Application Report
SPRA821 – May 2002

How To Use High-Speed RTDX Effectively

Doug Deao *Software Development Systems*

ABSTRACT

This application report familiarizes the reader with high-speed RTDX (HSRTDX). It is assumed that the reader already has a working knowledge of RTDX. This application report covers the HSRTDX target architecture, HSRTDX target application integration, HSRTDX performance hints, and a discussion on HSRTDX application intrusion. This application report does not cover RTDX basics. See the on-line help topics in Code Composer Studio™ for general information on RTDX.

Contents

Figures

1 Introduction

HSRTDX provides the user of TI's Code Composer Studio tool set a new dimension for application instrumentation. Whether you are simply looking for higher data logging rates or for increased responsiveness for RTDX enabled host controlled applications, HSRTDX provides over a 100x improvement when compared to the Traditional JTAG RTDX. To use HSRTDX you must have both a target DSP that is HSRTDX enabled and an XDS560 class emulation controller.

Only TMS320C6211™ DSP and TMS320C6711™ DSP devices currently carry the HS RTDX silicon module required for HSRTDX operation. Other devices do not operate in this mode currently. Plans are to include HSRDTX capability in future C6x1x™ DSP, C64xx™ DSP and C55xx™ DSP devices. Check the update advisor page often for Chip Support Package updates.

Important Note:

See the first section of the troubleshooting guide for guidance on converting the JTAG t2h and h2t test cases installed with CCS from JTAG RTDX to HSRTDX. Early versions of the C6211™ DSP were not characterized for running with HSRTDX. You should convert these simple test cases and run them to verify your target DSP performs HSRTDX properly. These test cases also serve as good examples of what is required to port HSRTDX to an existing RTDX application.

2 HSRTDX Target Architecture

HSRTDX relies on a silicon module (referred to as the HSRTDX unit) that provides an interrupt driven DMA interface between the EMU pin selected for RTDX data transport (EMU0 is the default) and the memory system of the target device. The EMU pin used for RTDX data transport may be modified through the RTDX configuration tool. All memory spaces addressable by the processor are accessible to the HSRTDX unit. HSRTDX uses three interrupts (INT3, INT11 and INT12 in the C6x1x™ DSP and C64xx™ DSP devices) to facilitate communications with the emulator, and to signal the RTDX library when a transfer is complete. The following diagram shows the basic HSRTDX target architecture.

