Convergence Instrument's

Development to Deployment Code Instrumentation Architecture

October 14 2010

Bruno Paillard

1 Introduction

Many modern instruments feature Personal Computer connectivity. A PC connection allows features that a stand-alone instrument cannot provide, such as data upload/download, improved user-interface for instrument management, instrument automation… etc.

Irrespective of the physical aspects of the connection, the PC connectivity is usually achieved through the use of an ad-hoc protocol that is specific to the instrument. This simple approach is easy to implement, but it has a few shortcomings, especially during the development phase of the instrument. For instance it only provides a limited set of functions. Adding functions to this set usually requires a redesign of the software on the PC side, but also on the instrument side.

In contrast all the PC-connected instruments at Convergence Instruments rely on the *Development to Deployment Code Instrumentation* interface (*DDCI*). This interface is realized using a software layer running in the instrument, as well as one on the PC. Together these two layers implement an extremely versatile communication function that can be used in all the phases of the application life-cycle from the development and debugging through production testing and finally to the user application.

2 The DDCI Concept

When developing PC-connected instruments that include complex embedded processing the traditional approach is to use two separate communication links:

- A specialized debugging link (emulator pod or the like) is used during the development phase. This link provides a tremendous amount of low-level visibility into the executing code, but it usually interferes with the real-time operation of the instrument. For instance the processor often has to be stopped before variables or registers can be observed. In addition it does not provide high-level analysis capabilities. It is a great tool to know if a variable is written properly, but not so good to analyze the frequency content of a stretch of signal.
- The application-level communication link is used to support connectivity at the application (user) level. It provides a limited set of functions that the instrument requires for its operation. These functions all need to be designed-in on the PC and on the instrument as part of the development process.

What the *DDCI* architecture provides is a unique communication channel that serves both purposes seamlessly. In so doing the *DDCI* architecture eliminates many of the tasks that are required to go from the development phase to the deployed application phase, thereby accelerating the development of the instrument.

- During development it provides real-time debugging at an application level that is usually not achievable using standard debugging and emulation techniques. In particular reading, writing and code-control functions do not require CPU halt. The interface allows the code to be instrumented in real-time and in the real operating conditions with very little overhead.
- After application deployment, the same interface is used to support user-control and communications with the instrument via the same set of functions used during the development phase.

The fact that the same functions and libraries are used in both phases of the application life-cycle insures that there is no additional development to support the instrument in the deployed phase. No additional development also means no unexpected behavior associated with this added development.

The development of the *DDCI* interface represents one of the initial steps that we take when we develop a new series of instruments. The *DDCI* interface is not specific to a particular instrument, but rather is designed for the processor platform that the instrument series is based on. This initial step represents a fair amount of work, but this cost is gained back many times over in the acceleration it provides to the development of the instrument.

3 Code Instrumentation

The *DDCI* interface consists in a powerful set of basic functions that provide visibility into the code running on the instrument's processor, in real time, while the instrument is processing real data.

- RAM (volatile) memory read and write: Any variable can be observed or written by the PC in real time.
- Flash (non-volatile) memory read and write: Useful for reading or writing calibration data, field-firmware upgrades…etc.
- Peripheral registers read and write: Useful for special hardware configuration, during debugging, production testing…etc.
- Force code execution (dynamic redirection): Allows the PC to take control of the instrument at a very low level.

Because these functions are always enabled, both in the instrument's CPU and on the PC, adding new features to the instrument's management software does not always require an instrument firmware upgrade. For instance adding a function to clear an existing variable in the instrument's firmware (for instance a variable containing the result of ongoing statistics being measured by the instrument) can be done directly using the resident *DDCI* interface. By contrast adding such a function using a traditional approach would require a modification of the instrument's firmware, in addition to a modification of the instrument management software on the PC.

