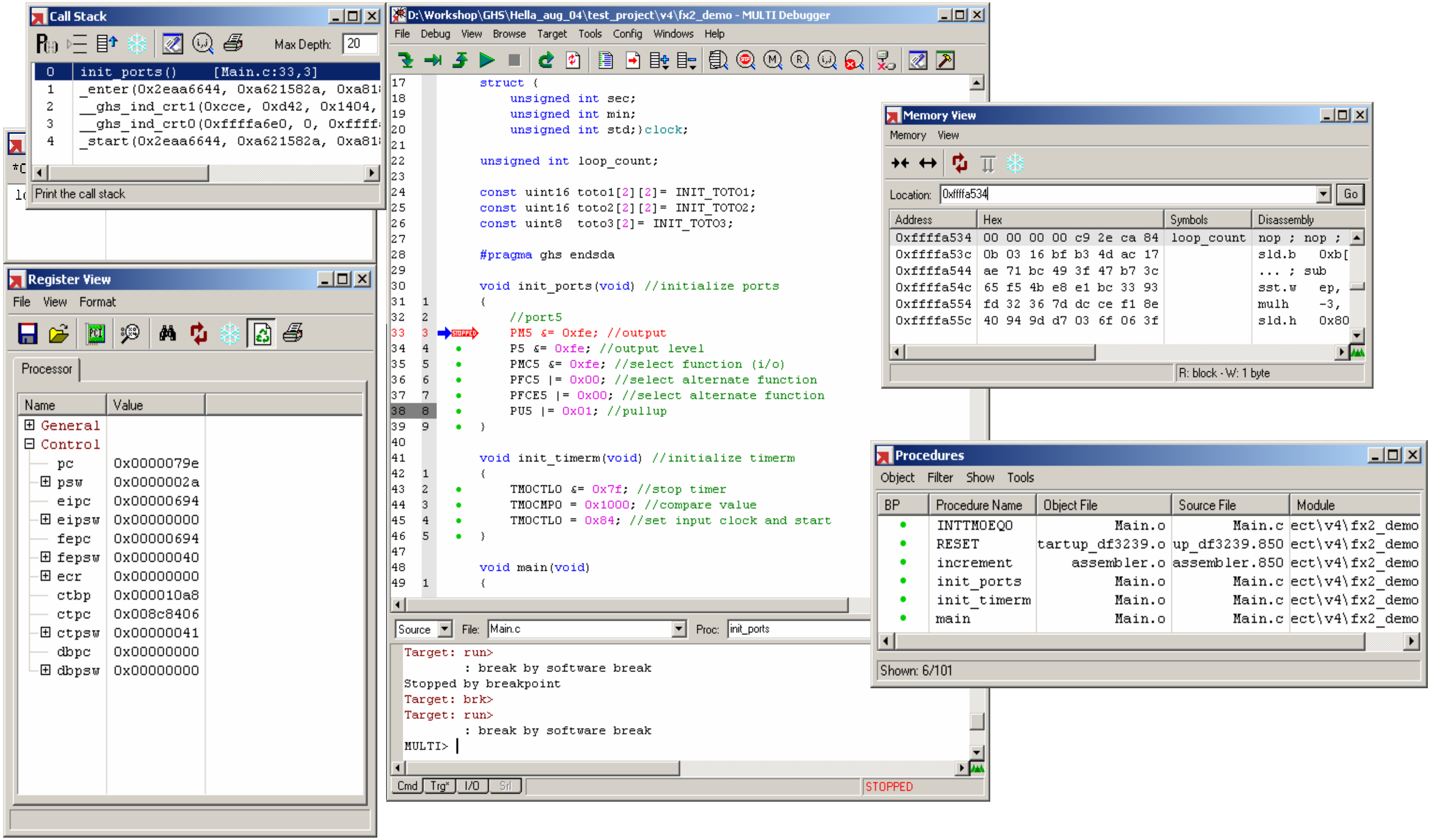
Connected Targets:

Name	Method	Processes	Status
C:\Tools\GHS\v800517D\sim850	Simulator		Conn

Displays the active connection!

Debugger features

The MULTI Debugger



```
0   init_ports()        [Main.c:33,3]
1   _enter(0x2eaa6644, 0xa621582a, 0xa81
2   __ghs_ind_crt1(0xcce, 0xd42, 0x1404,
3   __ghs_ind_crt0(0xffffa6e0, 0, 0xffff
4   _start(0x2eaa6644, 0xa621582a, 0xa81
```

```
        struct {
            unsigned int sec;
            unsigned int min;
            unsigned int std;}clock;

        unsigned int loop_count;

        const uint16 toto1[2][2]= INIT_TOTO1;
        const uint16 toto2[2][2]= INIT_TOTO2;
        const uint8  toto3[2]= INIT_TOTO3;

        #pragma ghs endsda

        void init_ports(void) //initialize ports
        {
            //port5
            PM5 &= 0xfe; //output
            P5 &= 0xfe; //output level
            PMC5 &= 0xfe; //select function (i/o)
            PFC5 |= 0x00; //select alternate function
            PFCE5 |= 0x00; //select alternate function
            PU5 |= 0x01; //pullup
        }

        void init_timerm(void) //initialize timerm
        {
            TMOCTL0 &= 0x7f; //stop timer
            TMOCMP0 = 0x1000; //compare value
            TMOCTL0 = 0x84; //set input clock and start
        }

        void main(void)
        {
```

```
Target: run>
          : break by software break
Stopped by breakpoint
Target: brk>
Target: run>
          : break by software break
MULTI>
```

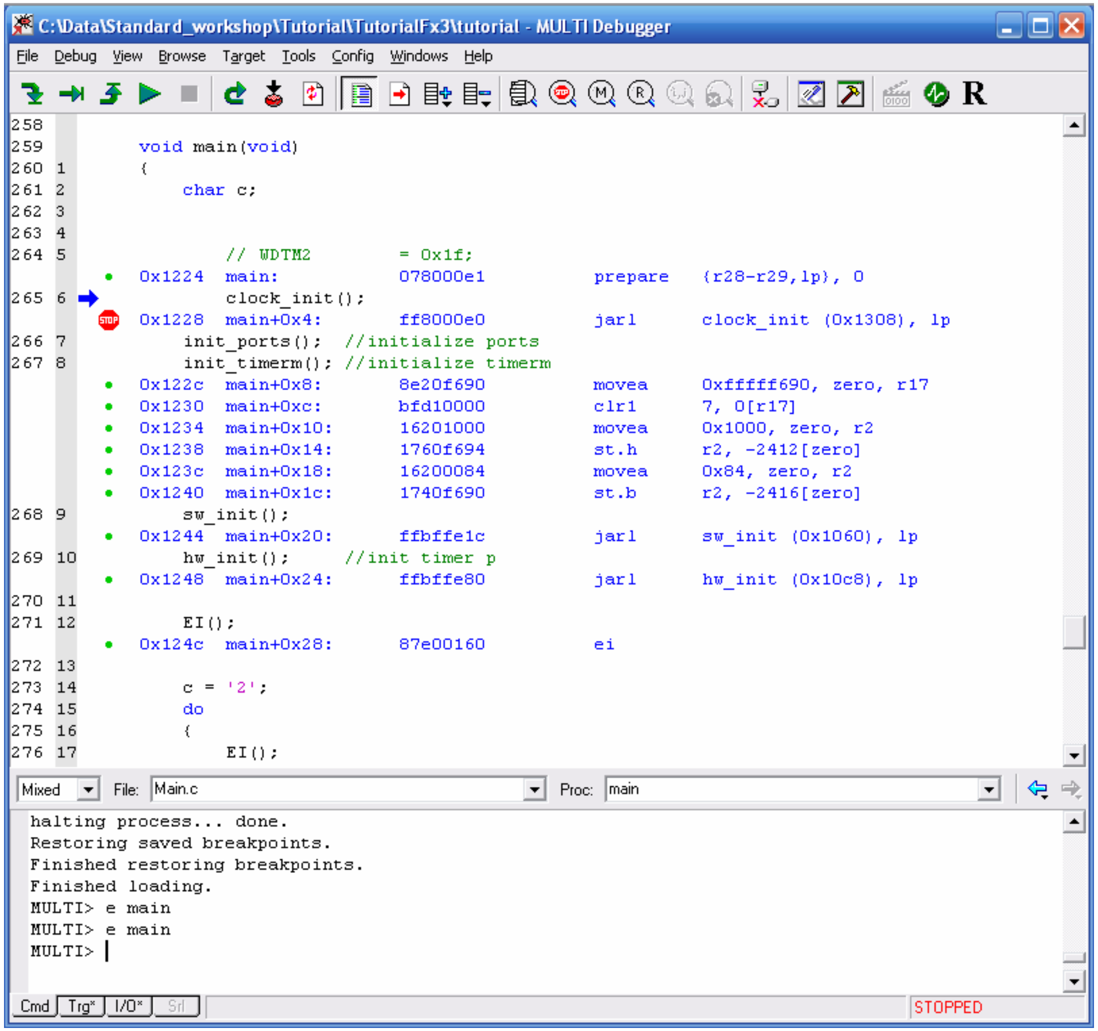
Address	Hex	Symbols	Disassembly
0xffffa534	00 00 00 00 c9 2e ca 84	loop_count	nop ; nop ;
0xffffa53c	0b 03 16 bf b3 4d ac 17		sld.b 0xb[
0xffffa544	ae 71 bc 49 3f 47 b7 3c		... ; sub
0xffffa54c	65 f5 4b e8 e1 bc 33 93		sst.w ep,
0xffffa554	fd 32 36 7d dc ce f1 8e		mulh -3,
0xffffa55c	40 94 9d d7 03 6f 06 3f		sld.h 0x80