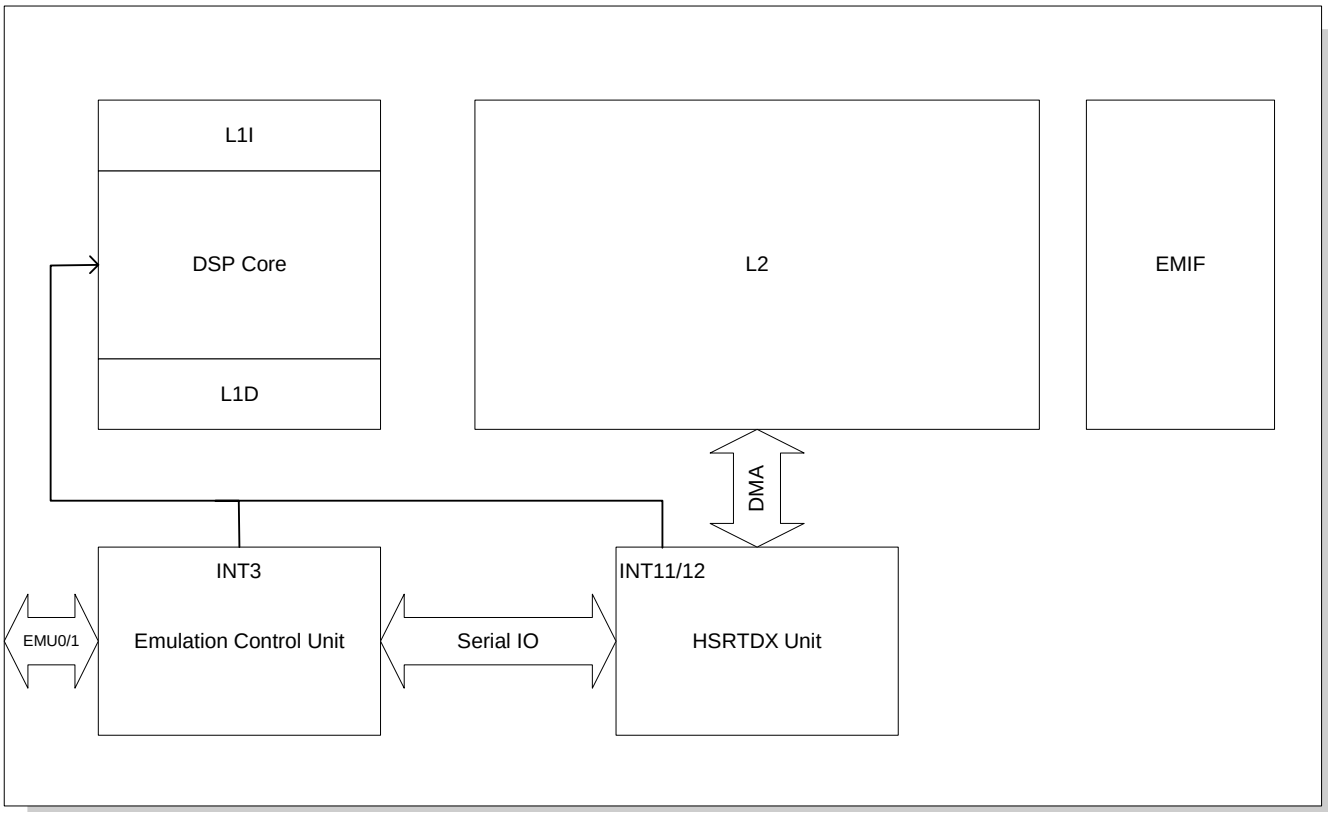

Figure 1. HSRTDX Block Diagram

The emulator uses INT3 to notify the target that it has a host-to-target transfer pending. INT11 is the HSRTDX transmit interrupt, which fires when a target-to-controller transfer completes. INT12 is the HSRTDX receive interrupt which fires when a host-to-target transfer is complete. The DSP is the master of the communications, therefore the emulator must notify the target DSP when it has a host-to-target transfer pending.

The XDS560 also provides support for multiprocessor systems, allowing two processors to drive the EMU pins (one driving EMU0 and the other driving EMU1) simultaneously.

3 Software Integration

Before integrating using your own code, follow the steps in Section 7.1 to convert the t2h and h2t test cases installed with CCS from JTAG RTDX to HSRTDX. If you are converting a BIOS application to use HSRTDX for RTA or your application code, look at the example projects at c:\ti\examples\dsk6211\hsrtdx or c:\ti\examples\dsk6711\hsrtdx.

3.1 Existing Application

If you are integrating with an existing application that uses RTDX follow these steps:

- Increase the size of the RTDX read buffers in your application code by one 32-bit word. You must do this without increasing the size of the RTDX_read() or RTDX_readNB() byte count parameter. The reason for this is that there is a design bug in the 6711 which causes the

last word of a read transfer to get trashed. To get around this problem the XDS560's HSRTDX driver transfers one extra word on host-to-target transfers. This means that only the target code needs to be updated and not the host code. This change must be made for all processors regardless of whether they contain the defect or not. If you are using RTDX in a BIOS application, BIOS already has this fix integrated.

- Integrate the required interrupt functions into your application's interrupt vectors. The C6x11's interrupt multiplexer reset configuration selects the RTDX Unit's interrupts.

  INT3 – RTDX_poll

  INT11 – RDTX_xmt

  INT12 – RTDX_rec

- Link your application code with the rtdxhs.lib (little endian library) or rtdxhse.lib (big endian library).

3.2 BIOS Application

If enabling HSRTDX for a BIOS application, enable RTDX with the BIOS configuration tool by following these steps:

- Modify the RTDX property for the *Input/Output=>RTDX - Real-Time Data Exchange Settings* from JTAG to HSRTDX.
- Modify the Host Link Type property for the *Input/Output=> HST - Host Channel Manager* to RTDX.
- If running with BIOS's Real Time Analysis tools make sure that the Enable Real Time Analysis property for *System=>Global Settings* is set to true.
- RTDX channels may be added to your application through either the standard RTDX library API or with the BIOS configuration tool. Highlight *Input/Output=> HST - Host Channel Manager*, then right click the mouse to bring up the Host Channel Manager menu and select *Insert HST*. Regardless of the method used, expand the size of the destination of your read buffer by one word more than your RTDX_read() or RTDX_ReadNB() transfer byte counts.

4 Performance

Since HSRTDX is transferring data at much greater rates than JTAG RTDX (2 Mbytes/sec versus 25-40KB/sec on the XDS510) there are additional factors to consider if you need to get as much performance out of HSRTDX as possible.

