xPC Target communication efficiency when multiple targets are involved

Mădălin Mătăsaru*

**University of Craiova, Department of Automatic Control, 200585-Craiova*
Romania (Tel: +40-251-438198; e-mail: madalin@automation.ucv.ro)

Abstract: Most studies that involve xPCTarget as a real time operating system in charge of executing a control task are focusing on the built in control and simulation capabilities. In this paper we are trying to see how accurate and how fit xPC Target is for multiple Targets control. We will try to see how the percentage of lost packages is influenced by the number of Target PCs connected to the master and also by parameter tuning (a desynchronization between master and Targets). Also we will use real experimental data to see what's the limit for the communication and response time to be reliable. Here the xPC Target is used as an operating environment for real time processing and to create a computer network system used for remote control.

Keywords: xPC Target, Matlab, Package Loss, Target Control, Capture packages, Error rates

1. INTRODUCTION

National Instruments and The MathWorks, Inc. make PC real-time simulation possible with the use of real-time kernels. The use of real-time simulators is increasingly becoming more popular as hardware-in-the-loop prototyping has proven as a reliable design tool, a decrease in time to market products, and as a safe low-cost experimental alternative to potentially damaging equipment.

In this work, the real time kernel xPC Target is used to determine the feasibility, accuracy, and determinism of bootloaders as an alternative to costly real-time simulators for smaller systems. Implications and modeling methodologies of simulating the power system in real-time using xPC are presented providing a discussion at the end of this paper on how communication between master PC and Targets is affected by different parameters (like sampling rate, generated signal frequency and others).

To define the communication network performance the use of several targets is necessary (Figure 1).

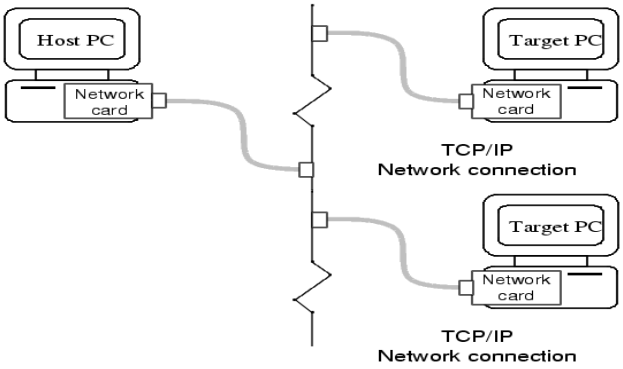

Figure 1. System Implementation

This strategy achieves network implementation based on message passing, which uses Real-Time Workshop, xPC Target and Simulink toolboxes from MATLAB by MathWorks Inc. Only one kind of local area network technology is used: Ethernet.

To study the number of sent packages we created a small application that monitors the packages sent through TCP\IP Protocol.

Basically the application starts a session and uses 4 variables: sourcePort, sourceIP and destPort, destIP to monitor communication. It displays only the IPs for the target and the sender (Figure 2).

In order to make sure that our application works correctly and the results are accurate we will compare our output against the output acquired by using Microsoft Network Monitor v3.4 (Figure 3).

This is a more complex application that captures absolutely all packages (not only TCP but also DHCP, ARP, HTTP, DNS, etc.) but allows us to filter data after almost any criteria: address, protocol used, etc.

