Freescale Semiconductor
Application Note

Document Number: AN4233

HSST Setup for VTB Fast Download

1. Introduction

The main purpose of this application note is to illustrate the steps required to setup the High-Speed Simultaneous Transfer (HSST) download method for Virtual Trace Buffer (VTB) fast download.

This is a VTB trace download method which is 6 to10 times faster than the default JTAG download method.

All the results and screenshots in this document are based on CodeWarrior10.1.5 release. For newer releases, there might be minor GUI changes but functionalities should be similar.

This application note assumes that the user is familiar with the common terms used in Software Analysis feature.

Contents

2. High-Speed Simultaneous Transfer (HSST)

HSST is a method to facilitate data transfer between an application that runs on a target (MSC8156 target) and a host side application (CodeWarrior application).

HSST download method uses the same physical communication layer as JTAG download method, but due different implementation the download speed is increased about 6 to10 times as compared to the JTAG download method.

While the JTAG download method transfers data by halting the target, exchanging the data followed by resuming the target, the HSST download method works faster because the transfer between target and host computer occurs without stopping the target.

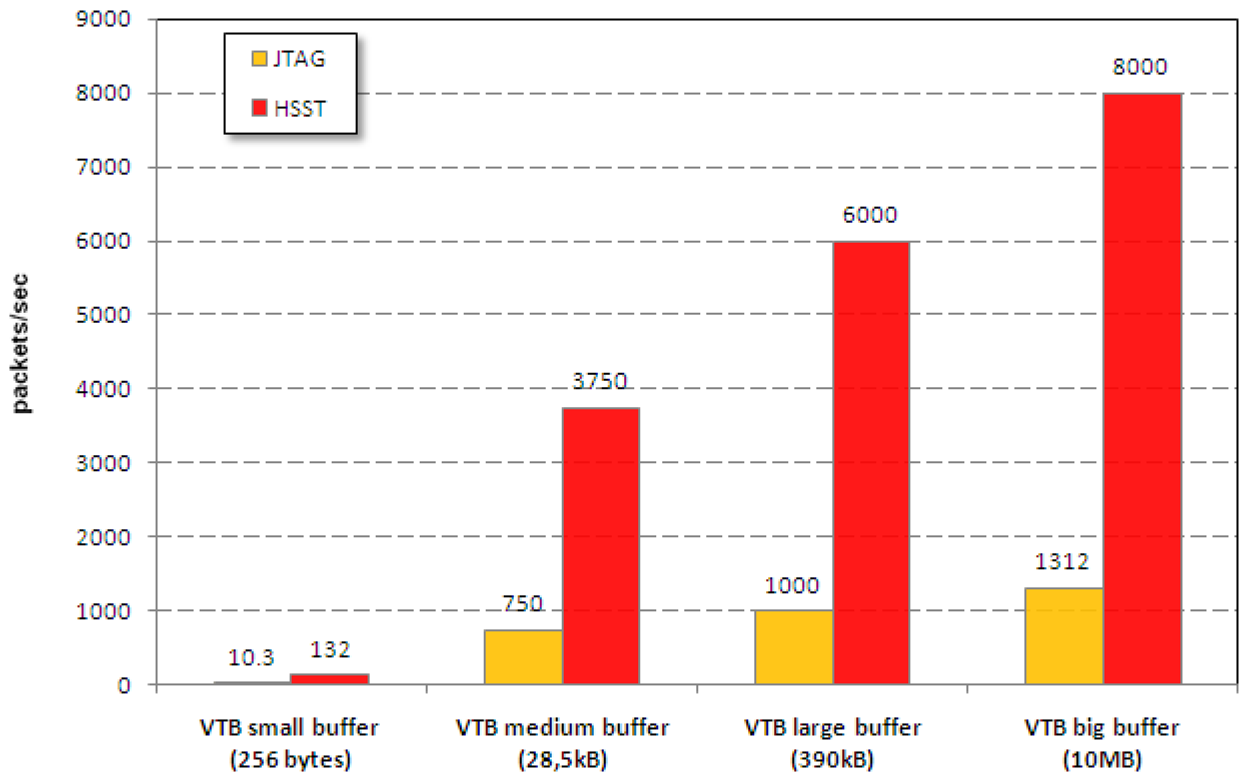
NOTE The target is not stopped during the data transfer. The drawback of this method compared with JTAG download method is, larger memory requirements from the target point of view. This is due to the mechanism implemented: when the target calls to standard I/O functions, these calls are redirected via HSST I/O calls at a lower layer implemented in the HSST libraries, in order to transfer the data from the target. The HSST libraries contain extra code and data sections that need to be linked with the user's main application.

NOTE HSST supports file I/O and console I/O also. These features are not covered into this application note. The focus is on necessary settings for VTB download only.

3. Benefits of HSST

The main benefit of HSST is the higher download speed required for transferring large amount of data during application profiling.
Figure 1 shows a comparative download speed between JTAG and HSST methods, using the same communication channel parameters. In this example, the packet size is 16 bytes.

Figure 1. Download Speed Comparison Between HSST and JTAG



For large and big chunks of download data, the HSST is about six times faster than JTAG method.

The default setup includes JTAG download method and the VTB size. It results in slow download speed while tracing huge applications.