- There is a direct relationship between HSRTDX performance and the JTAG port TCLK rate. HSRTDX shares a clock pin with the JTAG port. To determine your TCLK rate, run the following utility in a command window:

  xdsprobe –f brddat/ccbrd0.dat –k

If xdsprobe reports a TCLK less than 26 Mhz for a 150 Mhz c6x1x you can increase your HSRTDX performance by improving your board's JTAG electrical interface. On c64xx devices that are HSRTDX enabled, the TCLK rate at which maximum performance is achieved is expected to be at the XDS560's maximum TCLK rate of 35 Mhz.

- Packet size matters. If you are sending data in only one direction, create packets as large as possible. RTDX supports packets up to 65535 bytes deep. Since the emulator has no idea what your application requirements are (transfer rate versus latency) it accumulates blocks of RTDX messages in a buffer to transport back to the PC every 500 µsecs. The controller may have up to four outstanding transfers with the PC at a time. Once this limit is reached the controller continues to accumulate data, but does not attempt to transport the messages until the PC releases one of the last four outstanding buffers. This provides some balance for systems that require fast latency with small messages versus high bandwidth for large unidirectional messages.

- HSRTDX transfers are interrupt driven so be careful with the IER (enabling interrupts) and the GIE (global interrupt enable). Leaving interrupts turned off for a long period of time may slow HSRTDX transfers.

- Enabling the DSP's caching provides a noticeable performance boost. Once the instruction cache is turned on there is less traffic with the EMIF, thus providing more bandwidth to the HSRTDX unit.

- Expand the size of the RTDX write buffer in the target code. The RTDX_write() function simply copies the source data into the RTDX write buffer, and queues the data for transport to the host. You can modify the size of the buffer a few times to see where you get the best performance for your application. For details on modifying the RTDX write buffer size, bring up the CCS Help menu and select *contents*. Within *contents* select *RTDX* , then select *How to…* , then select *Modify the Target Buffer Size*.

5 RTDX Configuration Control Tool

The RTDX configuration control tool is accessible through the Tools=>RTDX=>Configuration Control menu. When an RTDX application is loaded, the RTDX configuration control tool (in background mode) automatically configures the RTDX mode of operation based upon the RTDX library linked with the application program, but it does not enable RTDX. You can enable RTDX in the configuration control window through the RTDX enable check box, or your host application can enable RTDX.

For HSRTDX applications EMU0 is used by default. If you wish to change to EMU1, simply select the RTDX configuration control tool from the Tools menu, disable RTDX by clearing the RTDX enable check box, then select the configure button. The RTDX configuration control properties window is then displayed. Select the port configuration page index tab and change the configuration by selecting the desired configuration from the list.

If you wish to use the external counter (selected through the Analysis Events setup window) for benchmarking or event counting while running HSRTDX, you must select EMU1 rather than use the EMU0 default. EMU0 is used to drive the external counter.

The RTDX configuration tool may also be used to select continuous or non-continuous modes. Non-continuous mode logs RTDX data to a file. The filename can be modified in the RTDX configuration control properties window.

6 HSRTDX Port Connect and Disconnect

After Loading your program (linked with rtdxhs.lib or rtdxhse.lib) or symbols, RTDX is automatically configured for HSRTDX with EMU0 selected for the data transfer by the RTDX configuration control plug-in. When execution is started with the Run command, the emulator *connects* to the target DSP. You may also connect manually by using the RTDX configuration control plug-in from the CCS Tools-> rtdx menu. The following commands cause the emulator to *disconnect* HSRTDX from the target causing data flow to stop, and then *connect* when execution is resumed with the Run command.

- Load Program
- Load Symbols
- Restart
- Reset DSP

A hardware reset while running also causes the HSRTDX hardware to be *disconnected* and then automatically *connected* for new execution if RTDX is enabled. Hardware reset at other times (such as while halted) may cause unpredictable HSRTDX behavior. Selecting a non-HSRTDX configuration with the RTDX Configuration Control plug-in, or performing a Run Free command from Code Composer's Debug menu *disconnect*s HSRTDX from the target DSP. If you start DSP execution by modifying the PC through Code Composer (without performing one of the listed commands) the HSRTDX hardware is not *connected.*

Also keep in mind that a peripheral module within the DSP performs HSRTDX data transfers. This peripheral must be reset by an external reset (reset the DSP device through the reset pin) or an internal reset (reset the DSP using CCS's Debug-> Reset CPU menu command) prior to performing a CCS command that *configures* (Loads) or *connects* (Runs) HSRTDX.