BP	Procedure Name	Object File	Source File	Module
●	INTTM0EQ0	Main.o	Main.c	ect\v4\fx2_demo
●	RESET	tartup_df3239.o	up_df3239.850	ect\v4\fx2_demo
●	increment	assembler.o	assembler.850	ect\v4\fx2_demo
●	init_ports	Main.o	Main.c	ect\v4\fx2_demo
●	init_timerm	Main.o	Main.c	ect\v4\fx2_demo
●	main	Main.o	Main.c	ect\v4\fx2_demo

Name	Value
⊞ General	
⊟ Control	
pc	0x0000079e
⊞ psw	0x0000002a
eipc	0x00000694
⊞ eipsw	0x00000000
fepc	0x00000694
⊞ fepsw	0x00000040
⊞ ecr	0x00000000
ctbp	0x000010a8
ctpc	0x008c8406
⊞ ctpsw	0x00000041
dbpc	0x00000000
⊞ dbpsw	0x00000000

The MULTI Debugger' features

- Source code: Mixed, full assembly or only C view


C:\Data\Standard_workshop\Tutorial\TutorialFx3\tutorial - MULTI Debugger
File
Debug
View
Browse
Target
Tools
Config
Windows
Help
R
Cmd
Trg*
I/O*
Srl

```
        void main(void)
        {
            char c;



                // WDTM2          = 0x1f;
0x1224  main:           078000e1        prepare   {r28-r29,lp}, 0
                clock_init();
0x1228  main+0x4:       ff8000e0        jarl      clock_init (0x1308), lp
            init_ports();  //initialize ports
            init_timerm(); //initialize timerm
0x122c  main+0x8:       8e20f690        movea     0xfffff690, zero, r17
0x1230  main+0xc:       bfd10000        clr1      7, 0[r17]
0x1234  main+0x10:      16201000        movea     0x1000, zero, r2
0x1238  main+0x14:      1760f694        st.h      r2, -2412[zero]
0x123c  main+0x18:      16200084        movea     0x84, zero, r2
0x1240  main+0x1c:      1740f690        st.b      r2, -2416[zero]
            sw_init();
0x1244  main+0x20:      ffbffe1c        jarl      sw_init (0x1060), lp
            hw_init();     //init timer p
0x1248  main+0x24:      ffbffe80        jarl      hw_init (0x10c8), lp

            EI();
0x124c  main+0x28:      87e00160        ei

            c = '2';
            do
            {
                EI();
```

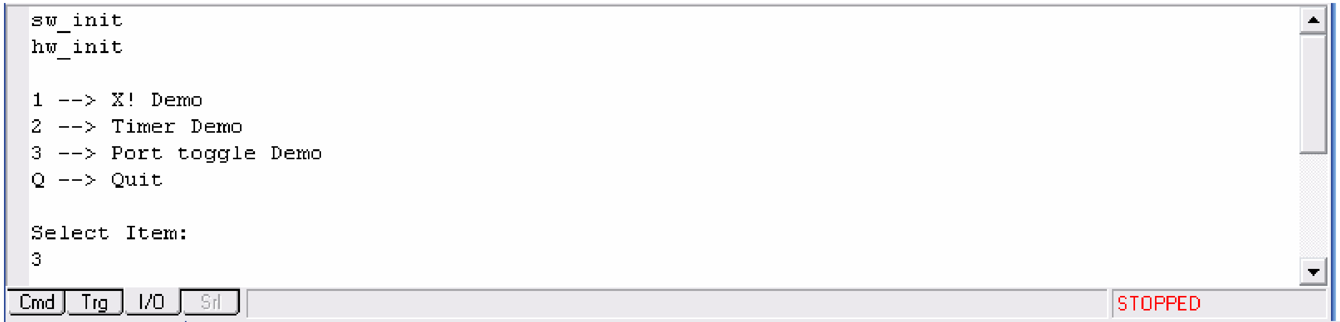
Mixed File: Main.c Proc: main

```
halting process... done.
Restoring saved breakpoints.
Finished restoring breakpoints.
Finished loading.
MULTI> e main
MULTI> e main
MULTI>
```

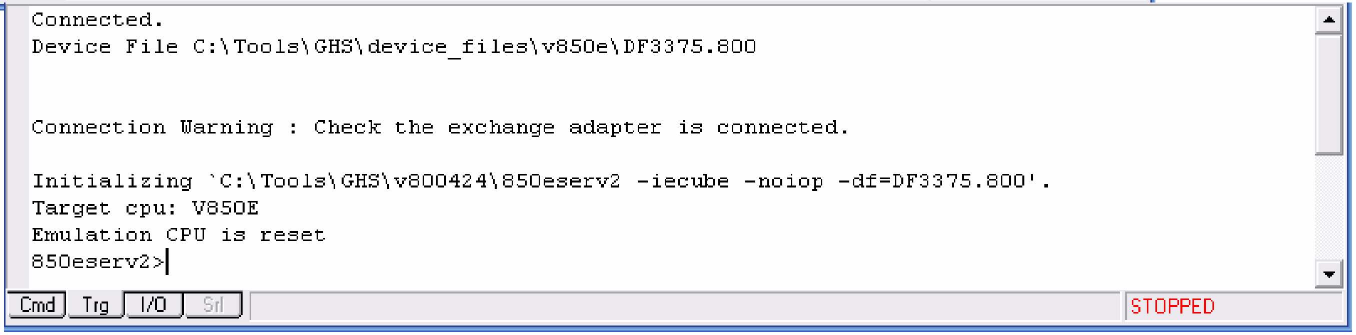
STOPPED

The MULTI Debugger' features

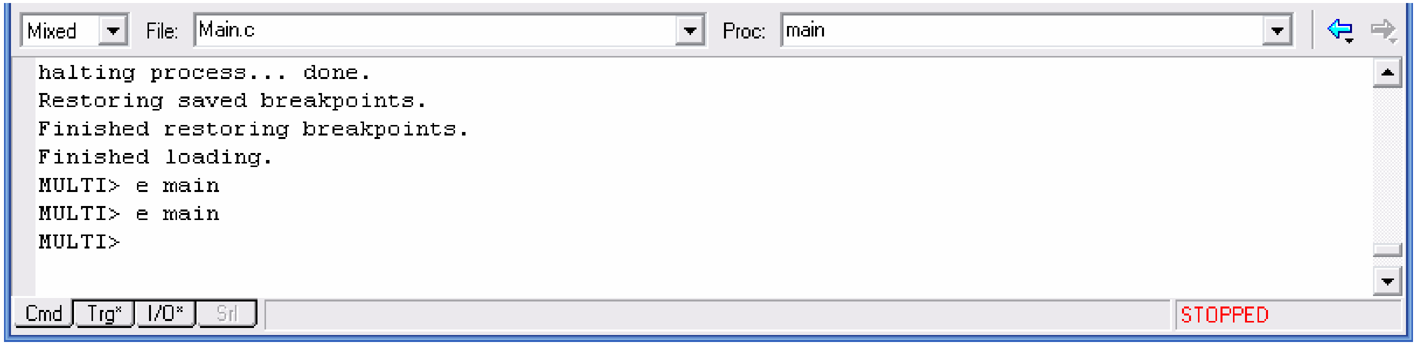
- I/O windows, target windows, debugger command pane window



```
sw_init
hw_init

1 --> X! Demo
2 --> Timer Demo
3 --> Port toggle Demo
Q --> Quit

Select Item:
3
```



```
Connected.
Device File C:\Tools\GHS\device_files\v850e\DF3375.800


Connection Warning : Check the exchange adapter is connected.

Initializing `C:\Tools\GHS\v800424\850eserv2 -iecube -noiop -df=DF3375.800'.
Target cpu: V850E
Emulation CPU is reset
850eserv2>
```