This type of setup is causing performance penalties in download speed since a high number of download actions are required to get the data from the target, and in the same time, the improper setup of the trace can cause application errors due to the broken application's real-time behavior (caused by multiple processor halts).

NOTE CodeWarrior offers even a faster method for download: HEAT. In most of the cases, this method cannot be used in user custom applications due to its hardware and software requirements. In these conditions, the HSST is only the best method to speed up the VTB download process.

4. Step-by-Step Configuration

This section lists the steps required for a successful setup of HSST VTB download method.

4.1. Assumptions About Use Case

The setup explained in this application note is based on the following assumptions:

1. You can run the application on all the six cores of the MSC8156 and each of the cores are traced. In case, you are using less than six cores, the setup remains the same.

   NOTE This application note can also be used for other types of DSPs. You might need to make specific adjustments to match the platform in use, for example, MSC8144 target.

2. The communication or connection between MSC8156 and the host PC is done using JTAG interface with either USB TAP or Ethernet TAP.
3. SmartDSP OS is not available. In this case, the recommended method is HEAT.
4. The VTB is placed in DDR and its size matches the large or big data download criteria, see Figure 1. for details.
5. The project used for demonstration is based on MSC8156 stationery project, modified to fill a large VTB buffer.

4.2. Step1 – Add HSST Support to Project

To support the HSST protocol, the application must be linked with two HSST libraries provided by Freescale:

1. **`hostio_be_x.elb`**: this library is located at `<CWInstallDir>\SC\StarCore_Support\compiler\lib\msc8156\`

   NOTE `<CWInstallDir>` is the directory where CodeWarrior is installed.

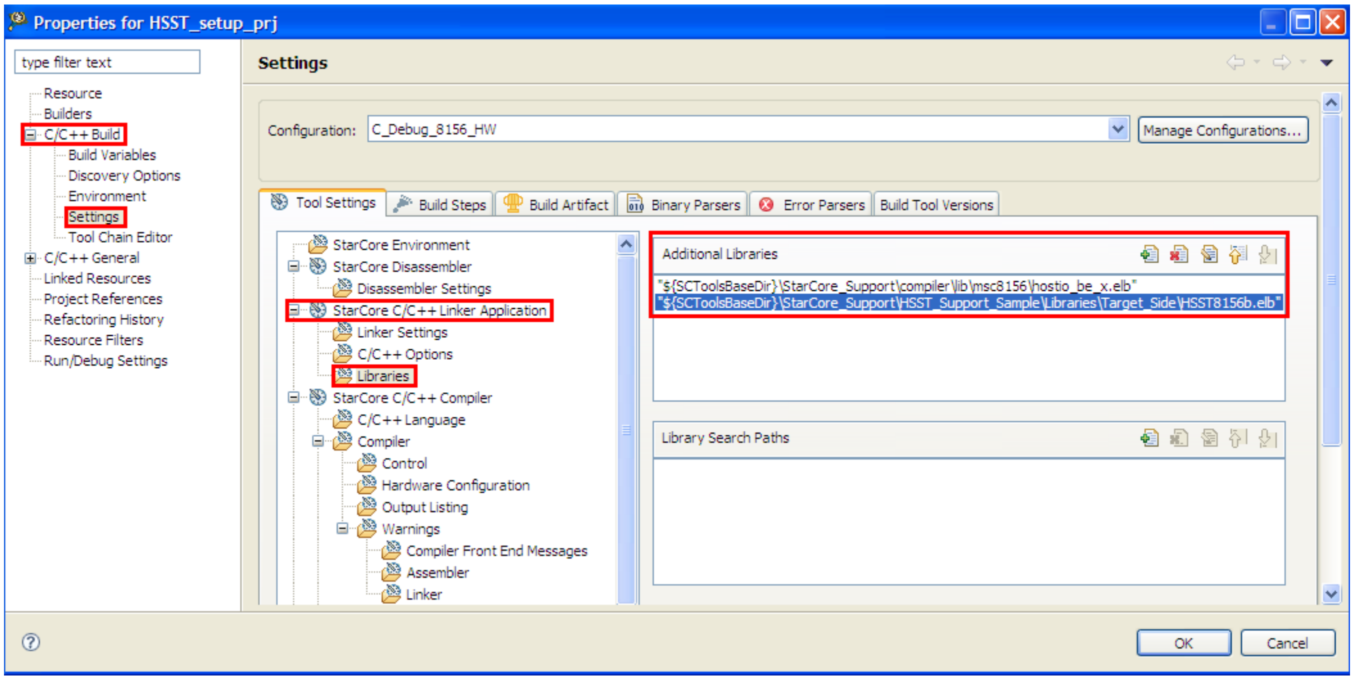
   NOTE Multiple versions of the hostio libraries are located in this folder. For the purpose of this application note, the library with big endian "_be" and long-long/double "-x" support has been selected.

2. **`HSST8156b.elb`**: this library is located at `<CWInstallDir>\SC\StarCore_Support\HSST_Support_Sample\Libraries\Target_Side\`

   NOTE If the project is built for another type of platform, make sure to select the appropriate library.

In order to link these two libraries with the application, the project linker settings need to be modified. Open the project properties, go to the linker library settings located under `C/C++Build\Settings\StarCode C/C++ Linker\Libraries` and add the two libraries into the **Additional Libraries** panel, see Figure 2. for details.

