Socket Communication on the XPS using TCL

BACKGROUND

Applications development on the XPS with TCL is simple, convenient and powerful. The XPS web interface terminal utility provides a full list of commands that can be selected for execution. The XPS command set is not abbreviated and the full power of applications development is in the hands of the user when developing with TCL. These commands include functions related to Digital and Analog I/O and consequently interfacing to external devices as well.

Commands Executed in the web interface terminal are appended to a list that can be used to automatically generate a TCL script. TCL scripts are stored locally on the controller hard drive and can be executed directly on the controller or transferred to the control PC via FTP. Scripts executed on a control PC will require a TCL interpreter. TCL interpreters are widely available for many Operating Systems and Newport provides a TCL interpreter with the XPS for Windows.

TCL scripts can be edited easily on the control computer with any text editor. A popular and convenient text editor for TCL scripts is Notepad++. Notepad++ will format the text with color coding. In the text editor additional structures or comments can be added beyond executed commands. There are many excellent resources online to learn TCL as well.

While TCL is an excellent resource for efficient applications development with the XPS, it is not without limitations. It is important to understand these limitations and operation when starting development with TCL.

OPENING A SOCKET

The XPS communicates with TCP/IP sockets. Each socket connection is established with an IP address, port number, timeout and unique identifier. A command to open a socket is as follows:

TCP_ConnectToServer 192.168.254.254 5001 120 SocketID

Here:

1. The IP address of the XPS is: 192.168.254.254
2. The Port # of the XPS is: 5001
3. The timeout is 120 ms
4. SocketID is the unique identifier

When opening a second socket, the only change to this command would be the socket identifier to preserve the unique naming. Each socket would then be addressed with the respective naming established with the connection. For example, we issue the firmware version get command and we format as follows (to make this request over socketID):

FirmwareVersionGet $socketID version

BLOCKING SOCKETS

The XPS uses blocking sockets for communication. The practical impact of blocking sockets is that an executed command will prevent other sequential commands from executing until the controller has sent a response indicating completion of the command. The XPS provides a numeric response to command execution to indicate successful completion (zero) or an error whether an error has occurred (nonzero response). The type of error will have a specific numeric code that can be queried.

Blocking sockets are quite useful for sequential command execution when preserving ordering. Blocking sockets also make it easier to identify the cause of errors, since each command will have a numeric response. However, blocking sockets can be inconvenient for execution that requires rapid timing. For example, we issue a command such as:

GroupHomeSearch $SocketID X

This command may require seconds to complete as a stage searches for home position. This means any commands issued sequentially after this command will be delayed by seconds until the controller has sent a response. As such we often want to use multiple sockets for communication.

MULTIPLE SOCKETS

The XPS supports up to 80 sockets open at any given time with 30 actively communicating. When using multiple sockets it is important to understand the impact of execution order along with timing. Let's consider multiple sockets communicating with two different stages (Group1, Group2).

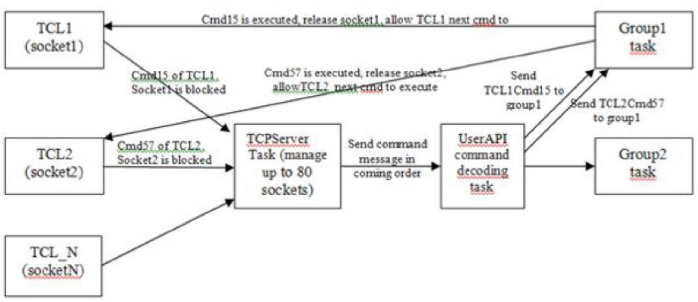
We can reference the below diagram to explain command execution over two sockets.

1. We send sequential commands from multiple sockets in the following order:

   TCL1 (socket 1), TCL2 (socket2)…TCLN (socketN)

Socket Communication on the XPS using TCL

2. TCL1 is blocked because execution of Cmd15 has not completed
3. TCL2 is blocked because execution of Cmd57 has not completed
4. TCLN is sent to the TCP Server Task Manager for queuing and decoded and sent to Group2 for execution. This command is executed first on Group2
5. Cmd15 has completed execution and the socket is released
6. TCL1 is the next command to execute on Group1 and the socket is blocked again
7. Cmd57 has completed execution and the socket is released
8. TCL2 is the next command to execute on Group1 and the socket blocked again

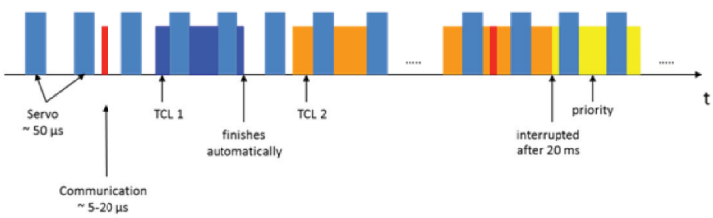


In summary, the command sequence will depend not only on execution order but when sockets are released. Commands sent to sockets that are not being blocked are executed first in order (FIFO) and as sockets become released so do additional commands execute in order of release. Now we have set forth the basics of ordering we considering the timing related to execution

SOCKETS AND COMMAND EXECUTION TIMING

TCL command execution is assigned as a low priority task by the XPS; this means that other motion related tasks are given priority (Corrector, Profiler, etc…). Thus, TCL commands are put in a message queue and executed progressively depending the CPU loading. This means that TCL command execution can have variable timing depending on the current CPU load and pace at which multiple commands are executed.

To illustrate this point, we consider multiple TCL commands sent for execution to the XPS. Below is a graphic illustrating TCL timing on a relative scale with servo and communication timing. In this scenario the servo loop rate is 20 kHz (default for XPS is 8 KhZ) and communication requires 5-20 microseconds. Here we execute TCL 1 command first, which is a relatively short command that completes quickly. We then execute TCL 2 after a brief delay. The command execution for TCL 2 however requires substantially more time and is interrupted by a higher priority task.



Consider repeated execution of TCL 2 with minimal delay; when controller CPU loading is light, the command executes without interruption. When the CPU loading is greater, the command will be interrupted and execution delayed. This is a by-product of the task priority assigned to TCL and will result in variable timing for rapid sequential commands. If commands are sent in very rapid succession this puts additional load on the CPU and likewise timing delays associated with resource management.

If regular TCL command execution timing is important, it is best to introduce delays (after command) in the TCL script to slow the frequency at which commands are sent. If best TCL timing is critical, try to reduce CPU loading by minimizing other tasks running.