If you are loading code for the first time after CCS initialization or you are loading a different program than the one you are currently running with HSRTDX, the HSRTDX disconnect, configure, and connect operations are performed but RTDX operation may not restart properly. Typically, this is due to cache coherency issues with older 6211 devices if the target's c6x instruction and data caches are not flushed prior to loading the new code. This is best accomplished by performing a DSP reset before loading new code. After performing the reset, properly reprogram the EMIF registers (normally through a gel script) to ensure your application and HSRTDX load and operate properly.

7 Troubleshooting

7.1 Test Cases

Early revisions of C6211/C6711 devices may not have been fully characterized for HSRTDX operation. It is strongly recommended that you convert the t2h and h2t test cases and run them to determine if the HSRTDX unit of your particular device is functional.

Convert t2h from JTAG to HSRTDX:

1. Open the t2h project at c:\ti\examples\dsk6711\rtdx\t2h\t2h.pjt
2. Change the rtdx.lib library to the rtdxhs.lib. Open up the Libraries folder for the project, highlight rtdx.lib, right click the mouse then select *Remove from project*. Then from the Projects menu select *Add Files to Project*, select *Files of type: Object and Library Files*, then select c:\ti\c6000\rtdx\lib\rtdxhs.lib (or rtdxhse.lib for a big endian target).
3. Open up the source folder and edit intvecs.asm. Comment out line 21 *JTAGRTDX .set 1* and add line 14 *HSRTDX .set 1*.
4. Build the t2h project.
5. Load t2h/Debug/t2h.out. Verify that HSRTDX has been automatically selected in the RTDX Configuration Tools window. The Hardware Mode should be HSRTDX and the Pins Used is either EMU0 or EMU1. Click on the Enable RTDX box.
6. Run. The CIO window shows the test progress by printing *Value N was sent to the host*. To verify the host received the data you may run the gpdprog (General Purpose Display) that is located at c:\ti\examples\hostapps\rtdx\gpdprog\new\gpdprog.exe. Click on Add Channel to bring up the Select Channel Properties dialog box. Set CHANNEL NAME to ochan, MAX MESSAGES to 100, MAX MEMBERS to 1, Channel Type to Read Only Channel, Data Type to 32-bit Integer; and then select OK. A data window pops up that fills in as the test runs. If you do not see an incrementing pattern of integers from 0 to 99 the test has failed. See Figure 2 and Figure 3.