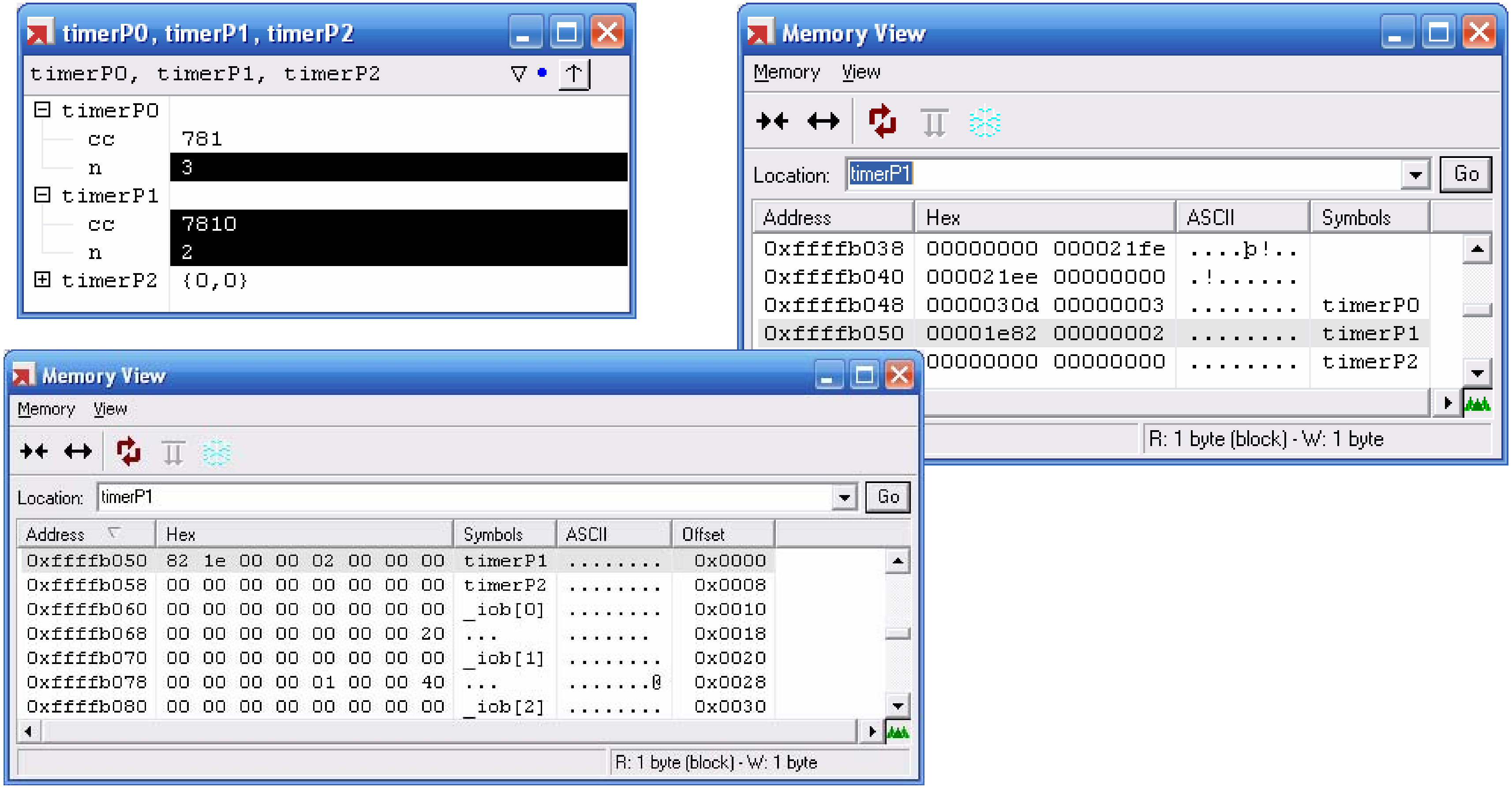


Mixed File: Main.c Proc: main

```
halting process... done.
Restoring saved breakpoints.
Finished restoring breakpoints.
Finished loading.
MULTI> e main
MULTI> e main
MULTI>
```

The MULTI Debugger’ features

- Watch window to view and edit variables, pointer, structures …etc…
- View and edit Memory with the memory view window.



timerP0, timerP1, timerP2

timerP0, timerP1, timerP2

⊟ timerP0	
cc	781
n	3
⊟ timerP1	
cc	7810
n	2
⊞ timerP2	{0,0}

Memory View

Location: timerP1 Go

Address	Hex		ASCII	Symbols
0xffffb038	00000000	000021fe	....þ!..	
0xffffb040	000021ee	00000000	.!......	
0xffffb048	0000030d	00000003	....	timerP0
0xffffb050	00001e82	00000002	....	timerP1
	00000000	00000000	....	timerP2

R: 1 byte (block) - W: 1 byte

Memory View

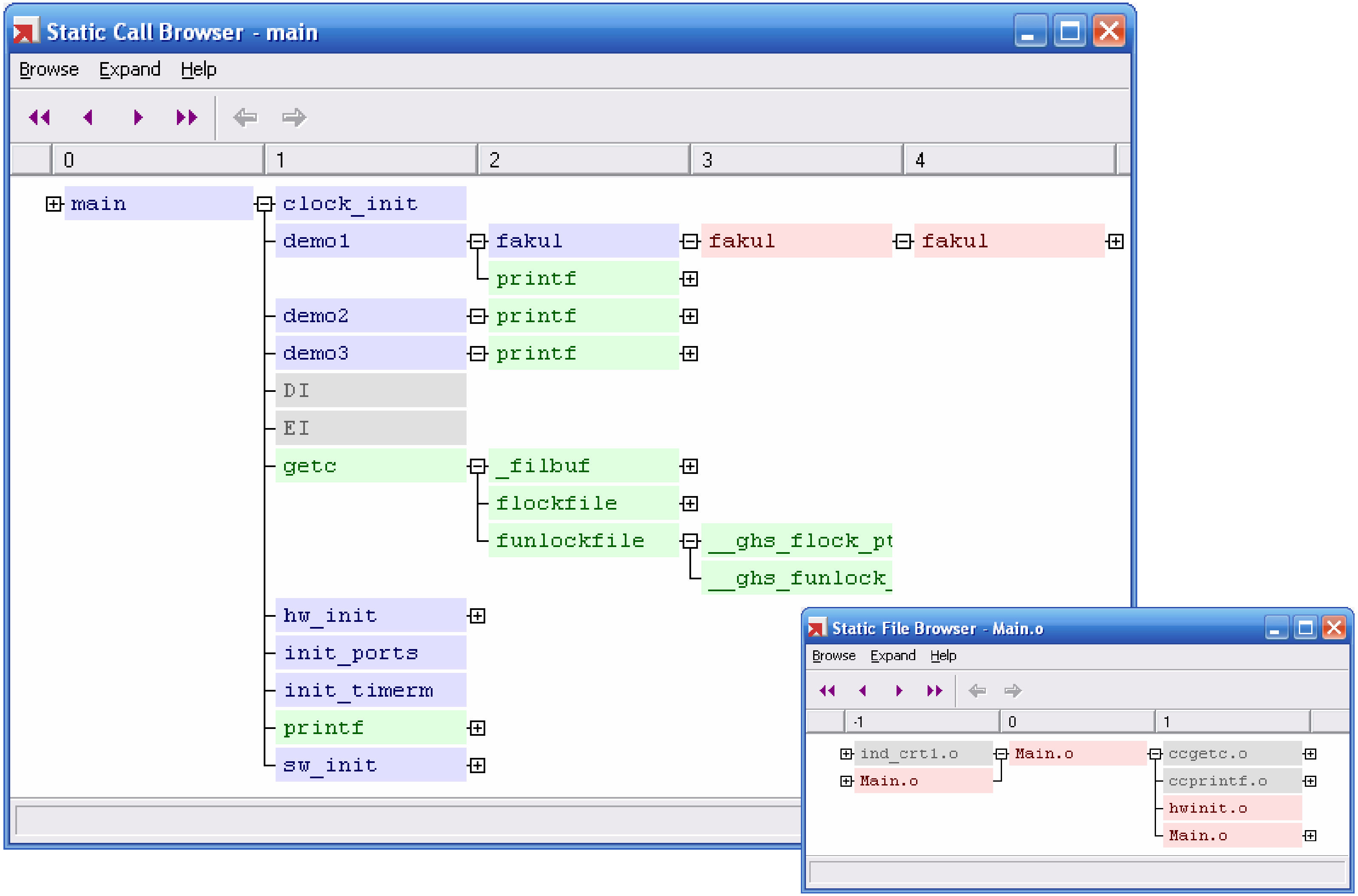
Location: timerP1 Go

Address	Hex	Symbols	ASCII	Offset
0xffffb050	82 1e 00 00 02 00 00 00	timerP1	....	0x0000
0xffffb058	00 00 00 00 00 00 00 00	timerP2	....	0x0008
0xffffb060	00 00 00 00 00 00 00 00	_iob[0]	....	0x0010
0xffffb068	00 00 00 00 00 00 00 20	...	....	0x0018
0xffffb070	00 00 00 00 00 00 00 00	_iob[1]	....	0x0020
0xffffb078	00 00 00 00 01 00 00 40	...	.......@	0x0028
0xffffb080	00 00 00 00 00 00 00 00	_iob[2]	....	0x0030