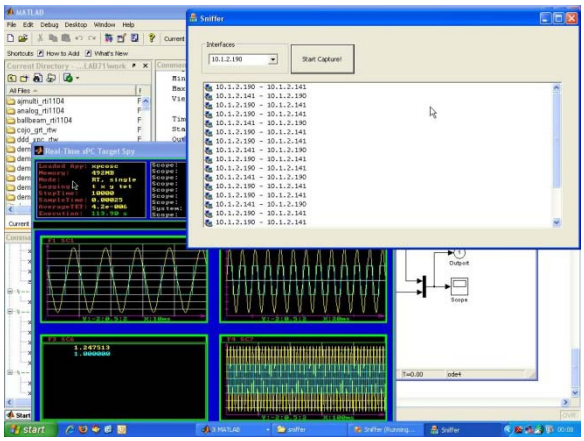

Figure 2. Sniffer application

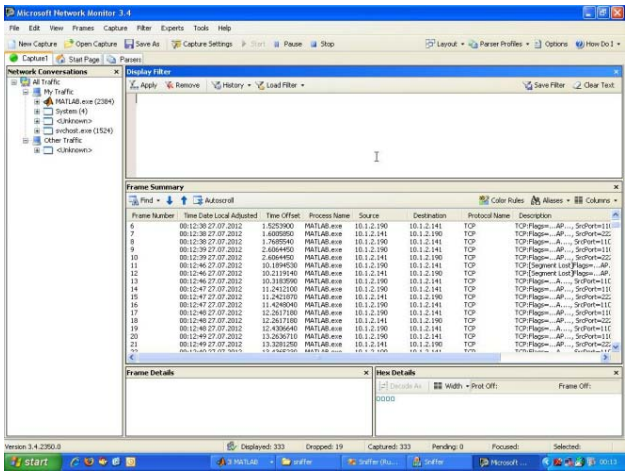

Figure 3. Microsoft Network Monitor

Three computers (one host and two targets) and a compatible data acquisition card are required to provide the interface between the software and the hardware to be controlled (a nonlinear, open loop unstable and time varying device). Measured signals can be displayed in real time or saved for further analysis.

Several strategies for managing time delay within control laws have been studied by different research groups. For instance Nilsson proposes the use of a time delay scheme integrated to a reconfigurable control strategy based upon a stochastic methodology. On the other hand, in (Wu, 1997) a reconfiguration strategy based upon a performance measure from a parameter estimation fault diagnosis procedure is proposed. Another strategy has been proposed by Jiang et al. (1999) where time delays are used as uncertainties, which modify pole placement of a robust control law.

In (Izadi-Zamanabadi and Blanke, 1999) it is presented an interesting view of fault tolerant control approach related to time delay coupling. Reconfigurable control has been studied from the point of view of structural modification since fault appearance as presented by Blanke et al. (2007), where a logical relation between dynamic variables and faults are established. Alternatively reconfigurable control may perform a combined modification of system structure as is studied in (Benítez-Pérez and García-Nocetti, 2005) and (Thompson, 2004).

Another technique like gain scheduling (Khalil, 2002) may give an interesting approximation to several time delay scenarios, however complexity related to system modelling during fault conditions is out the scope of this paper.

Some considerations need to be stated in order to define this approach. Time delays are bounded and restrictive to scheduling algorithms. Global stability can be reached by using classical control strategy for online time delays.
The control algorithm was designed and implemented by the Matlab/Simulink software (Release 14) with Real-Time Workshop and xPC Target Toolbox. These tools are able to automatically generate stand-alone real-time applications from the Simulink models that run on the so-called Target PC, while their development is carried out on a separate host computer.

The paper is organized as follows. In Section 2 xPC Target application and how we are going to use it is presented. The experiment's setup in a step by step manner is described in Section 3. Section 4 presents how we have to run our experiment in order to get correct experimental data. Finally, Section 5 concludes the paper.

2. xPC TARGET

xPC Target is a host-target solution for prototyping, testing, and deploying real-time systems using standard PC hardware. It is an environment that uses a target PC, separate from the host PC, for running real-time applications.

In this environment we use the desktop computer as a host PC with MATLAB®, Simulink®, and Stateflow® (optional) to create models using Simulink blocks and Stateflow diagrams. After creating our model, we can run simulations in real-time.

xPC Target allows us to add I/O blocks to our model, and then use the host PC with Real-Time Workshop®, Stateflow Coder (optional) and a C compiler to create executable code. The executable code is downloaded from the host PC to the target PC running the xPC Target real-time kernel. After downloading the executable code, we can run and test your target application in real-time.