Figure 2. Adding HSST Libraries to Project



Properties for HSST_setup_prj

Settings

Configuration: C_Debug_8156_HW

Additional Libraries

"${SCToolsBaseDir}\StarCore_Support\compiler\lib\msc8156\hostio_be_x.elb"
"${SCToolsBaseDir}\StarCore_Support\HSST_Support_Sample\Libraries\Target_Side\HSST8156b.elb"

Library Search Paths

NOTE In case, your project already contains multiple libraries, after adding these two new libraries, it is recommended to use *reread-lib* linker additional option in order to force the linker to reread all libraries until it cannot resolve any more references. Otherwise, depending on the additional libraries include order, the linker might return errors.

Make sure you have instructed the linker to generate and keep the *.map file for the project. This file is needed to confirm/check that the setup of HSST is done correctly. If the map file contains the symbols related with HSST (like "`_hsst_syscall`") then it is guaranteed that HSST method for download is used. Otherwise, the default JTAG Stop-Transfer method will be used.

Also, the `*.map` file will be used to allocate the VTB buffer into a DDR free memory region after the application is built with HSST support.

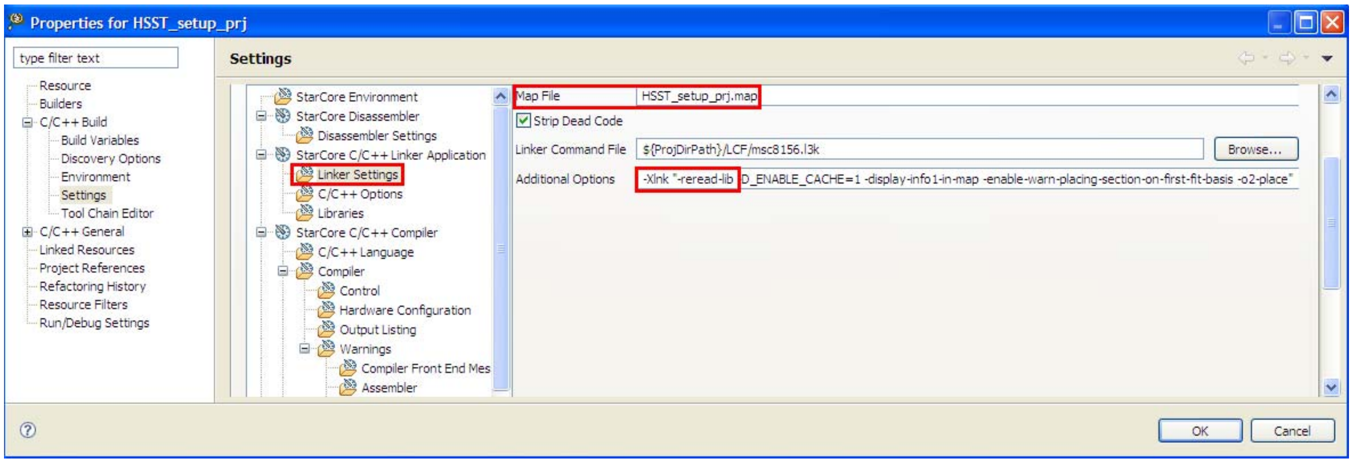


Figure 3. Linker Configuration for Map File Generation and Additional Options

Map File: HSST_setup_prj.map

✓ Strip Dead Code

Linker Command File: ${ProjDirPath}/LCF/msc8156.l3k

Additional Options: -Xlnk "-reread-lib D_ENABLE_CACHE=1 -display-info1-in-map -enable-warn-placing-section-on-first-fit-basis -o2-place"

4.3. Step2 – Setup HSST Communication Channel Parameters

In addition to the HSST libraries, you need to declare two global variables that are used to configure the communication channel between the target and the host PC.

```
void* hsst_hostio_stream = NULL;
char* hsst_channel_name = (char *)0xffffffff;
```

NOTE To reduce the number of modifications added to the project, these settings will be done in the same file with the settings for the Software Analysis interrupt handler.

The application that is executed on the target will not change the communication channel name `hsst_channel_name` during the execution. The debugger handles this automatically.

4.4. Step3 – Setup Software Analysis Interrupt

A DPU interrupt will be triggered in order to transfer the VTB trace data from the target to the host PC when the writes into the VTB reach at, *VTB Trace Event Request Address*.

The Enhance Interrupt Controller (EPIC) can get the source of interrupt from two places: interrupt debug A (DPUA) and interrupt debug B (DPUB) respectively. Depending on DPU control register (DP_CR) settings, either DPUA or DPUB will trigger the interrupt that should handle the VTB data transfer from target (MSC8156) to the host PC.

The source of the interrupt is controlled by `DP_CR.DETB` bits. By default, the software analysis engine configures these bits to trigger a DPUA to the EPIC.

NOTE For demonstration purpose, this application note considers that the VTB download interrupt source is DPUA. In case the VTB interrupt debug request is changed, make sure that the change is consistent over the next steps.