R: 1 byte (block) - W: 1 byte

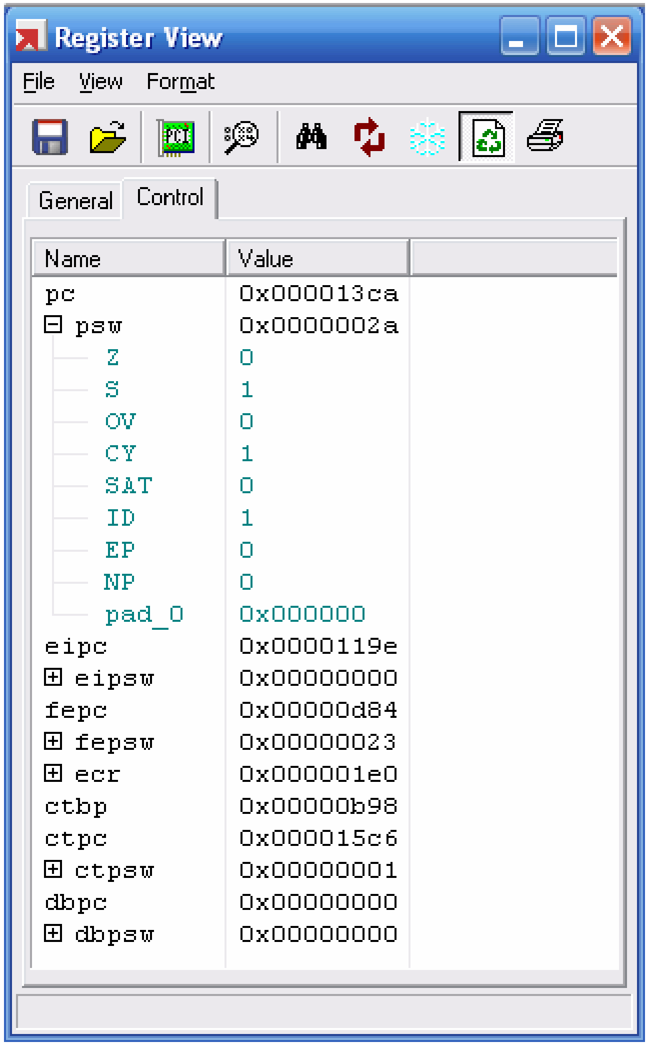
The MULTI Debugger' features

- Browser/graphical view of the calls, include files, procedure…

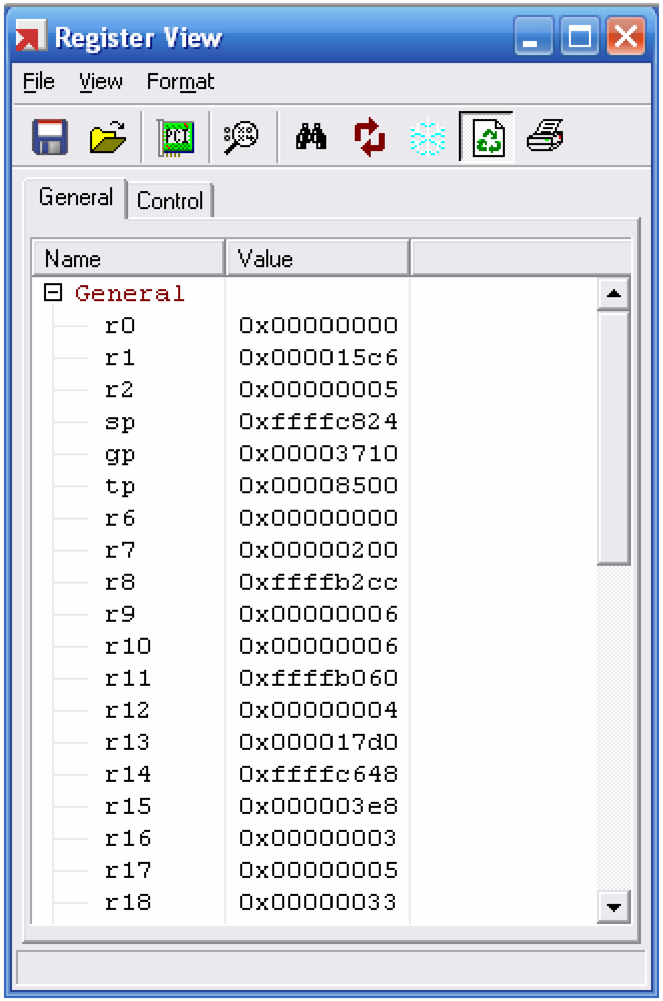


The MULTI Debugger' features

- Register windows. Can be customised.



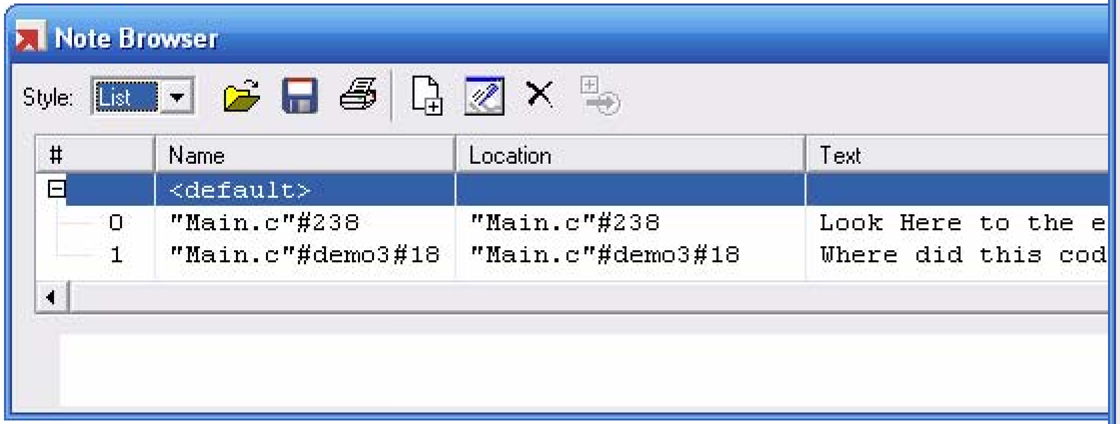
Name	Value
pc	0x000013ca
⊟ psw	0x0000002a
Z	0
S	1
OV	0
CY	1
SAT	0
ID	1
EP	0
NP	0
pad_0	0x000000
eipc	0x0000119e
⊞ eipsw	0x00000000
fepc	0x00000d84
⊞ fepsw	0x00000023
⊞ ecr	0x000001e0
ctbp	0x00000b98
ctpc	0x000015c6
⊞ ctpsw	0x00000001
dbpc	0x00000000
⊞ dbpsw	0x00000000



Name	Value
⊟ General	
r0	0x00000000
r1	0x000015c6
r2	0x00000005
sp	0xffffc824
gp	0x00003710
tp	0x00008500
r6	0x00000000
r7	0x00000200
r8	0xffffb2cc
r9	0x00000006
r10	0x00000006
r11	0xffffb060
r12	0x00000004
r13	0x000017d0
r14	0xffffc648
r15	0x000003e8
r16	0x00000003
r17	0x00000005
r18	0x00000033

The MULTI Debugger' features

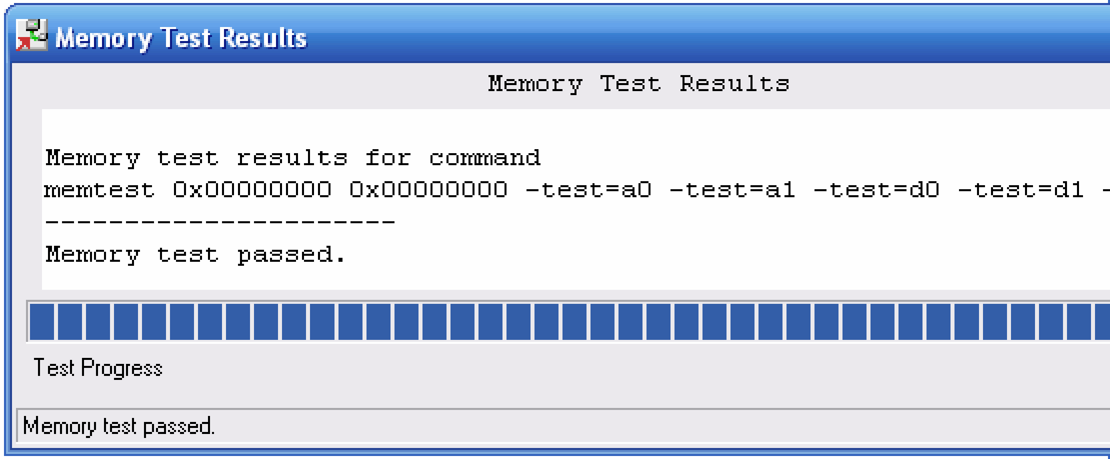
- Debugger Notes


Note Browser

Style: List

#	Name	Location	Text
⊟	<default>		
0	"Main.c"#238	"Main.c"#238	Look Here to the e
1	"Main.c"#demo3#18	"Main.c"#demo3#18	Where did this cod

