Spotlight on: Automating testing with DSS Test Server

Introduction:

Automated testing has always been a critical component of the software development process. For users of Code Composer Studio (CCS), automated testing of their applications can be accomplished by using Debug Server Scripting (DSS). DSS is a set of cross platform Java APIs to the Code Composer Studio (CCS) debugger. This allows scripting through Java or scripting languages such as JavaScript.

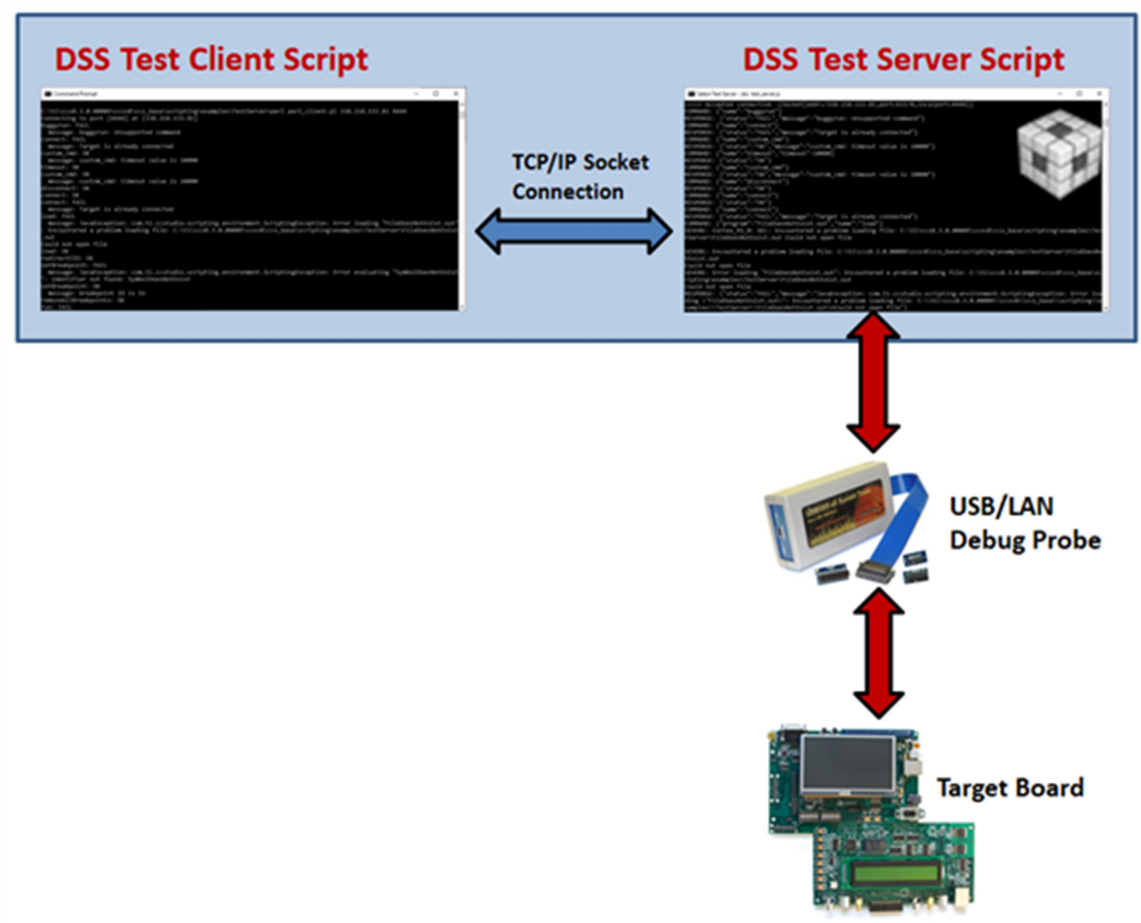
DSS comes with a variety of example scripts to help users get started. While most of these examples are simple self-contained scripts to demonstrate specific DSS functionality, some are more comprehensive examples that creatively highlight what you can do with DSS. These examples can function as useful "out-of-the-box" utilities. "loadti" is one of the better known examples, which is used by many customers as a convenient "out-of-the-box" command-line loader. Perhaps the least known example is the "DSS Test Server". This article hopes to bring more attention to this seldom used (but perhaps most intriguing) example.