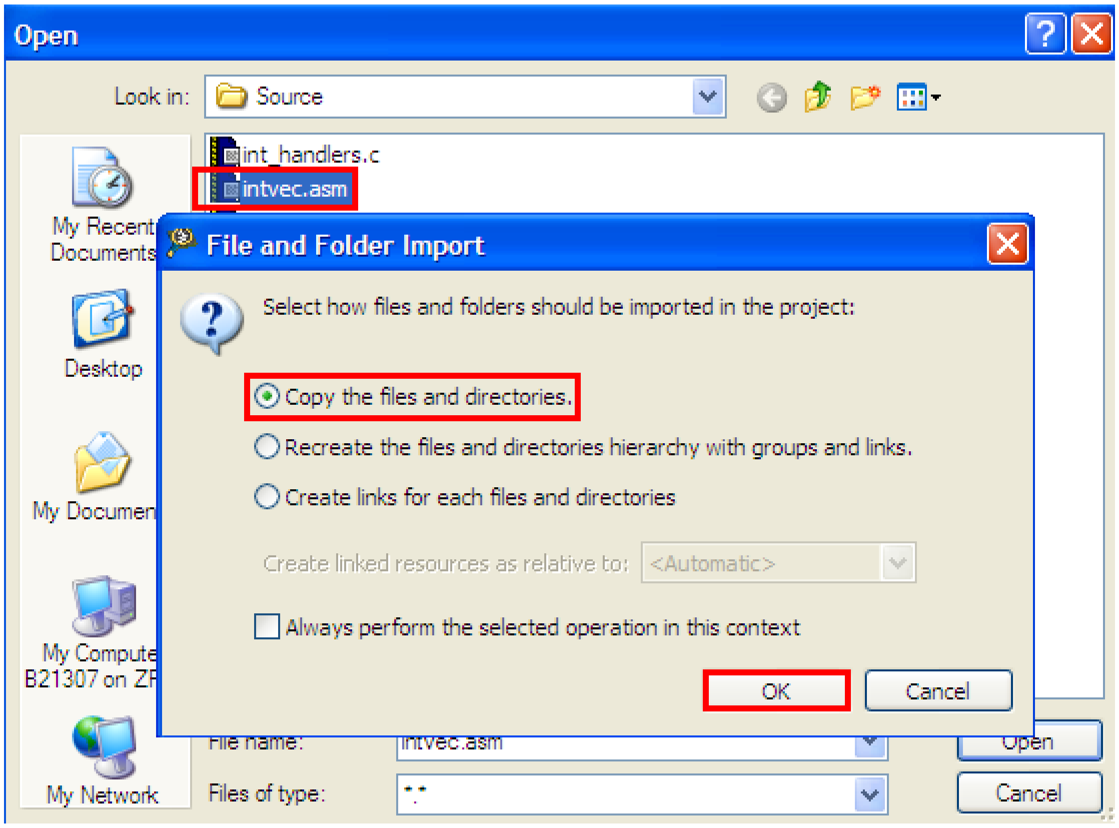
Based on the aspects presented above, the project needs to configure the EPIC. Once the *VTB Trace Event Request Address* is reached and the DPUA interrupt is triggered, the interrupt handler must take care of transferring the collected data from target to the host PC.

The easiest way to configure the EPIC is to take advantage of the demo projects provided with CodeWarrior. In order to configure the EPIC, the `intvec.asm` file is needed.

One example of `intvec.asm` file that can be used is, the one located at `<CWInstall>\SC\StarCore_Support\SoftwareAnalysis\swandemo_8156ads_hsst\Source\`.
For the correct EPIC configuration, this file needs to be added into the project.
Alternatively, you can use the file `sa_intvec.asm` located at `<CWInstall>\analysis\1.1.0\lib\host\engine\modules\HSST\`. It is recommended to use the one from the demo because these files are checked for correctness for each CodeWarrior release.

Use **Add Files…** from the context menu to select the file and then choose **Copy the files and directories** option. This option is required because the file needs to be modified for the purpose of this demonstration.

Figure 4. Add intvec.asm into Project



Select how files and folders should be imported in the project:

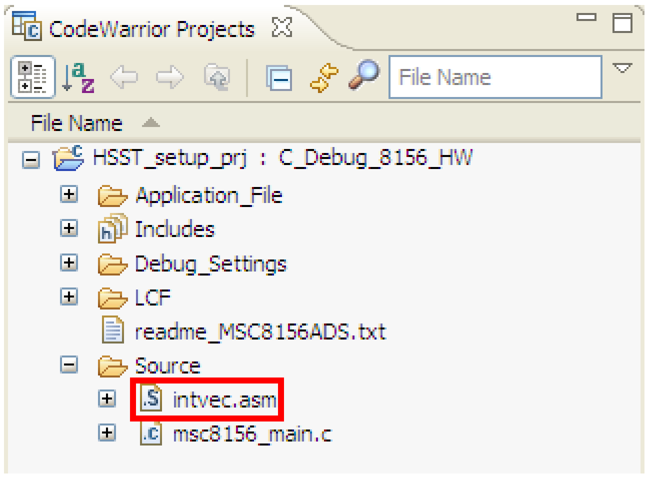
[✓ Copy the files and directories.]
Recreate the files and directories hierarchy with groups and links.
Create links for each files and directories

Create linked resources as relative to: <Automatic>

☐ Always perform the selected operation in this context

If the file import was successful, it should be displayed in the project source folder. In this particular case, the file has been imported into the project `Source` folder.