4 Symbolic Access

All the functions of the *DDCI* interface feature symbolic access. The symbolic names given to variables and functions in the instrument's firmware are used to access them, instead of hard-coded addresses. This is a very powerful feature of the *DDCI* interface. During development a great deal of time is wasted tracking software bugs that are due to inconsistencies between the addresses at which variables are accessed and the addresses at which they really reside in the instrument's memory. Any rebuild of the instrument's firmware may lead to changes in variables addresses. If the instrument management software on the PC is not modified accordingly, accesses occur at the wrong addresses. Because the instrument management software on the PC - through the *DDCI* interface - accesses variables and functions using their symbolic names, accesses always occur at the correct addresses. This is true even after a rebuild of the instrument's firmware. This powerful feature saves time during development and eliminates a strong source of software bugs. Yet it is flexible enough to provide access to any variable or function in the instrument's firmware.

5 Improved Code Reliability

Another strong advantage of the *DDCI* interface is that it provides, with very little effort, much greater real-time visibility into the operation of the code running on the instrument's CPU. This greater visibility during development directly translates into more reliable embedded code. Because the developer can observe and act on variables with minimal interference on the execution of the code and very little effort, a great deal more is observed during development than what would be the case in a more traditional environment. In addition what is observed is more representative of reality because the *DDCI* interface provides much less interference with the real-time execution of the code than traditional emulation techniques.

6 Analysis with Hardware in the Loop

In many applications that implement complex signal processing algorithms it is necessary to simulate the algorithm to validate its operation. Such simulations have to be designed using specialized simulation software, and are often plagued by the lack of availability of real-life data. Often hardware-specific factors, such as the characteristics of the acquisition chain used by the instrument, or the effect of noise cannot be properly taken into account by the simulation.

The real-time instrumentation provided by the *DDCI* interface makes it possible to bring back the data seen and processed by the instrument in real-life conditions. Because the *DDCI* interface is based on the LabVIEW development environment the extensive LabVIEW signal analysis libraries are available to add powerful analysis and display capabilities on the PC-side. This enables the analysis of the high-level behavior of the signal-processing code with ease, and with real-life conditions and data. Often the simulation step can be bypassed completely, with better results based on real data.

For instance in the *Noise Sentry* sound-level meter/datalogger the response of the microphone, and the effect of the instrument enclosure on this frequency response can be measured precisely. From this measurement an optimal equalizing filter can be designed and implemented in real-time, and the flatness of the equalized response can be verified. This would be impossible to do without access to the signal measured by the microphone itself, taking into account the acoustic interaction with its production enclosure.

As another example, consider the case of the *Vibration Sentry* vibration-level meter/datalogger. Power consumption is affected by the rate of signal observation. Through the *DDCI* interface power consumption data can be acquired easily, in real-life conditions, using a production-level instrument rather than a test bench.

7 Production Testing

Because it can provide low-level access to the instrument, the *DDCI* interface has a special role to play during production testing. Using *DDCI* it is very easy to build automated test benches that thoroughly exercise the individual hardware elements of the instrument quickly, fully and consistently. Defects are easily detected and the defective instruments rejected. In addition, the high-level analysis provided by the *DDCI* interface allows the implementation of very elaborate test and calibration procedures. For instance in the *Noise Sentry* sound-level meter/datalogger, the frequency response of the instrument is precisely measured and corrected using an equalizing filter that is optimized for each individual instrument out of the production line. In contrast most sound level meters are only calibrated at 1 kHz.

8 A New Paradigm for Embedded System Development

Taken together, all the features of the *DDCI* interface offer a new paradigm for the design of embedded systems. The difficulties associated with the development of systems with embedded processing are mostly due to the lack of visibility into the code executing on the embedded system. In contrast the development of code on computer systems is facilitated by the fact that the in-system development and debugging tools offer great real-time visibility into the code running locally, on the computer itself. What the *Development to Deployment Code Instrumentation* architecture offers is simply a way to interact with the code running on the remote embedded platform with the same ease and functionality as if it was part of the computer system hosting the development tools. In short it provides the same comfort of development that all computer-system software developers are used to.