What is the DSS Test Server?

The DSS Test Server was created as a raw "proof-of-concept" example for a customer who needed the ability to have a script be able to interface to an existing CCS debugger instance. The customer relied on configuring and launching the CCS debugger once, leaving it running indefinitely, while scripts would subsequently be run - each attaching to the existing debugger instance and performing the series of debug actions needed to carry out a particular test. Each script would finish execution, but leave the debugger up and running after completion for the later scripts to connect to. In DSS, the default supported behavior is to have each script launch a new debugger instance each time, perform all the desired debug actions, and terminate the debugger instance when the script completes execution. This would not satisfy the customer requirement so a new solution was developed with collaboration by several SDTO, BU, and field applications folks. That solution is how the DSS Test Server example was born.

Okay… interesting background… but still… what exactly is it?

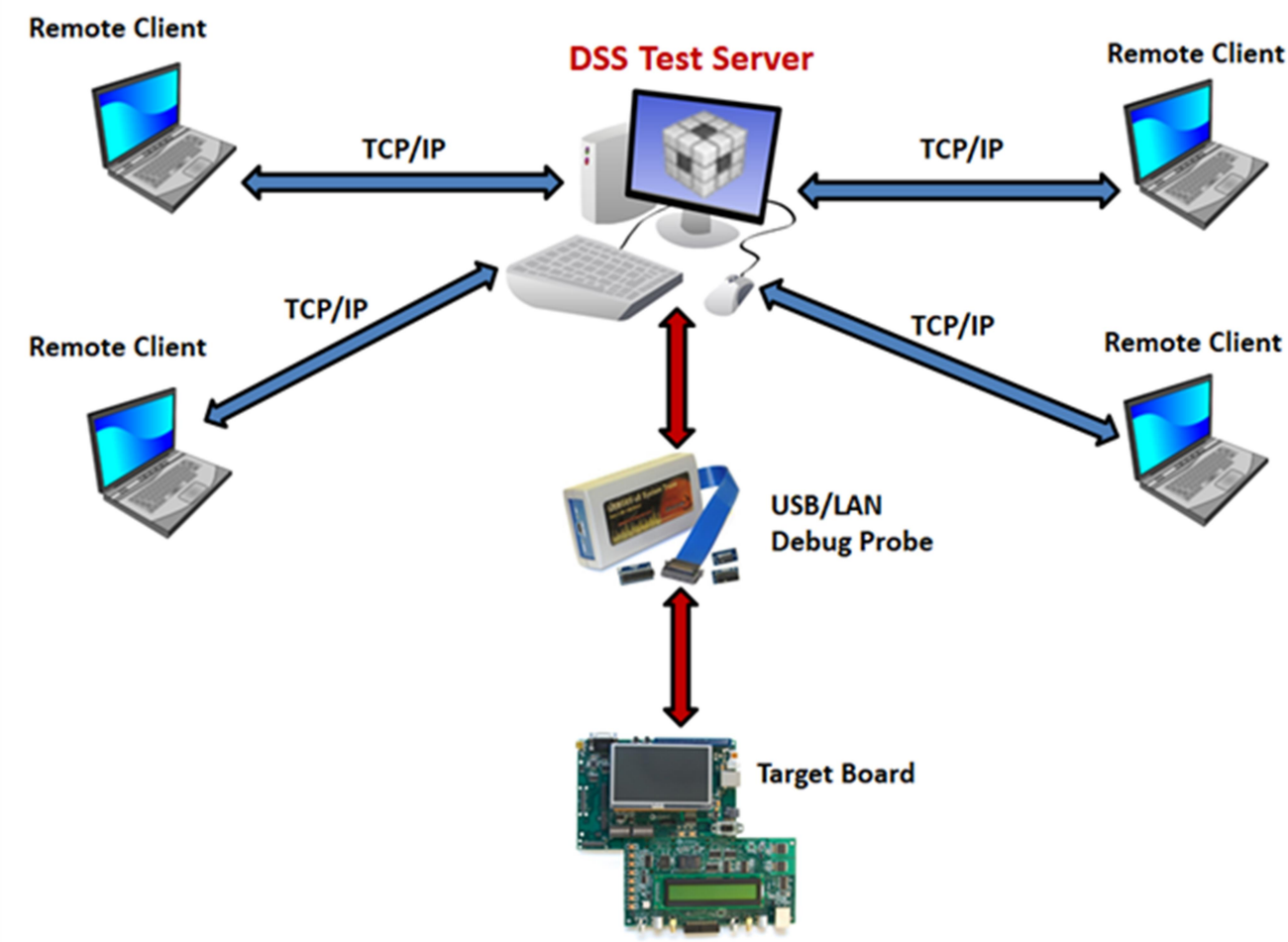
The example worked by having a main (DSS Test Server) script create a CCS debugger instance and start a debug session for each available CPU in its own Java thread. Each thread would then listen for commands on an open TCP/IP socket. Remote clients (via client script) can communicate to the active debug session by connecting to the socket and sending supported debug commands that are handled by the server script. The "remote" client machine can be the same as the host machine that is running the main server script ("localhost"). Once all the commands in the client script are executed, the Test Server would then close connection with the client, and then wait for the next set of commands to execute from other client scripts. This solution was able to satisfy the customer and enable them to use DSS.