Figure 5. intvec.asm file location



Modify the newly added file to handle only the software analysis *VTB Trace Event Request Address* interrupt. The modifications are shown below:

Original intvec.asm	Modified intvec.asm
*jmp >**_timer0_handler*** *nop* *dup 4* *nop* *endm* *jmp >**_timer1_handler*** *nop* *dup 4* *nop* *endm* *jmp >_sa_ter_handle* *dup 13* *nop* *endm*	*jmp >**___EmptyIntHandler*** *nop* *dup 4* *nop* *endm* *jmp >**___EmptyIntHandler*** *nop* *dup 4* *nop* *endm* *jmp >_sa_ter_handle* *dup 13* *nop* *endm*

NOTE This method was chosen for its simplicity. You can implement your own method but special attention needs to be paid for EPIC offset from VBA. If the offset is not computed correctly, the `_sa_ter_handle` will not be executed and the download will not work.

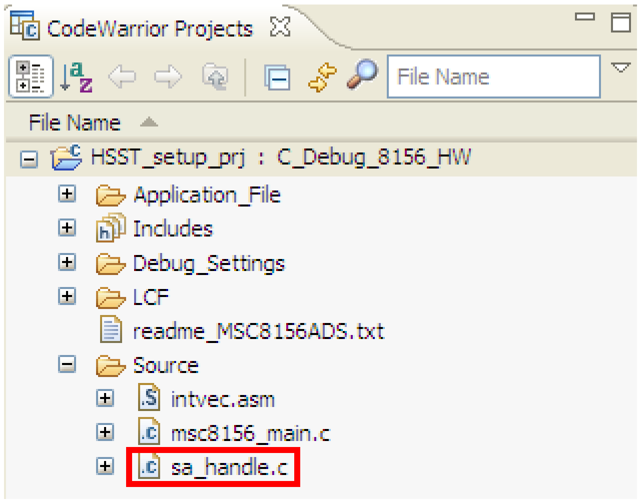
4.5. Step4 – Add Support for Software Analysis Interrupt Handler

After step 3 is complete, add the code for the interrupt handler. CodeWarrior provides a standard file `sa_handle.c` that can be used to configure the software analysis interrupt handler.

For the same reasons as explained in section 4.4, the recommended file is the one located in CodeWarrior demo folders `<CWInstall>\SC\StarCore_Support\SoftwareAnalysis\swandemo_8156ads_hsst\Source`.

In order to add it into the project, follow the same procedure as shown in Figure 4. In the project **Source** folder, the newly added file should be displayed.

Figure 6. sa_handle.c File Location



This file contains two major settings:

1. In the beginning of the file, the global variables of the HSST communication channel are added as explained in section 4.3.
2. Set the names of the files and location where the VTB data will be written using the HSST communication channel that is opened between the target and the host PC.

Regarding the second aspect, you are free to change both file names and location in order to match its restrictions. By default, the settings refer to `C:\` drive as this path exists on host PC.

```
switch(coreID){
      case 0:
            if(first_entry_0){
                        fp = fopen("C:\\sa_ter_hsst_0.txt","wb");
                        first_entry_0 = 0;
                  }
                  else{
                        fp = fopen("C:\\sa_ter_hsst_0.txt","ab");
            }
            break;
```

NOTE If the file name and path is changed in this file, you must update these settings accordingly in the Trace and Profile setup, see Figure 8.

Since the project used for demonstration is using all 6 cores to execute and trace the application, the number of HSST communication channels that will be opened by the target to communicate with the host PC will also be 6.

Therefore, the data/program sections associated with each HSST communication channel must be placed in each of the core's private memory. By default, for a stationery project the `program/data/rom/bss` sections are placed in M3 shared memory. If this default setting is not changed, the HSST VTB download method will not work.

To place the `sa_ter_handle()` into each core private memory, you can use the `*.appli` file support.

The following module needs to be added into the view used to compile the project:

```
module "sa_handle" [
            rom = M3__cacheable_wb__sys__shared__rom
            bss = M3__cacheable_wb__sys__private__bss
            data = M3__cacheable_wb__sys__private__data

            function _sa_ter_handle [
                        program = M2__cacheable__sys__private__text
                  ]
            ]
```

The `_sa_ter_handle` function program will be placed in `M2__cacheable__sys__private__text` and the data into `M3__cacheable_wb__sys__private__data`.