To accomplish our goals we will use the oscillator model, called xpcosc.mdl to demonstrate signal tracing with target scopes. Target scopes are used to trace or display signals on a video monitor attached to the target PC.

After building and downloading the model to the target PC, four scopes of type 'target' are added to the application; each scope having a different acquisition mode.

The four scopes are identified by the following scope numbers: 1, 3, 6, and 7. The signals 'Signal Generator' (oscillator input) and 'Integrator1' (oscillator output) are added to and displayed on each scope. The duration of the simulation can be fix or it can be set to run permanently.

The idea is that the master PC and target PC exchange data permanently and we can track the number of sent/received packages.

3. EXPERIMENT SETUP

As we said before we need one computer to be used as master. We don't need anything special just a Intel Ethernet card (we use INTEL PRO 100 S), a FAT 32 formatted hard disk and Matlab with xPC target module installed. One more thing we need is that these 3 PCs (master and 2 slaves) to be connected through a Ethernet network. Using the master PC we will create a startup disk for the first target PC: we need to type 'xpcexplr' in Matlab main screen and in the new window choose Communication (under the first target PC) with options

TCP\IP for communication type and all needed information to identify the target PC (Figure 4).

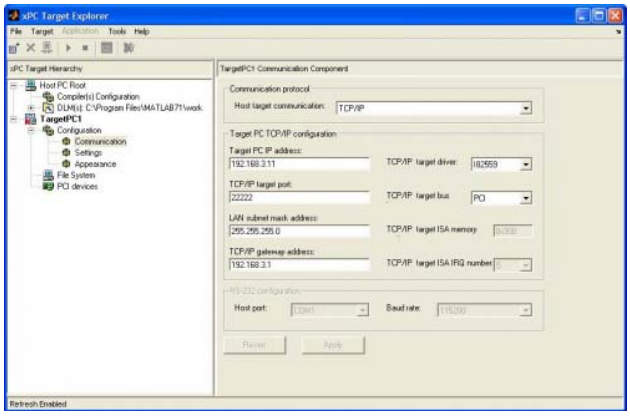

Figure 4. xPC Setup

On the tab Configuration we select BootFloppy as a Target boot mode, we insert a floppy disk and we write the xPC Target operating system kernel. Using the new created disk we boot the first target PC. The procedure is repeated for the second Target PC using the corresponding IP address.

After the setup is complete we run *xpcexplr* and we connect to each Target PC.

The next step is running *xpcosc* command to load the model (Figure 5) and then after setting Simulation option to External we run Incremental Build. We need to create a *m file* where we set the four scopes and then Run the *m file*. By running the command *xpctargetspy* we can actually see the output on the Target Pcs (Figure 6).

After the communication was established and the Host PC transmits data to the Target PCs every millisecond we can start our tracking applications. We already established that we are going to use Microsoft Network Monitor v 3.4 and an application developed with the specific purpose of displaying the packages sent and received by our Host PC.

