Event handling using EPICS variable with LabVIEW

Aniruddh Mali
M.E Student, EC Department
HGCE
Vahelal, India
aniruddhkmali@gmail.com

Ramesh Joshi
High Power ICRH Division
IPR
Gandhinagar, India
arjoshi07@gmail.com

***Abstract*— The Event Structure only registers events generated by direct user interaction with the front panel. In this example, the indicator value is changed programmatically, thus LabVIEW does not capture the Value Change event. Furthermore, Value Change events do not occur if you use VI Server, global variables, local variables, etc. to change VI or front panel objects programmatically.**

There is one exception: Events for controls and indicators can programmatically be generated with a property node, using the property Value (Signalling). This write-only property creates an event that acts like a front panel value change each time it is written. Beware of using this property in a loop, however, because if it is written in each iteration of the loop, it will generate a value change trigger for each iteration, regardless of whether the value was actually changed. We have requirement to handle the event based on value change from outside of LabVIEW variable. We handle LabVIEW event structure based on EPICS process variable change event. It has been successfully integrated with the system and tested with the system.

Keywords*— *LabVIEW, event, EPICS

I. Introduction

EPICS is a highly configurable toolset for building distributed control systems that scale to accommodate large projects[1]. It has C and C++ interfaces for the integration of new hardware and software[2], full source code is available.

While this provides the best performance, highest flexibility and is easily understood by experienced programmers, the initial EPICS setup does already require a network connection and two computers: the real time target and a Unix or Win32 (Windows NT, 9x, 2000) host. Some application engineers who are unfamiliar with the multifaceted EPICS toolset prefer to start with a purely visual environment on a single PC. LabVIEW is a tool where the engineer most familiar with the application task can quickly start the implementation. This paper presents ways of integrating LabVIEW into the distributed EPICS environment.

II. EPICS Channel Access

EPICS communicates via the Channel Access (CA) network protocol. Front-end input/output controllers (IOCs) run a CA server, presenting values as well as time stamps, limits, units, alarm status and other attributes. CA clients locate the server based on channel names. They establish a connection and subscribe to changes in value or connection status. Management of the connection status as well as high throughput are key features of CA [3, 4]. CA server and client libraries are available to C/C++ software on Unix and Win32. Since the CA libraries need to monitor network connections for incoming requests and data, the user program has to implement periodic calls into the CA libraries. The CA server monitors a dedicated UDP port for search requests. It is therefore suggested to run only one CA server per computer since an additional server would use a nonstandard UDP port, unknown to most CA clients.

A. Labview Extension Options

LabVIEW can call Win32 DLLs, communicate via ActiveX and DDE or perform low-level UDP/TCP network calls. Using the latter would result in a re-write of the CA libraries. Since LabVIEW code is unlikely to compete with a C/C++ implementation, this was not attempted. Using the CA libraries as DLLs is problematic because LabVIEW would have to initiate the periodic network processing. DDE has been used for the CA client library: A separate program implemented the periodic processing of CA network connections, presenting the data to MathWorks MATLAB via DDE[2]. LabVIEW can use this DDE interface, but DDE is deprecated with the advent of newer technologies, namely Win32 COM (Component Object Model) and ActiveX [5].

B. CA Support

Several LabVIEW VIs shield the user from the underlying COM calls. Serving a number is reduced to one initial call to a "Create" VI that takes the name of the new process variable, followed by calls to a "Set" VI whenever the value changes, see Fig. 1. In this example LabVIEW serves a read-only process variable to EPICS clients. Fig. 2 shows a more realistic setup as suitable for a setpoint, a variable that is to be changed both locally on the LabVIEW front panel and remotely via a CA client. After creating a process variable for the setpoint, additional informational parameters are configured and then the value of a user interface knob on the front panel is served. In addition, LabVIEW polls for input from CA and modifies the value of the knob in response. While other languages can asynchronously react to ActiveX events, invoking callbacks immediately after the event arrives, LabVIEW offers only a polling or waiting mechanism to check for events.