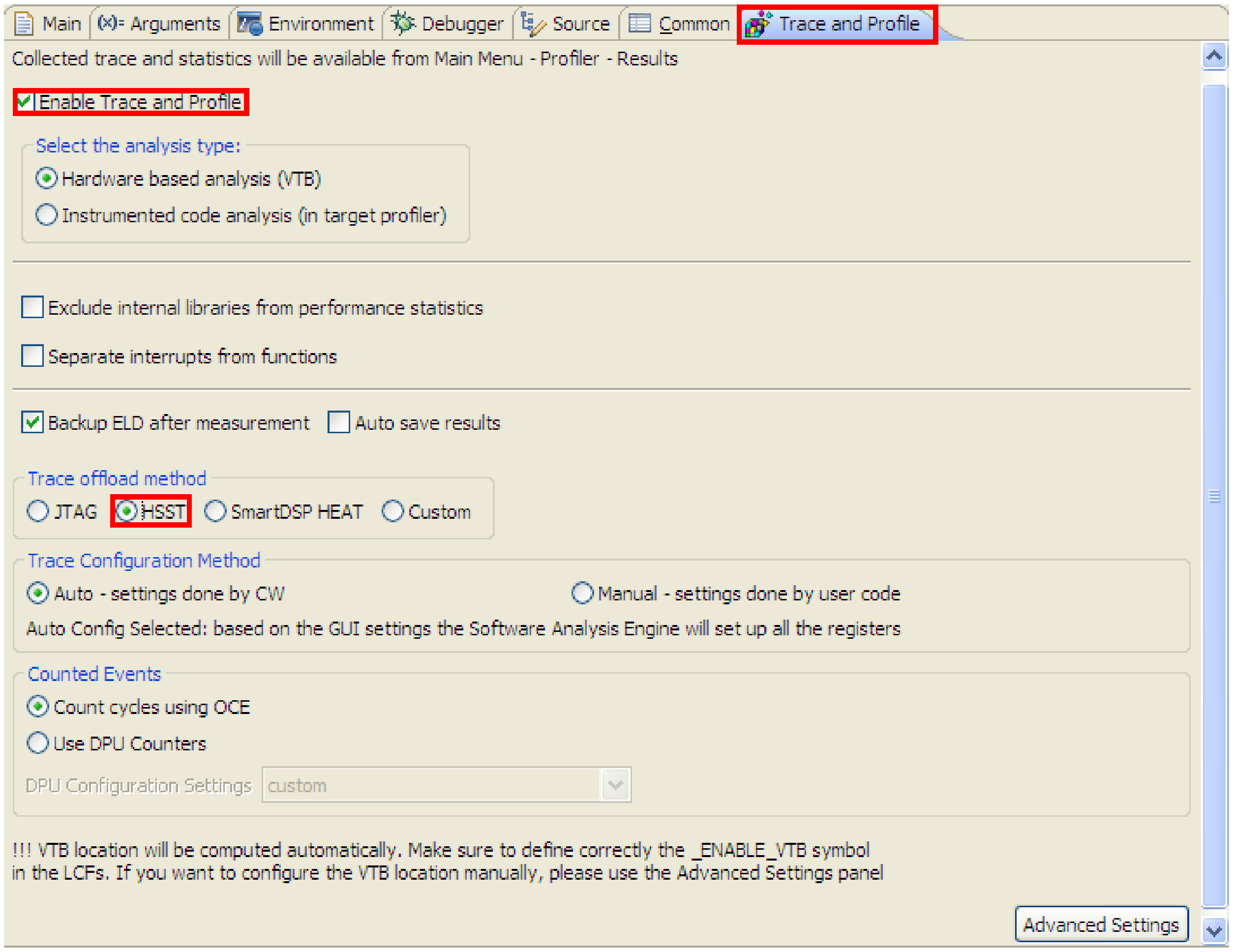
NOTE Make sure the name of the module and linker input sections are matching the ones from the user project. Otherwise, the compiler will return an error.

Now the project can be built and the map file checked to see if the modifications made to the project are in place.

4.6. Step5 – Configure Trace and Profile

The last step required to collect the VTB data using HSST download method is the Trace and Profile configuration. In the Trace and Profile configuration window, you must select HSST as **Trace Offload Method**, see Figure 7.

Figure 7. Enable Trace and Profile to Download Data Using HSST Method



Collected trace and statistics will be available from Main Menu - Profiler - Results

[✓] Enable Trace and Profile

Select the analysis type:
- [✓ Hardware based analysis (VTB)]
- Instrumented code analysis (in target profiler)

[ ] Exclude internal libraries from performance statistics

[ ] Separate interrupts from functions

[✓] Backup ELD after measurement [ ] Auto save results

Trace offload method: JTAG [✓ HSST] SmartDSP HEAT Custom

Trace Configuration Method: [✓ Auto - settings done by CW] Manual - settings done by user code

Auto Config Selected: based on the GUI settings the Software Analysis Engine will set up all the registers

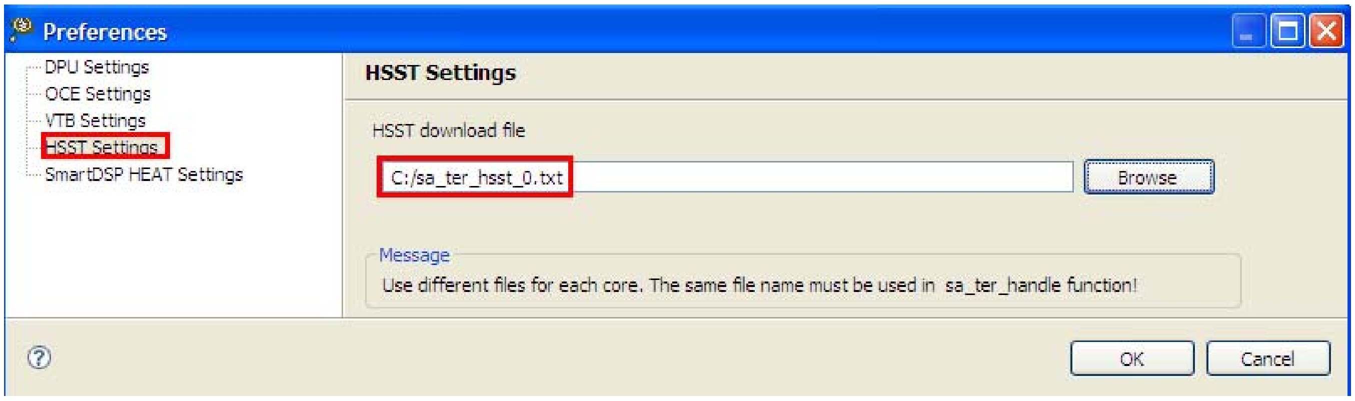
Counted Events:
- [✓ Count cycles using OCE]
- Use DPU Counters

DPU Configuration Settings: custom

!!! VTB location will be computed automatically. Make sure to define correctly the _ENABLE_VTB symbol in the LCFs. If you want to configure the VTB location manually, please use the Advanced Settings panel

In **Advance Settings**, the path of the HSST data files needs to be specified for debug configurations associated with the cores.