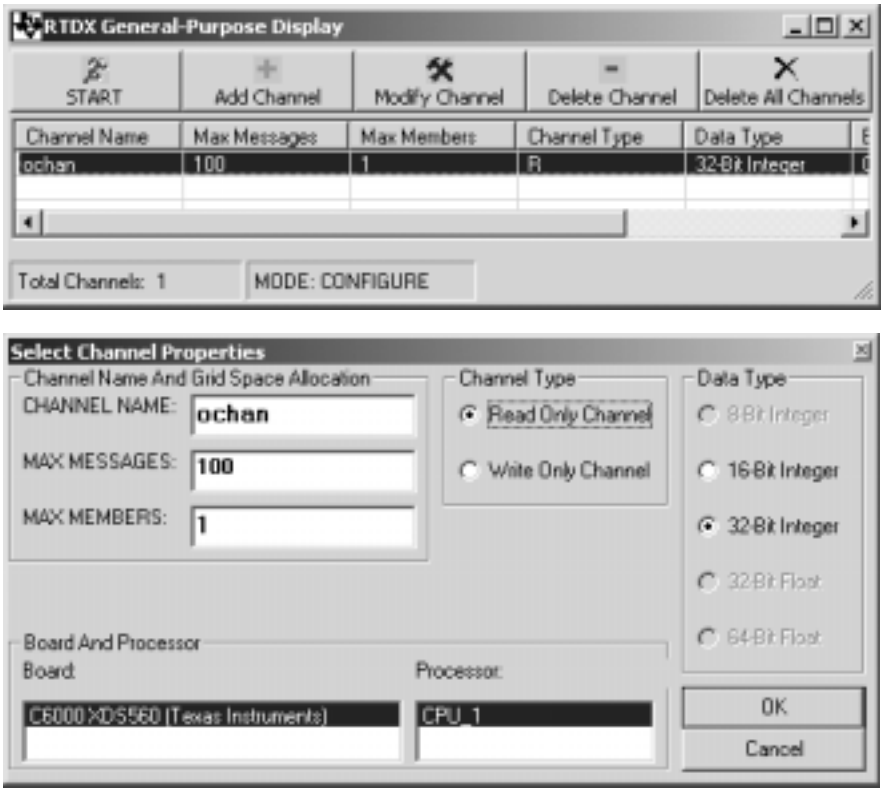

Figure 2. t2h General-Purpose Display - Setup

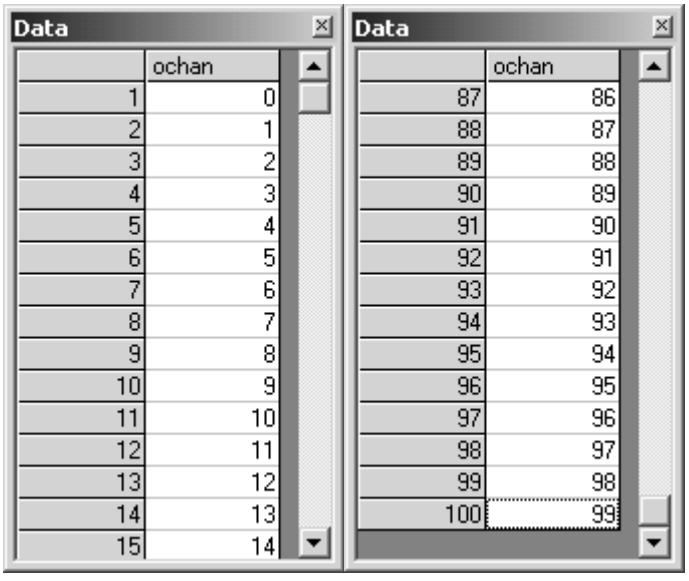

Figure 3. t2h General-Purpose Display -Test Results

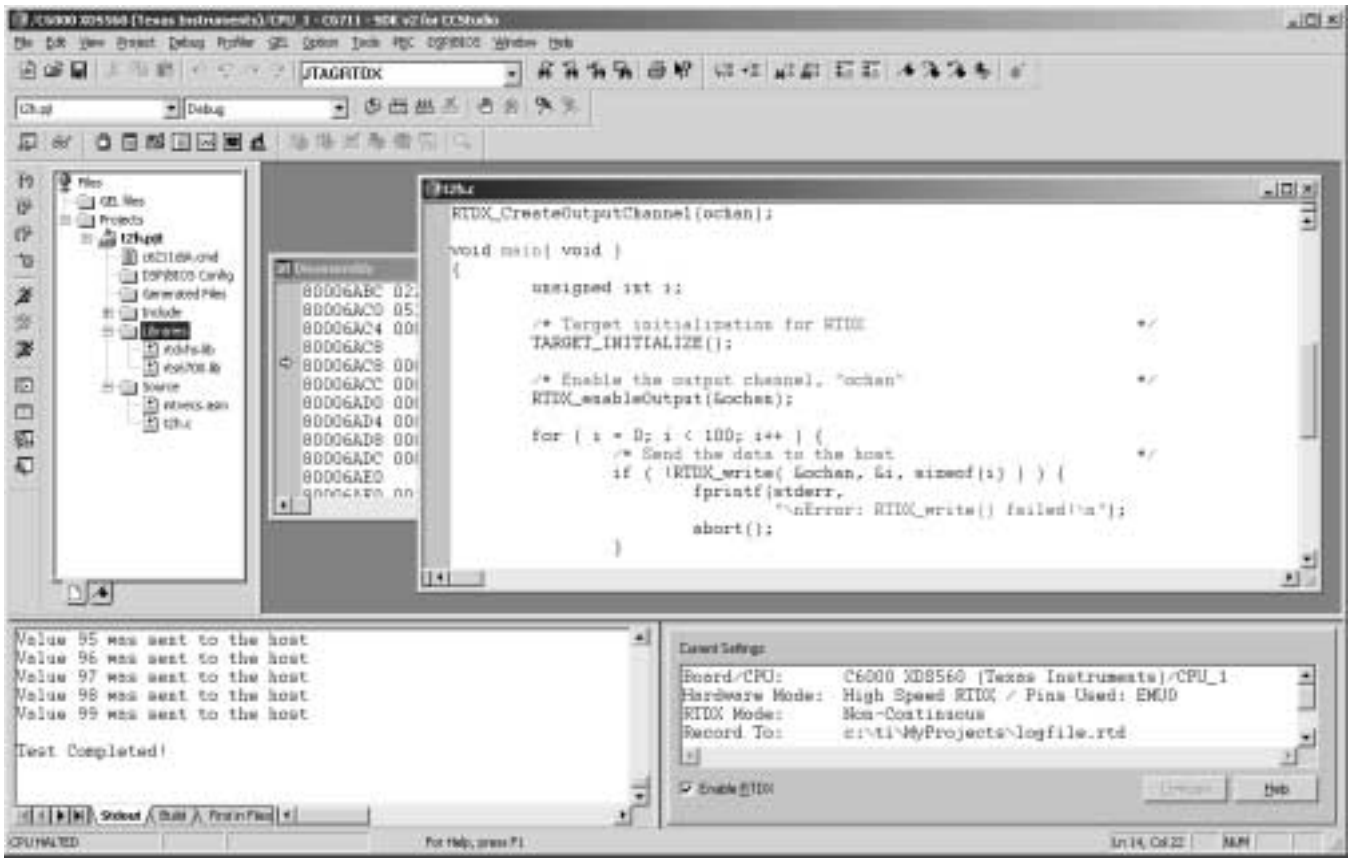

Figure 4. t2h CCS Test Results

Convert h2t from JTAG to HSRTDX:

1. Open the h2t project at c:\ti\examples\dsk6711\rtdx\h2t\h2t.pjt
2. Change the rtdx.lib library to the rtdxhs.lib. Open up the Libraries folder for the project, highlight rtdx.lib, right click the mouse then select *Remove from project*. Then from the Projects menu select *Add Files to Project*, then select *Files of type: Object and Library Files*, then select c:\ti\c6000\rtdx\lib\rtdxhs.lib (or rtdxhse.lib for a big endian target).
3. Open up the source folder and edit intvecs.asm. Comment out line 21 *JTAGRTDX .set 1* and add line 14 *HSRTDX .set 1*. IF YOU PERFORMED THIS STEP FOR t2h THEN YOU CAN SKIP THIS STEP SINCE intvecs IS A COMMON FILE.
4. Modify in h2t.c the variable recvd from an int to a two int array. Then modify every occurrence of recvd to recvd[0] in h2t.c.
5. Build the h2t project.
6. Load h2t/Debug/h2t.out. Verify that HSRTDX mode has been automatically selected in the RTDX Configuration Tools window. The Hardware Mode should be HSRTDX and the Pins Used is either EMU0 or EMU1. Click on the Enable RTDX box.

7. To generate data from the host, run the gpdprog (General-Purpose Display) that is located at c:\ti\examples\hostapps\rtdx\gpdprog\new\gpdprog.exe. Click on Add Channel to bring up the Select Channel Properties dialog box. Set CHANNEL NAME to ichan. MAX MESSAGES to 100, MAX MEMBERS to 1, Channel Type to Write Only Channel, Data Type to 32-bit Integer, and then select OK. A Data window pops up that is filled in as the gdpprog sends data to the target DSP—after CCS starts target execution with a Run command. The CIO window shows the test progress by printing *Value N was read from the host*. If you do not see an incrementing pattern of integers from 0 to 99 the test failed.