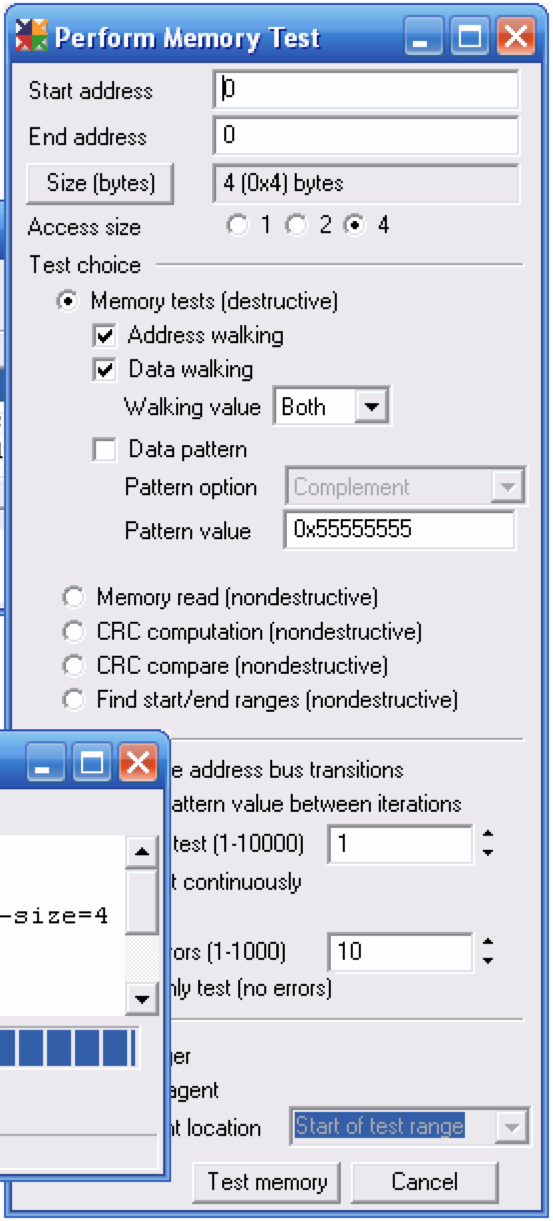
- Memory test tools


Memory Test Results

```
Memory Test Results

Memory test results for command
memtest 0x00000000 0x00000000 -test=a0 -test=a1 -test=d0 -test=d1 -size=4
----------------------
Memory test passed.
```

Test Progress

Memory test passed.


Perform Memory Test

Start address: 0
End address: 0
Size (bytes): 4 (0x4) bytes
Access size: ○ 1 ○ 2 ● 4

Test choice

● Memory tests (destructive)
☑ Address walking
☑ Data walking
Walking value: Both
□ Data pattern
Pattern option: Complement
Pattern value: 0x55555555

○ Memory read (nondestructive)
○ CRC computation (nondestructive)
○ CRC compare (nondestructive)
○ Find start/end ranges (nondestructive)

e address bus transitions
attern value between iterations
test (1-10000): 1
t continuously
ors (1-1000): 10
ly test (no errors)

er
agent
t location: Start of test range

850eserv2 target server features

850eserv2 target server features

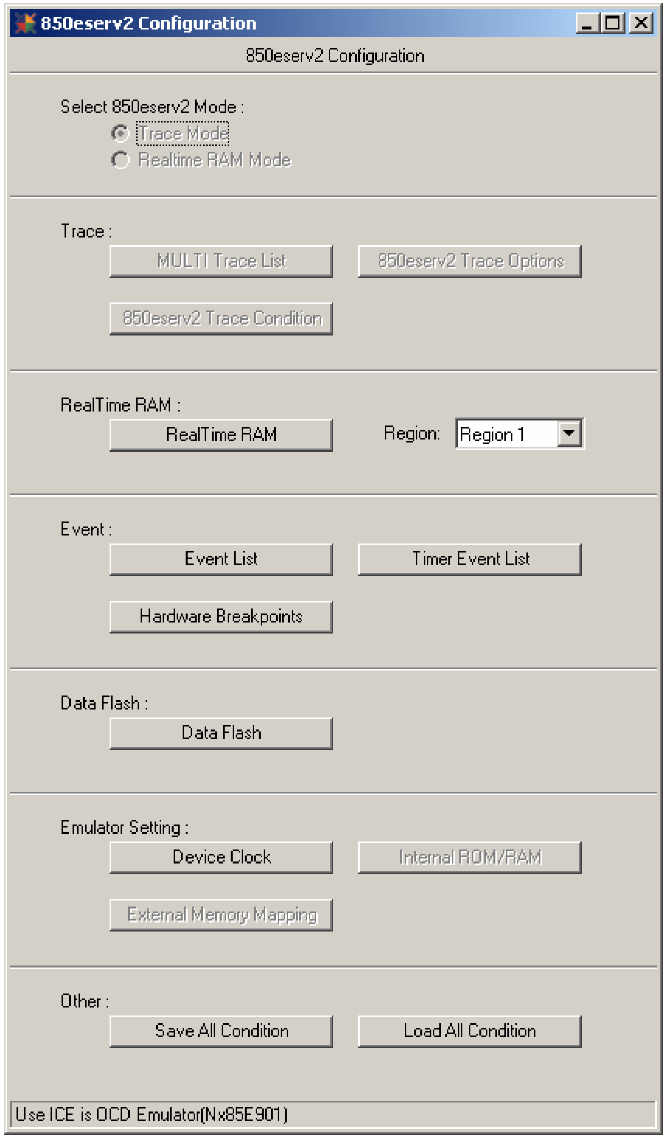
- 850eserv2 supports several V850 specific debbugging features, which are :
  - Events
  - Hardware breakpoints
  - Dataflash access and mofication
  - Supports GHS SuperTrace Probe
  - Supports RealTime RAM monitoring feature of QB-V850MINI
  - Hot-Plugin support (V850E2 only)

- 850eserv2 provides a GUI to setup and access this special features. To open the 850eserv2 configuration window enter following command on MULTI command line:

```
MULTI> target 850win
```

850eserv2 target server features

- 850eserv2 configuration window provides access to following debugging features:
  - Setup and access RealTime RAM functionality of QB-V850MINI
  - Setup and access of Data Flash memory of particular V850 derivates
  - Setup of Events and Hardware Breakpoints