Figure 8. HSST Advanced Settings



HSST Settings

HSST download file: C:/sa_ter_hsst_0.txt

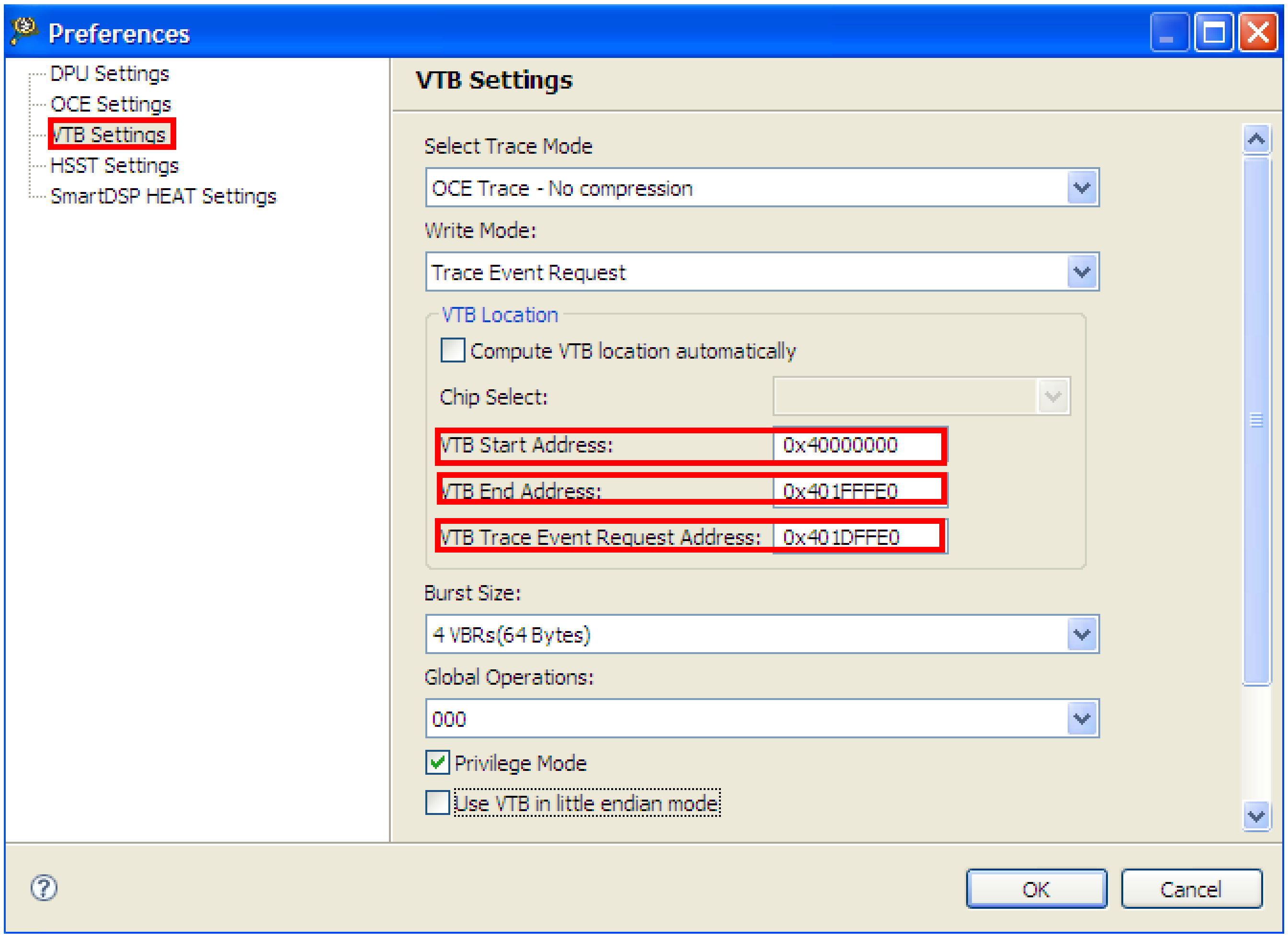
Message: Use different files for each core. The same file name must be used in sa_ter_handle function!

NOTE The path of the file must be identical with the one specified in section 4.5.

If the file does not exist, it will be created with the default name specified in `sa_handle.c` when the application is started. If the file already exists in the specified location, it will be overwritten when the application is started.

You need to allocate the VTB addresses in order to save the trace data. Figure 9 shows an example of VTB allocation in DDR for one core (for example, `core#0`).