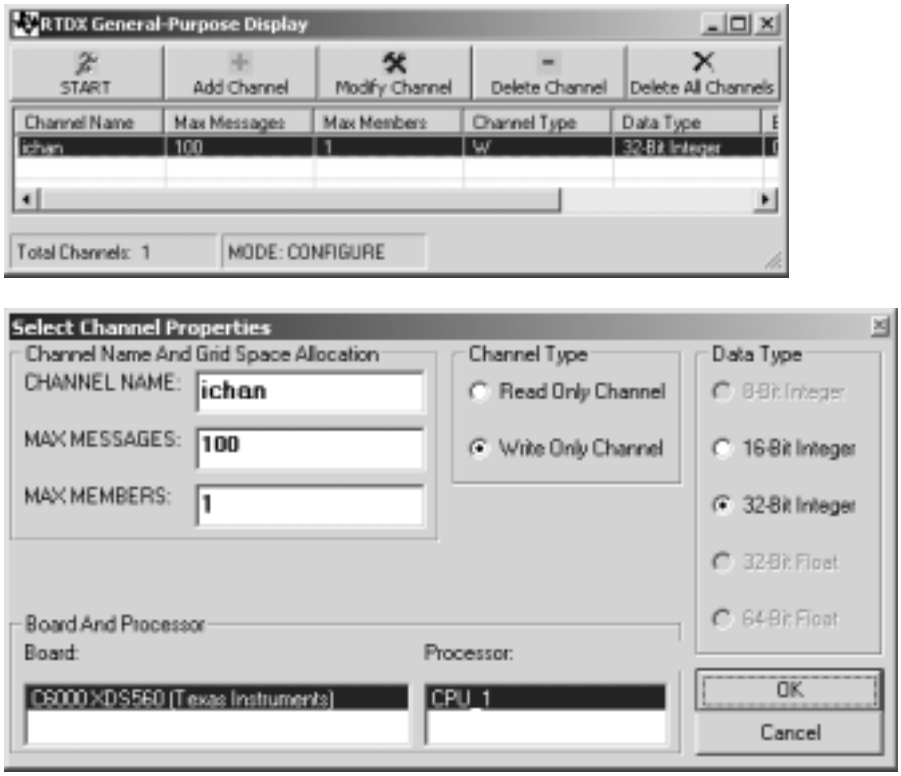

Figure 5. General-Purpose Display Setup

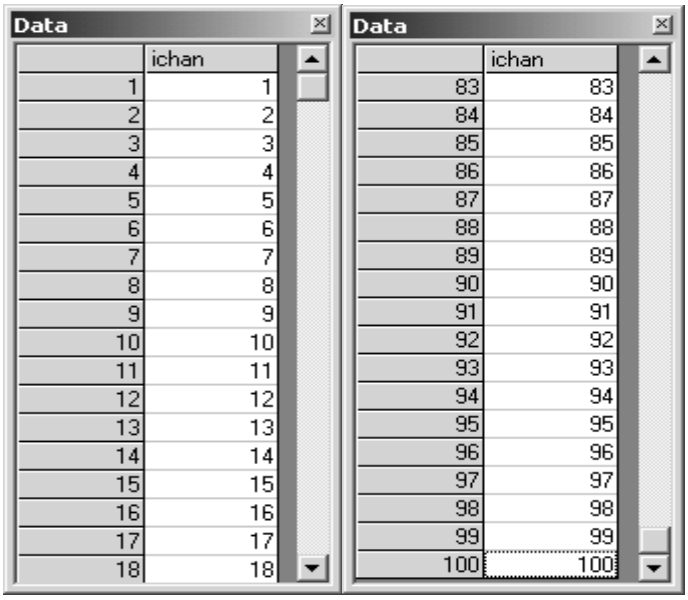

Figure 6. ht2 Test Results

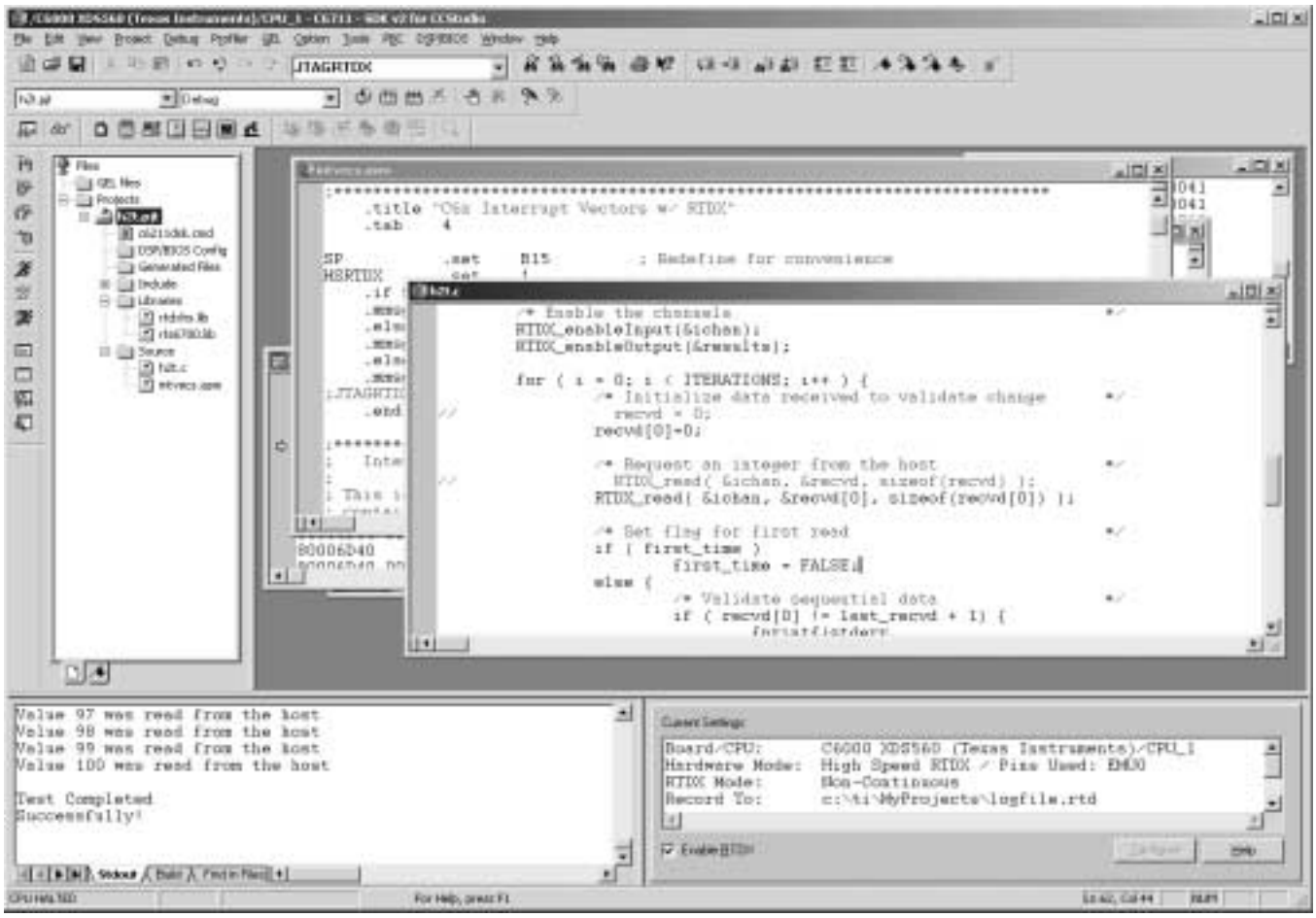

Figure 7. h2t CCS Test Results

7.2 Trouble Starting HSRTDX Transfers

If you are having problems getting HSRTDX data transfers started, try the following procedure:

1. Reset your target and bring up CCS. Before you can load code you need to have the EMIF programmed. You can do this with a GEL script if your reset code does not perform this function.
2. Load your code or symbols with CCS (File=>Load Program or File=>Load Symbol).
3. Bring up the RTDX Configuration Tool. Check that the Hardware Mode is HSRTDX and Pins Used is either EMU0 or EMU1 (if not you have not linked your code with the right library).
4. Enable RTDX by clicking on the RTDX Check box in the main RTDX Configuration Tool window (if RTDX is not enabled, RTDX communications between the host and target does not start).
5. Set a software breakpoint on the INT3 interrupt vector (0x00000060).
6. Run.

If the breakpoint is never taken, the RTDX process is not being enabled in the DSP by the emulator. Enabling RTDX in the DSP is the first RTDX transaction that takes place when execution is started. If INT3 is not being taken, the first thing to look at is to make sure the IER has INT3, INT11, INT12 and NMI enabled, and the GIE bit in the CSR is on.

Check the Interrupt Multiplexer Register. HSRTDX interrupts are not multiplexed to the core's interrupt inputs if the INTSEL field for interrupts 11 and 12 are not the reset value (int 11 – 01010, int 12 – 01011).