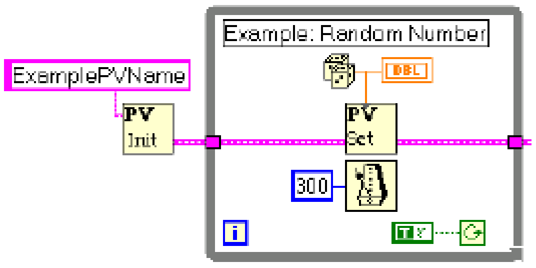
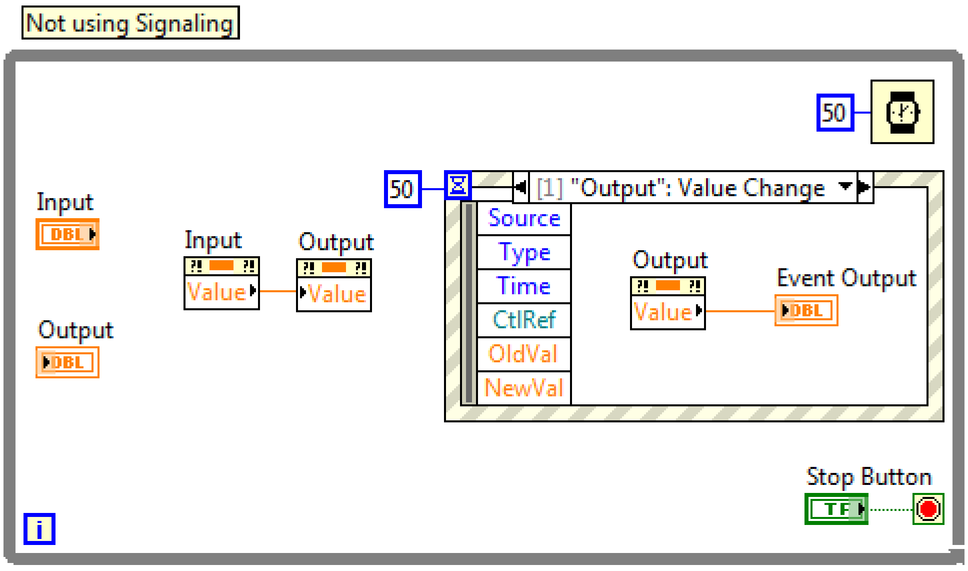
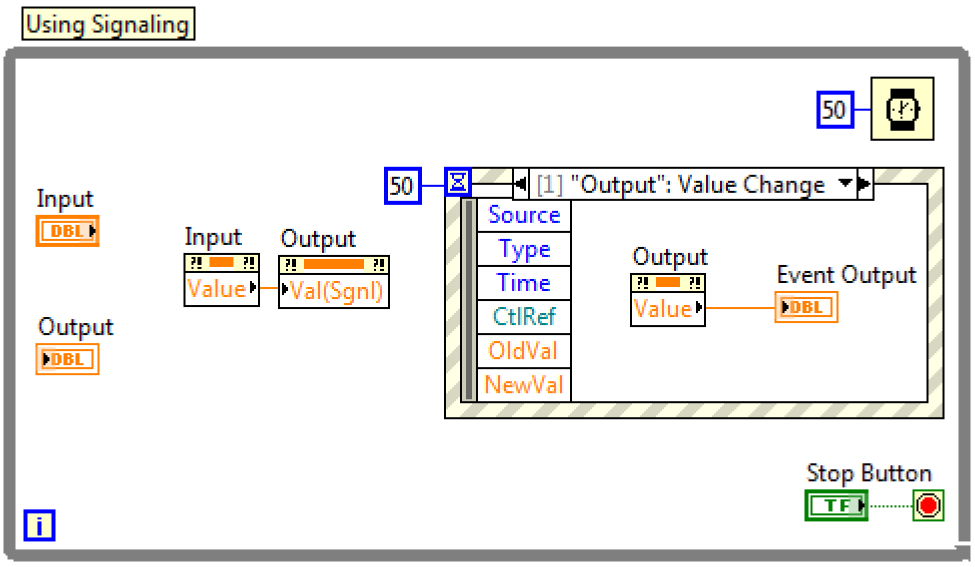


Figure 1: Serving a random number from LabVIEW

See the figures below and the attached VIs for a demonstration.





In both VIs, the value of the control "Input" is written programmatically to the indicator "Output." Both VIs also have an event structure meant to capture a value change event of the "Output" indicator. This event structure updates a third indicator, "Event Output", with the value of "Output."

In the first VI, the value of "Output" will follow the value of "Input", but "Event Output" will not follow the value of "Output." This is because the programmatic change to the value of "Output" does not trigger the event structure. In the second VI, however, "Event Output" will be updated to the correct value because the Value (Signalling) property was used.

Conclusion

It must be noted that the performance of LabVIEW systems is highly dependent on the specific implementation; we exemplified this with regard to setpoints and reaction times. And while EPICS IOCs have a known, reasonable degradation with network load, where the system will stop responding to network requests but still perform local control, this is not possible with a LabVIEW/ActiveX approach because each COM call is a round-trip request. When the ActiveX CA server suffers from heavy network load, the LabVIEW program will degrade accordingly.

References

[1] L.R. Dalesio at al, “EPICS Directions to Accommodate Large Projects and Incorporate New Technology”, ICALEPCS 99, Trieste, 1999.

[2] J.O. Hill, K.U. Kasemir, J.B. Kowalkowsky, “Integrating Commercial and Legacy Control Systems with EPICS”, PAC’97, Vancouver, 1997.

[3] J.O. Hill, “Channel Access: A Software Bus for the LAACS”, ICALEPCS ‘89, Vancouver, 1989.

[4] J.O. Hill, “EPICS Communication Loss Management”, ICALEPCS ‘93, Berlin, 1993.

[5] Microsoft Platform Software Development Kit”, part of Microsoft Visual Studio 6.0, April 1998.