Figure 9. Example of VTB Allocation for One Core



Select Trace Mode: OCE Trace - No compression
Write Mode: Trace Event Request
Compute VTB location automatically: ☐
Chip Select:
VTB Start Address: 0x40000000
VTB End Address: 0x401FFFE0
VTB Trace Event Request Address: 0x401DFFE0
Burst Size: 4 VBRs(64 Bytes)
Global Operations: 000
Privilege Mode: ✓
Use VTB in little endian mode: ☐

NOTE For each of the cores, the VTB should be private and must not overlap.

If you find it difficult to set the VTB addresses, check the *Compute VTB location automatically* checkbox to set the default allocation used by CodeWarrior.
If you have decided to use the default allocation, make sure you have done the appropriate settings into the linker files:

1. `mmu_attr.l3k` set the _ENABLE_VTB symbol with the memory bank you want to use.

```
// Definitions for the VTB
// if 1 = reserve VTB in M2 memory
// if 2 = reserve VTB in M3 memory
// if 3 = reserve VTB in DDR memory
// else VTB will not be configured automatically
   _ENABLE_VTB =            3;
```

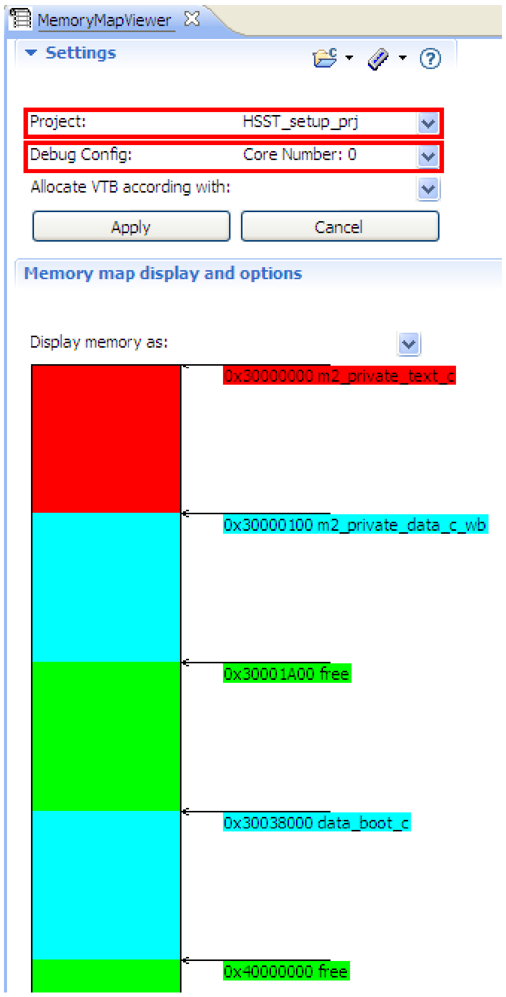
2. `common.l3k` set the size of reserved VTB for each physical memory.

```
// Reserve VTB are in physical memory
// Set VTB start address and size for M2, M3 and DDR
   _M2_VTB_size= 0x1000;   //4K for each core
   _M3_VTB_size= 0x8000;   //32K for each core
   _DDR_VTB_size= 0x1FFEE0;//2M for each core
```

Additionally, if you still face issue with VTB allocation, CodeWarrior offers a feature called **MemoryMapViewer** that can be opened from **Window > Show View > Other… > Other > MemoryMapViewer** that can be used to identify a free memory location, see Figure 10.

Figure 10 displays the MemoryMapViewer feature to help you to identify free memory regions.

Figure 10. MemoryMapViewer



Project: HSST_setup_prj
Debug Config: Core Number: 0
Allocate VTB according with:

0x30000000 m2_private_text_c
0x30000100 m2_private_data_c_wb
0x30001A00 free
0x30038000 data_boot_c
0x40000000 free

5. Things to Consider while Using HSST

In order to make sure that the data is collected successfully, follow the details which if ignored might cause wrong results:

1. If the size of the VTB is too big and the application trace does not reach the *VTB Trace Event Request Address,* the HSST transfer will not be done since the DPUA interrupt is never triggered. In this case, at the end of application the entire VTB will be downloaded using default Stop-Transfer method (JTAG). Make sure that the VTB is filled at least once during application tracing.

2. At times, no matter which download method is used, the last portion of the VTB (before the last triggered interrupt and the current location when the application was stopped) will be transferred using JTAG Stop-Transfer method. This cannot be avoided.

3. It is recommended not to use any kind of I/O in the application when HSST is used as offload method to trace the application. For example, if there is a `printf()` function which will be executed using HSST and the VTB is almost full when the offload interrupt is triggered both `printf()` and HSST offload methods might use the communication channel. In this case, some data will be lost.

4. In some cases, it is possible to see some gaps or even the whole VTB is lost. This is because the method to transfer the data between targets and host PC is using a Non-Maskable Interrupt. When the VTB reaches the VTB Trace Event Request, the hardware triggers an interrupt to begin the transfer. If other NMIs are triggered just before the VTB interrupt, those interrupts might fill the remaining VTB and the start addresses of this buffer might be overwritten as the write pointer is reset to the VTB start address. To avoid this kind of situation make sure that between the VTB Event Request Address and VTB End Address there are at least 100 bytes for each possible NMI.

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064, Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-521-6274 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Document Number: AN4233 AN4233

24 November 2010