If you do not find a problem with the interrupt enable (IER or GIE) or Interrupt Multiplexer Register, then check if INT3 is being cleared inadvertently by your initialization code. We have seen cases where INT3 is posted from the emulator to the DSP shortly after execution is started, and then cleared by the DSP's interrupt initialization code. Avoid this problem by not clearing INT3 in your initialization code. We have seen this problem with BIOS applications (BIOS_init() function) when running at very slow functional clock rates (below 1 Mhz).

7.3 Data Corruption

If you have a problem with data corruption or if CCS hangs during a RTDX transfer, try reducing your TCLK rate. This can be accomplished with CCS setup.

8 Multiprocessor Notes

If you have a multiprocessor system you may select two processors with which to perform HSRTDX transfers. After you have brought up CCS for each processor through PDM, load your application code or symbols. The first processor whose code is loaded is automatically assigned to EMU0 and the second processor loaded is assigned to EMU1. The EMU pins are not available for triggers or benchmarking unless you use the RTDX Configuration Tool to:

1. Disable RTDX (clear the Enable RTDX check box)
2. Select Configure to bring up the RTDX Configuration Control Properties window

3. Select the Port Configuration Page
4. Select NONE for the configuration

This frees the EMU pin for other uses like triggering other processors to halt.

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third–party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Mailing Address:

Texas Instruments
Post Office Box 655303
Dallas, Texas 75265