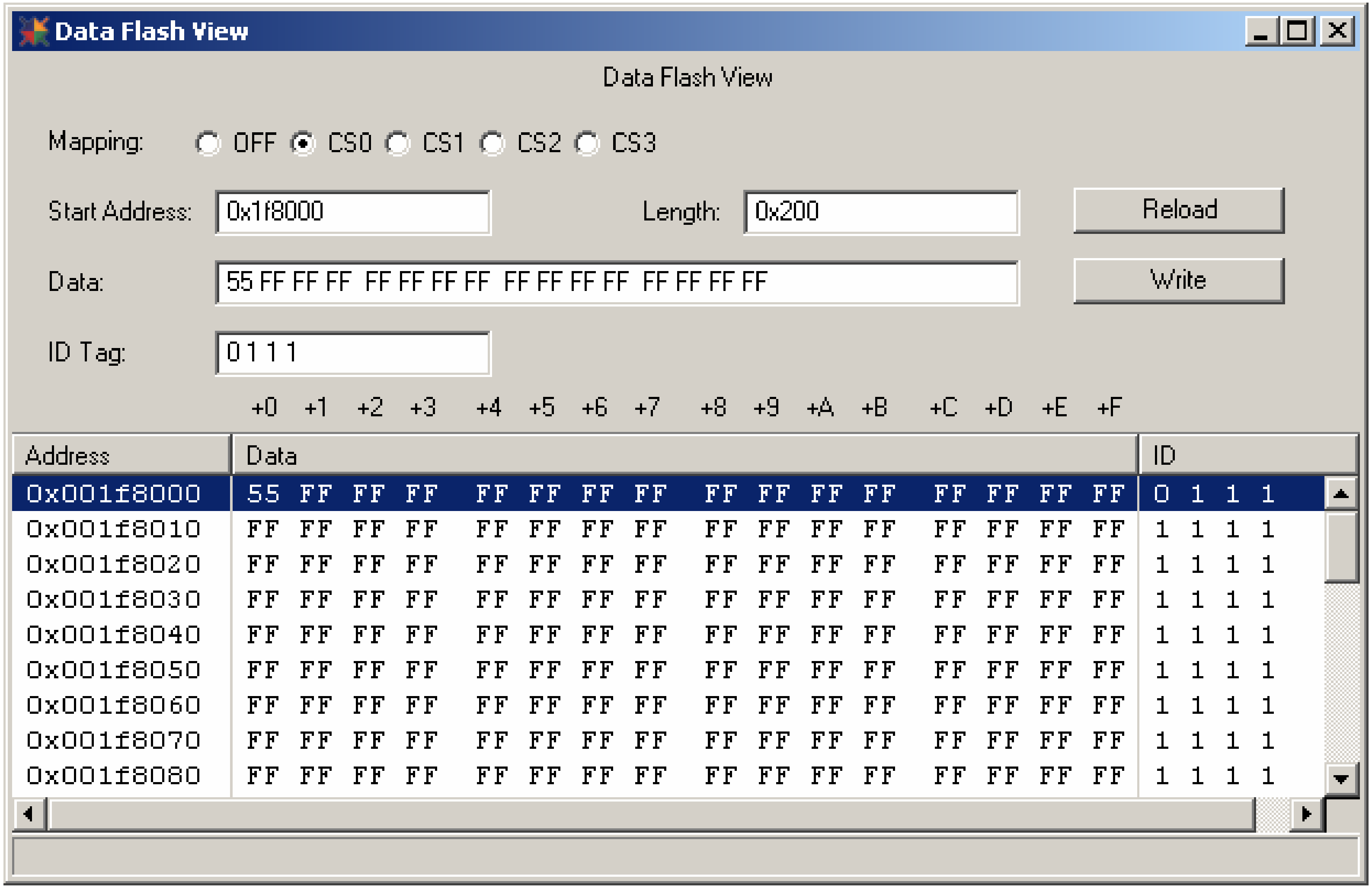
850eserv2 Configuration

Select 850eserv2 Mode :
● Trace Mode
○ Realtime RAM Mode

Region: Region 1

Use ICE is OCD Emulator(Nx85E901)

Data Flash window



Data Flash View

Mapping: ○ OFF ● CS0 ○ CS1 ○ CS2 ○ CS3

Start Address: 0x1f8000

Length: 0x200

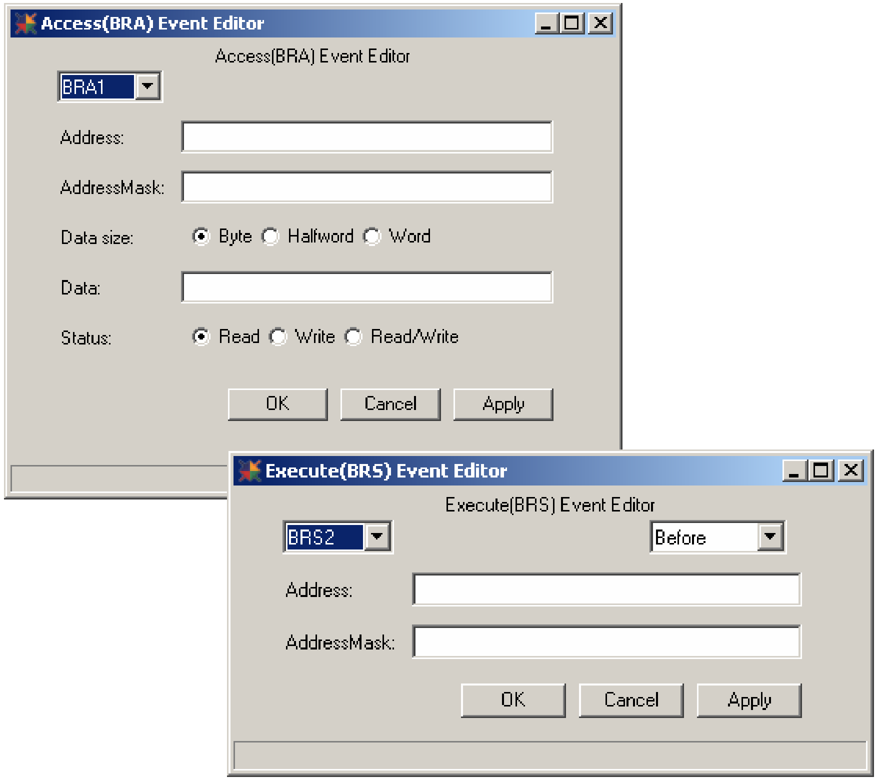
Data: 55 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF

ID Tag: 0 1 1 1

+0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +A +B +C +D +E +F

Address	Data	ID
0x001f8000	55 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF	0 1 1 1
0x001f8010	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF	1 1 1 1
0x001f8020	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF	1 1 1 1
0x001f8030	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF	1 1 1 1
0x001f8040	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF	1 1 1 1
0x001f8050	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF	1 1 1 1
0x001f8060	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF	1 1 1 1
0x001f8070	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF	1 1 1 1
0x001f8080	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF	1 1 1 1

Setup of Events and Hardware Breakpoints



Access(BRA) Event Editor

Access(BRA) Event Editor

BRA1

Address:

AddressMask:

Data size: ● Byte ○ Halfword ○ Word

Data:

Status: ● Read ○ Write ○ Read/Write

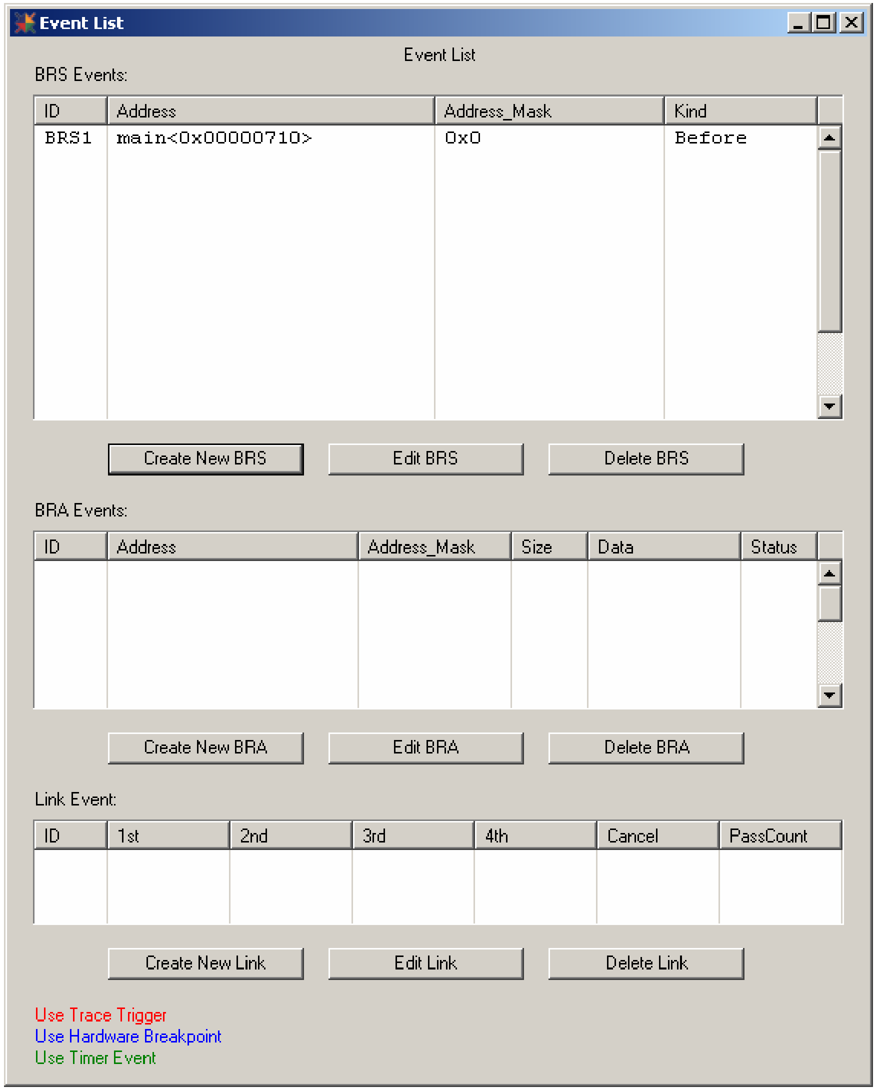
Execute(BRS) Event Editor

Execute(BRS) Event Editor

BRS2 Before

Address:

AddressMask:



Event List

Event List

BRS Events:

ID	Address	Address_Mask	Kind
BRS1	main<0x00000710>	0x0	Before

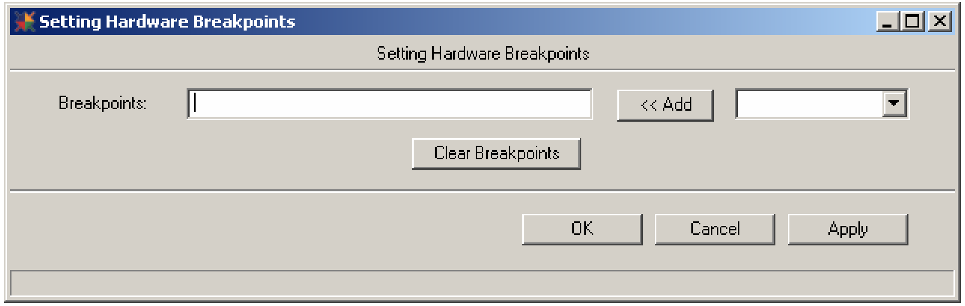
BRA Events:

ID	Address	Address_Mask	Size	Data	Status

Link Event:

ID	1st	2nd	3rd	4th	Cancel	PassCount

Use Trace Trigger
Use Hardware Breakpoint
Use Timer Event

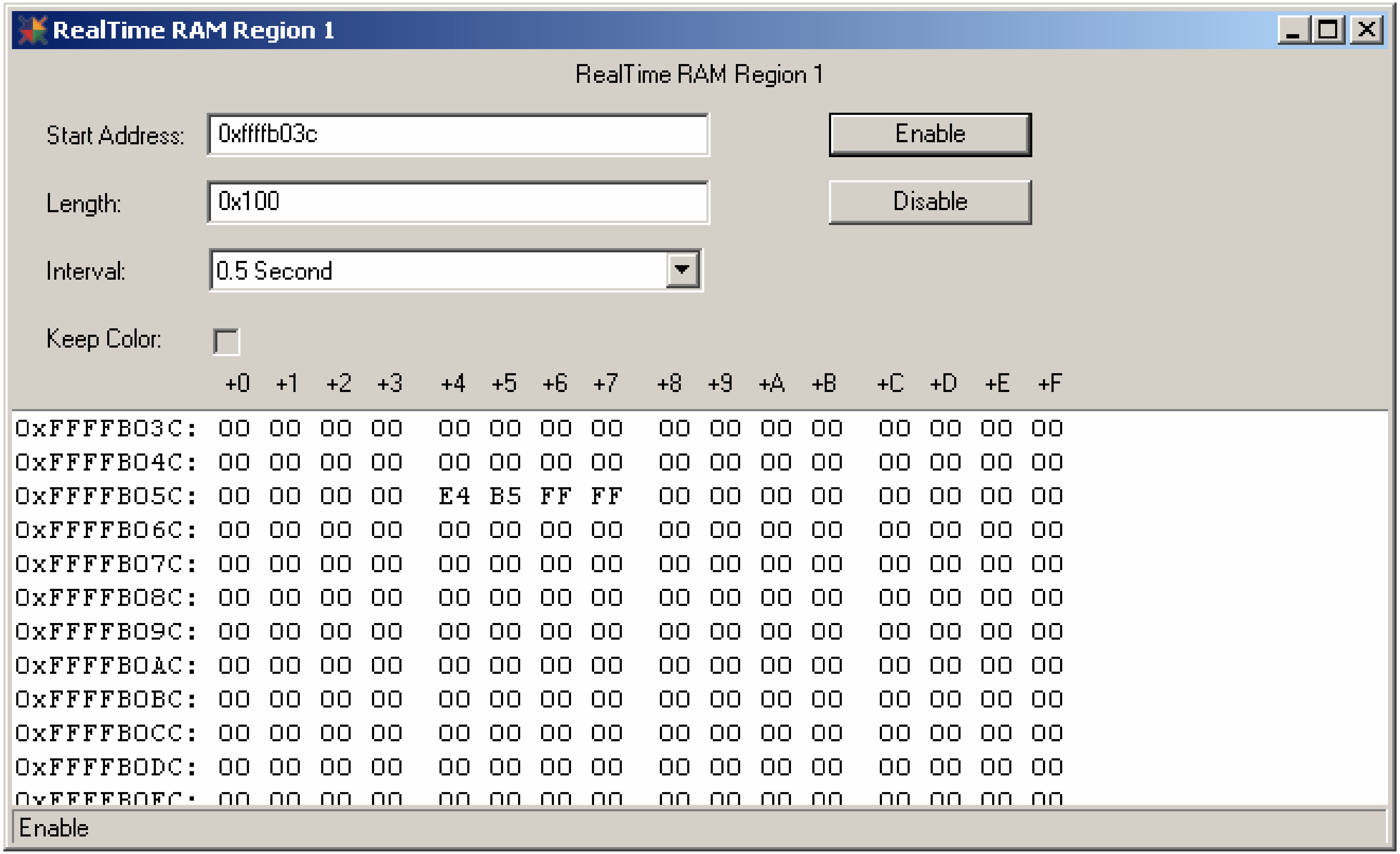


Setting Hardware Breakpoints

Setting Hardware Breakpoints

Breakpoints:

RealTime RAM monitoring feature



RealTime RAM Region 1

Start Address: 0xffffb03c

Length: 0x100

Interval: 0.5 Second

Keep Color: ☐

```
             +0 +1 +2 +3  +4 +5 +6 +7  +8 +9 +A +B  +C +D +E +F
0xFFFFB03C:  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00
0xFFFFB04C:  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00
0xFFFFB05C:  00 00 00 00  E4 B5 FF FF  00 00 00 00  00 00 00 00
0xFFFFB06C:  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00
0xFFFFB07C:  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00
0xFFFFB08C:  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00
0xFFFFB09C:  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00
0xFFFFB0AC:  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00
0xFFFFB0BC:  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00
0xFFFFB0CC:  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00
0xFFFFB0DC:  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00
0xFFFFB0EC:  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00
```

Enable

850eserv2 Target Commands

- These commands are directed to the target server.
  - The commands have direct hardware access

- A complete description is available in the manual
  - **V850/850E ICE SERVER Reference Manual**
    -> sv-v850e2-us-xx.pdf

- A choice of available commands can be found below

DFDUMP	Dumps to Data Flash memory
FERASE	Erases flash memory blocks
DFMAP	Maps the Data Flash area
DFSAVE	Saves to ID tag format from Data Flash memory
FLOAD	Downloads to Flash memory
IDTAG	Writes ID-tag to Data Flash memory

850eserv2 Target Commands

BRA	Sets bus event detectors
BRS	Sets execution event detectors
FBREAK	Sets Software break setting mode in Flash memory
FLSF	Sets and Displays Fail-safe-break
HWBRK(B)	Causes a break
LINK	Sets sequential events
PB	Sets and Displays Peripheral break
SHOWALL(SA)	Displays current trace analyzer settings

850eserv2 Target Commands

CPU	Displays or changes the internal ROM/RAM size
DCLOCK	Sets and displays the target's minimum operating frequencies
HSPLOAD	Downloads at high speed
ILLOPBK	Sets operational mode when the illegal op-code exception occurs
MAP	Sets the emulator memory configuration
FLASH	Shows CPU Flash memory information
HELP	Displays command summary
VERIFY	Turns memory verify on or off
VERSION	Shows version of 850eserv2

850eserv2 target server manual

- Latest target server manual for 850eserv2 will be installed with the MULTI installation and also with every target server update installation.

- Latest 850eserv2 target server manuals in PDF format can be found in the directory <GHS_install_dir>\manuals

- Document filename is **sv-v850e2-us-xx.pdf**

- The document contains all information about the connection process from MULTI debugger as well as reference information about 850eserv2 commands and target server specific debugging features

Scripting

Target Setup

- The simplest way of using a script
  - Configuring the target board and CPU using either
    - An MBS file + connection manager
    - An RC-File creating the connection
  - Example of a typical script file automating the target connection

```
connect 850eserv2 -minicube2 -noiop -id ffffffffffffffffffff -df=DF3793tg.800 -p=csib0
target dclock 4000 32768 swoff
SERVERTIMEOUT = 1000
// Make connection faster
target sfr PCC=0
```

Scripting Principles

- The Multi Debugger has a command interface, we refer to as the command-pane.
  - Any Multi debugger commands can be used within scripts
    - Example: **`view variable`**
  - Any commands of the debug target server can be used
    - Example: ***`target`*** **`sfr P0`**
      - The keyword ***`target`*** sends the command directly to the debug server!
  - There are conditionals, loops and special synchronization instructions available, such as
    - Create/Modify a variable: **`$Variable=5`**
    - Waiting for some time (in ms): **`wait –time 1000`**
    - Looping:

```
while ( $i>0) {
       Port=($i<<2);
       wait -time 1000;
       $i--;}
```

Command Manipulation and Macro Commands

- **alias** [*string1* [*string2*]]
  - Creates or lists aliases that translate one specified string into another string.
- **cedit** *command*
  - Executes the *command* specified as its argument and places the command output in an Editor window.
    - This is useful for examining the output of commands that print large amounts of information.
- **define** *name*([*arguments*]) { *body* }
  - Creates a macro for later use in the Debugger.
    - Example:

```
MULTI> define sum(x, y) {return(x + y)}
MULTI> sum(3,6)
```

Command Manipulation and Macro Commands

- **macrotrace**
  - Prints the stack of all presently executing macro commands.

- **return** [*expr*]
  - Returns from the currently executing macro, evaluating *expr*, if specified, and returning it as the macro value.

- **sc** ["*command*" | <*filename*]
  - Performs syntax checking on either a single command or an entire script file and all nested script files.

- **shell** [-w] *commands*
  - Invokes a shell to run the specified *commands*.