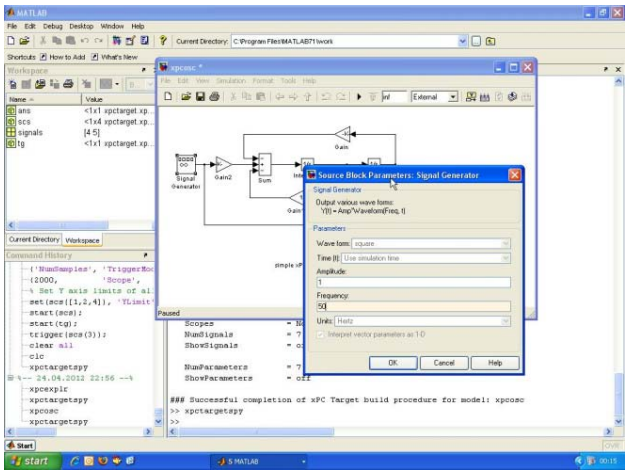

Figure 5. xpcosc.mdl

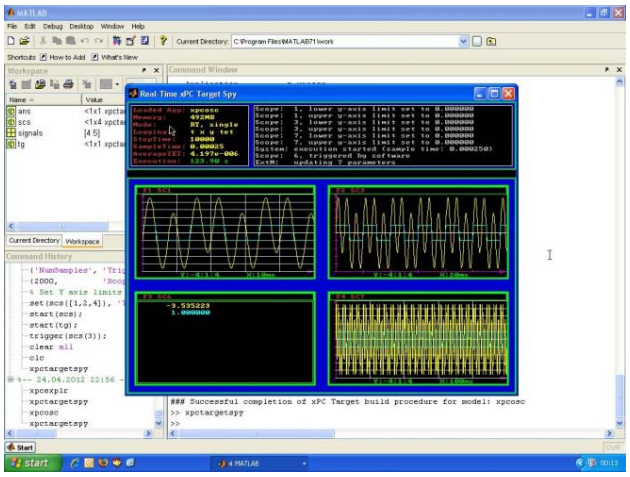

Figure 6. xpctargetspy command output.

4. RUNNING THE EXPERIMENT

After everything is set up and we start sending/receiving packages we can easily track them. As it can be seen in Figure 7 the average number of total packages sent and received in one second is about 257 packages when using only one target PC.

When a second PC is inserted into the network the number of packages sent (for real) drops to an average of about 203 packages per second. This is also because collisions appear. We must be aware that UDP protocol does not guarantee the order of packages transmission or that they reach the destination, and also doesn't verify network load.

In our experiment we have to keep in mind the fact that the packages counted at return have travelled through the network 2 times and that the delay is bigger than the case in which we would measure only the time they spend travelling from one computer to another.

It is also highlighted here, from the point of view of the approached subject, that the response time of a system in a network under which data collecting, the command and processing of the system are made in different places and not in the same place in which the actuators are held.

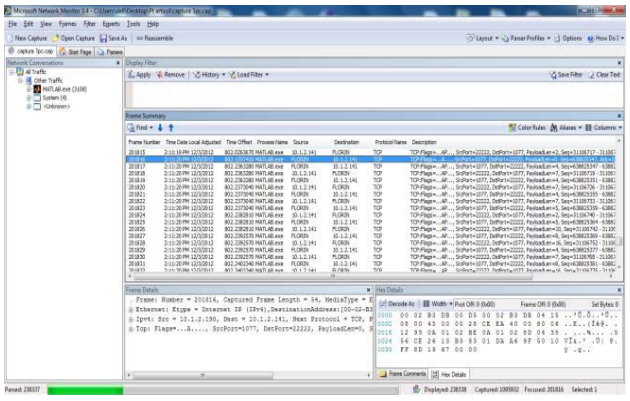

Figure 7. Microsoft Network Monitor v3.4 output for one xPC target

In different cases, after modifying the value for sampling rate in the program used for testing we get the next values:

1) For a sampling rate of **10ms** and same value for the update/change time of the input (generated signal) for both programs running we get a percent of lost packages of ~14% (Figure 8).

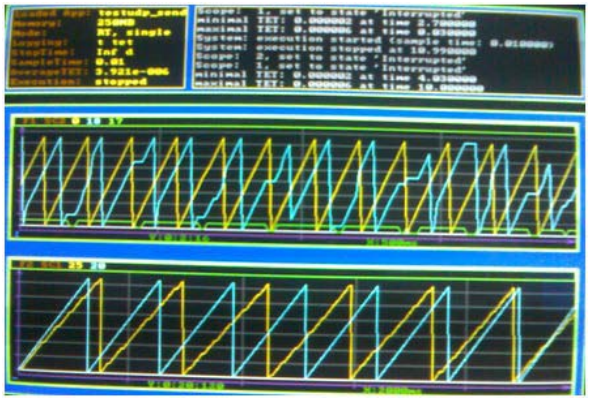

Figure 8. Output for a sampling rate of **10ms**

As the figure shows, in the upper section is represented the generated signal and the signal received back from the network, and in the lower section we have the values of the counters.
In this case the ratio between sent and received packages is almost 7 to 6. The number of displayed samples here is 1000.