How else can it help?

Remote Debug

While the customer (at the time) was running both client scripts and the main server script from the same machine, the DSS Test Server also provided an example of how to have truly remote clients, from a variety of locations, establish a connection to the test server and send debug commands to be executed.



An environment can be set up where the DSS Test Server can execute client scripts from multiple users in different remote locations. Why is this better than having remote users simply use CCS/DSS with a remote LAN debug probe connected to the target? The first issue would be the need for an external LAN debug probe in the first place, which can run a few hundred dollars for an XDS220 to a more than a thousand for an XDS560v2. Secondly, not all evaluation kits support an external debug probe setup. And finally, there are the added complexities of such a LAN debug probe setup. Even though a SimpleLink LaunchPad can support external debug probes, it is simply much easier (and cheaper) to just plug the

LaunchPad directly to the computer with the included USB cable and use the default on-board XDS110 debug probe. Then there is the issue of connecting to an existing running instance of the CCS debugger, which a direct use of CCS/DSS with a LAN emulator does not help with. Another benefit of the DSS Test Server is that the remote client machines do not require an installation of CCS/DSS to simply send commands to the Test Server (easy setup!).

Simplified and Optimized Test Environment

Other customers adopted the DSS Test Server for their Test Framework. The default DSS approach of self-contained DSS scripts resulted in large number of complex scripts to cover many use cases. There was quite a bit of overhead in re-configuring and re-launching the debugger each time. Using the DSS Test Server helped alleviate some of these issue. The Test Server (along with the CCS debugger) only needs to be relaunched in cases where the connection changes, hence this overhead is mostly gone. Client scripts no longer need to configure and launch the debugger each time, but instead can connect to a running test server and immediately access the target right away. Creating a new client script to access the target is much simpler, without the need for any DSS knowledge.

While the customer Test Framework currently runs the client scripts on the same machine as the test server, another benefit for remote debug environments is the simplified setup due to the lack of dependencies on CCS/DSS by the remote client machine (as mentioned in the previous section).

Summary

The DSS Test Server has been part of a suite of DSS examples found in CCS for years. Often overlooked, it has gotten praise from the people who have taken the time to explore its capabilities and integrate it into their test environments. While people are encouraged to take this fairly “raw” example and modify/enhance it to fit their needs, it can also be used “as-is” right “out-of-the-box”. Check it out!

References:

DSS Test Server: http://software-dl.ti.com/ccs/esd/documents/dss_test-server.html