Command Manipulation and Macro Commands

- **substitute** *cmd_string*
  - Allows the output of Debugger commands to be reused when issuing other Debugger commands.
    - For example, if you wanted to use a graphical file chooser to specify the path to a file to edit, you can issue the following command:

```
substitute edit "%EVAL{filedialog}"
```

- **unalias** *string*
  - Reverses a previous **alias** command; disassociates *string* from its substitution.

Conditional Program Execution Commands

- **break**
  - Breaks out of a loop created with the Debugger **while** command. This is similar to the **break** command in C.

- **if** *expr* {*commands*} [else {*commands*}]
  - Specifies a conditional command execution.

- **while** (*expr*) {*commands*}
  - Executes the command list *commands* as long as *expr* (an expression in the current language) evaluates to a non-zero value.
    –
    ```
    while ( $i>0) {
         Port=($i<<2);
         wait -time 1000;
         $i--;}
    ```

Record and Playback Commands

- **>** [*file* | t | f | c]
  - Controls or displays the status of ***command*** recording, where:
    - *file* -- Sets the command recording file to *file* and turns on command recording.
    - t -- Turns on command recording (to the most recently set command recording file).
    - f -- Turns off command recording (but does not close or reset the command recording file).
    - c -- Turns off command recording and closes the command recording file. (A new recording file will need to be set before recording can be performed again.)
    - If no argument is specified, the **>** command displays the current command recording status.

- **<** *file*
  - Starts command playback from the specified *file*.

Record and Playback Commands

- **>>** [*file* | t | f | c]
  - Controls or displays the state of ***screen*** recording (recording commands and their output), where:
    - *file* -- Sets the screen output recording file to *file* and turns on screen output recording.
    - t -- Turns on screen output recording (to the most recently set screen output recording file).
    - f -- Turns off screen output recording (but does not close or reset the screen output recording file).
    - c -- Turns off screen output recording and closes the screen output recording file. (A new recording file will need to be set before recording can be performed again.)
  - If no argument is specified, the **>>** command displays the current screen output recording status.

Record and Playback Commands

- Common Rules for Operation
  - If you use the “**>” *or*** “**>>”** command when a recording file is already set, the old recording file will be closed and all subsequent commands will be recorded to the new *file*.
  - Scripts may include other scripts, to a maximum script depth of 500.
  - The playback file should not contain any lines that begin with **>** or **<**.
  - You cannot play back from a file that is open for recording, or record to a file that you are playing back.
  - Some commands can cause errors that may abort playback. You can use the Continue running script files on error GUI option (or the **ContinuePlaybackFileOnError** configuration option) to prevent these commands from stopping a playback.

External Tool Commands

- **make** [*string*]
  - Executes the system command **make** and passes to it the arguments you supply in *string*. If **make** succeeds, it kills the current process, removes all state information, reloads the program you are currently debugging, and allows you to continue debugging.

- **socket** [-global] *port_number*
  - Opens a socket connection using the specified port number. The socket connection allows an external program to send commands to the Debugger and receive output from the Debugger.
    - For example, if you started MULTI on a machine named myhost and used the command “socket 40000”, you could run the command telnet myhost 40000 to connect a telnet window to the Debugger.
    - Instead of using telnet, you can run any program that connects to host on port 40000 interacting with Multi.

History Commands

- **!** *num* | *string* [*args*]
  - Re-executes commands.
  - Example

```
– MULTI> echo hello
  hello
– MULTI> !echo hello
   hello hello
```

- **!!** [*args*]
  - Re-executes the last command.

- **h** [clear | *num*]
  - This command has three forms:
    - **h** -- Lists the existing command history.
    - **h** clear -- Clears the command history.
    - **h** *num* -- Lists the most recent *num* entries in the command history.

Controlling the Application

- Run command
  - ‘r’ starts the execution
  - ‘c’ continues execution from current pc

- Break commands
  - b <@count> label { commands to execute }
  - B[i|I] <@count> address { commands to execute }
    Example:

    -
      ```
      b @1 "applilet2_src\main.c"#LEDToggle#2
      {WarmStartCounter=0;}
      ```
  - B[x|X] label | address { commands to execute }
    - Sets a breakpoint at the end of a subroutine
  - Attributes in uppercase, like I or X, are temporary breakpoint
    - Once hit, they are removed …

Running Multi Debugger without GUI

- Automated scripting may NOT require a human interface.
  - Use Multi Debugger with “-run” option!

    Example:
    - **`multi -nosplash -rc run5.rc -run -- %1`**

      This executes multi in background, running the script run5.rc, which defines the connection method and all required actions.

```
remote 850eserv2 -minicube2 -noiop -id ffffffffffffffffffff -
df=DF3793tg.800 -p=csib0
wait
target dclock 8000 32768 swoff

bi _exit {
  echo Disconnect now
  disconnect
  }
bI main {
  echo Hello
  c
  }
```

Python

- A new Python script allows you to spawn a Debugger window for a target listed in the current Debugger window.

Memory Test

- The Multi debugger provides built-in memory tests.
  - Example:

```
MULTI> memtest 0x03ff5000 0x03ffe000 -size=4 -test=cc
CRC result: 0x0557d2dc
Memory test passed.
```

- Please do not use the target agent for memtest purpose, as it is not using proper memory locations, as per status of today!

(latest GHS Release can be used!)

Useful alias Settings

bra:	target bra
brs:	target brs
fload:	target fload
hb:	target b
mw:	target m -d4
sfr:	target sfr
timer:	target timer
ver:	target version

Sample Startup Script

- The script below is an automated startup connecting either to IECUBE or MINICUBE.

```
configure ContinuePlaybackFileOnError On
>yesno.rc
$bb=0
$i=100
>c
if(_REMOTE_CONNECTED == 0) {
     connect 850eserv2 -minicube -id=ffffffffffffffffffff -noiop -df=DF3375.800
     connect 850eserv2 -iecube -noiop -df=DF3375.800
     target dclock 6000 0 swoff
     }
echo MULTI
c
if(_REMOTE_CONNECTED == 1) {
     while( $i >0) {
          <yesno.rc
          if( $bb==1) { break; }
          wait -time 1000;
          $i--;
          }
     }
halt
```

850eserv2 Target Commands

- These commands are directed to the target server.
  - The commands have direct hardware access

- A complete description is available in the manual
  - **V850/850E ICE SERVER Reference Manual**
    -> sv-v850e2-us-xx.pdf

- A choice of available commands can be found below

DFDUMP	Dumps to Data Flash memory
FERASE	Erases flash memory blocks
DFMAP	Maps the Data Flash area
DFSAVE	Saves to ID tag format from Data Flash memory
FLOAD	Downloads to Flash memory
IDTAG	Writes ID-tag to Data Flash memory

850eserv2 Target Commands

BRA	Sets bus event detectors
BRS	Sets execution event detectors
FBREAK	Sets Software break setting mode in Flash memory
FLSF	Sets and Displays Fail-safe-break
HWBRK(B)	Causes a break
LINK	Sets sequential events
PB	Sets and Displays Peripheral break
SHOWALL(SA)	Displays current trace analyzer settings

850eserv2 Target Commands

CPU	Displays or changes the internal ROM/RAM size
DCLOCK	Sets and displays the target's minimum operating frequencies
HSPLOAD	Downloads at high speed
ILLOPBK	Sets operational mode when the illegal op-code exception occurs
MAP	Sets the emulator memory configuration
FLASH	Shows CPU Flash memory information
HELP	Displays command summary
VERIFY	Turns memory verify on or off
VERSION	Shows version of 850eserv2

User Manuals and further reading

GHS MULTI reference manuals

- All GHS manuals in PDF format can be found in the subdirectory <GHS_install_dir>\manuals

- Next to this all information can also be accessed by the GHS help system (Menu: Help -> Manuals)

- Most important manuals are:
  - **build_v800.pdf**: Contains all important information on the GHS build process, compiler, linker, etc.
  - **debug.pdf**: Contains all important information on the GHS MULTI debugger, generic debugger features and the GUI
  - **debug_cmd.pdf**: Contains a detailed reference of all commands of the GHS MULTI Debugger command line
  - **script.pdf**: Contains all neccessary information about the GHS MULTI scripting abbilities (e.g. MULTI-Python integration, MULTI-Python API reference)

850eserv2 target server manual

- Latest target server manual for 850eserv2 will be installed with the MULTI installation and also with every target server update installation.

- Latest 850eserv2 target server manuals in PDF format can be found in the directory <GHS_install_dir>\manuals

- Document filename is **sv-v850e2-us-xx.pdf**

- The document contains all important information about the connection process from MULTI debugger as well as reference information about 850eserv2 commands and target server specific debugging features

Further reading

- Renesas CPU core architecture manuals for V850E, V850ES or V850E2M

- Renesas Device Hardware user manuals for corresponding CPU derivates (e.g. V850ES/Fx3, V850E2/Dx4, etc...)

- ANSI C / C++ specification

- MISRA-C specifications (MISRA-C:2004, MISRA-C++:2008)

ARIGATO
- Thank you -
– 感謝您 –
- Dankeschön -

Renesas Electronics Corporation