2) For a sampling rate of **100ms** and same value for the update/change time of the input (generated signal) for both programs running we get a percent of lost packages of ~0% (Figure 9).

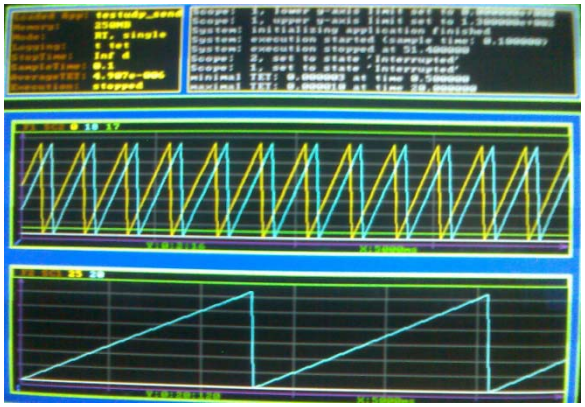

Figure 9. Output for a sampling rate of **100ms**

Due to the fact that no losses were recorded, on the second target scope the lines are not overlapping. The number of displayed samples here is 300.

3) For a sampling rate of **1ms** and same value for the update/change time of the input (generated signal) for both programs running we get a percent of lost packages of ~66% (Figure 10).

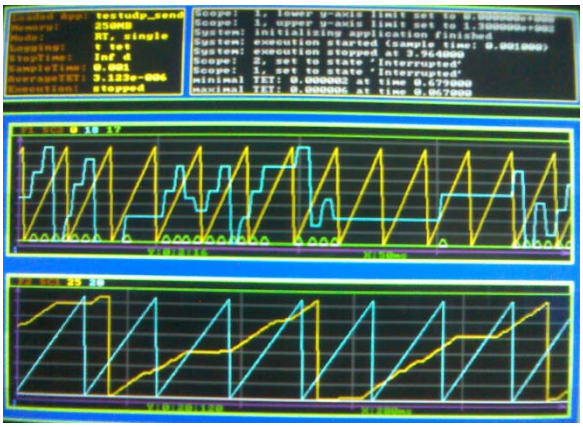

Figure 10. Output for a sampling rate of **1ms**

It seems that the number of sent packages is almost 3 times larger than the number of received packages. The number of displayed samples here is 1000.

4) If we change the sampling rate and we choose different values for the sending and receiving components of our system, more precisely a greater time for the sampling rate of the sender and a smaller time for the receiving component **10ms sent**, **1ms receive;** we get a small percent of losses: **~5%** (Figure 11).

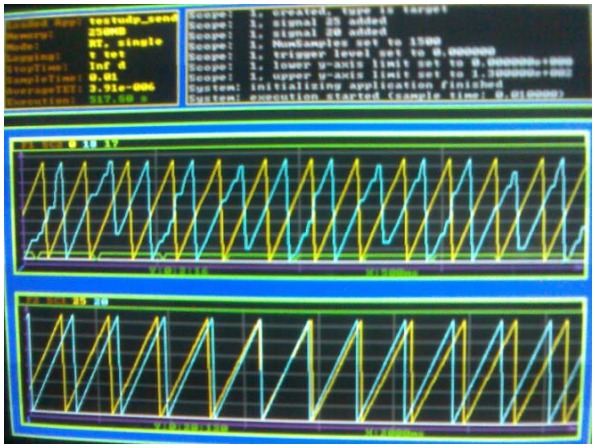

Figure 11. Output for a sampling rate of **10ms sent**, **1ms receive**

In this case the ratio between sent and received packages is almost 19 to 18. The number of displayed samples here is 1500.

5) For **10ms sent** and **9ms receive** the recorded loss rate is **~0,6%** much smaller then 10ms for both. (Figure 12).

In this case the ratio between sent and received packages is almost 164 to 163. The number of displayed samples here is 1500.

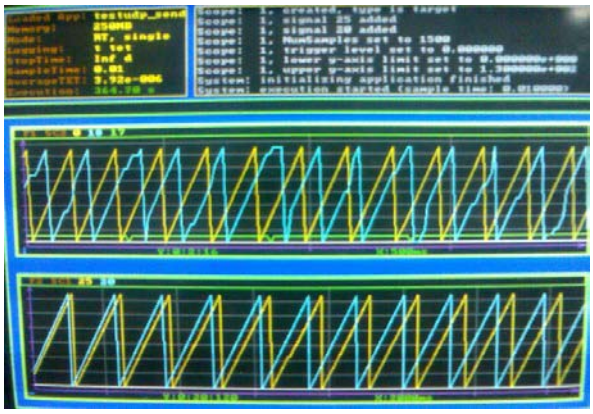

Figure 12. Output for a sampling rate of **10ms sent**, **9ms receive**

5. CONCLUSIONS

An obvious conclusion would be related to the direct connection between sampling rate values and the percentage of lost packages: when the sampling rate value increases the percentage of lost packages decreases.

This is somehow normal because there are more packages being sent when the sampling rate has a small value and there is a greater network load thus the chances of package loss and delay increases.

An interesting result is the great difference in the value of package loss percentage between setting both sender and destination at a rate of 10ms and a sampling rate of 10ms at the sender side, greater than 9ms set for the destination side (it will receive more often). A conclusion that can be drawn from here it is that it helps a lot to have the destination end receive more often than the sender side can transmit.

The most dramatic case, in which the rate of lost packages is about 66% is when the value of sampling rate is very low (1ms). This can be explained by a high package arrival rate and low processing speed.

Future work shall be focused on repeating the experiment using a greater number of PCs and see which is the maximum number supported. We will change the network topology and see how it does affect package transfer itself. In the end a comparison between xPC Target and other real-time software environments will be made in order to see which can handle more Targets in the most accurate way.

ACKNOWLEDGMENT

This work was supported by the strategic grant POSDRU/89/1.5/S/61968, Project ID61968 (2009), co-financed by the European Social Fund within the Sectorial Operational Program Human Resources Development 2007-2013.

REFERENCES

Benítez-Pérez, H. and García-Nocetti, F. (2005). Reconfigurable Distributed Control. Springer Verlag.

Blanke, M., Kinnaert M., Lunze J., and Staroswiecki M. (2003). Diagnosis and Fault Tolerant Control. Springer Verlag.

Izadi-Zamanabadi, R. and Blanke M. (1999). A Ship Propulsion System as a Benchmark for Fault-Tolerant Control. *Control Engineering Practice*, vol. 7, pp. 227-239.

Jiang, J. and Zhao, Q. (1999). Reconfigurable Control Based on Imprecise Fault Identification. *Proceedings of the 1999 American Control Conference*, San Diego, June, 1999, vol. 1, pp. 114-118.

Khalil, H. (2002). Nonlinear Systems. Third Edition. *Prentice Hall*.

Nilsson, J. (1998). Real-Time Control with Delays. *PhD. Thesis, Department of Automatic Control, Lund Institute of Technology*, Sweden.

Thompson, H. (2004). Wireless and Internet Communications Technologies for monitoring and Control. *Control Engineering Practice*, vol. 12, pp. 781-791.

Wu, N.E. (1997). Reliability of Reconfigurable Control Systems: A Fuzzy Set Theoretic Perspective. *Proceedings of the 36th Conference on Decision & Control*, San-Diego, USA, 1997, vol. 4, pp. 3352-3356.