

SPECIFYING AN IVI CLASS FOR DIGITAL TEST INSTRUMENTATION

**Margaret Cadogan
Teradyne, Inc.
Boston, MA 02111**

**Phone (617) 422-3837 Fax (617) 422-3440
Email: maggie.cadogan@teradyne.com**

**Teresa Lopes
Teradyne, Inc.
Boston, MA 02111**

**Phone (617) 422-3377 Fax (617) 422-3100
Email: teresa.lopes@teradyne.com**

Abstract - This paper presents an overview of the work being done by Teradyne in conjunction with the IVI Foundation to specify an IVI class for digital instrumentation. The Interchangeable Virtual Instruments (IVI) Foundation was formed in August of 1997 to define standard specifications for programming common test instrument capabilities.

The paper will present the major architectural aspects of digital test instrumentation, and how those features can be grouped into classes for the purpose of writing an instrument independent driver. Topics discussed will include derivation of capability classes, class extensions, simulation, and range checking. Examples of how the IVI digital class would apply to the Teradyne M9-Series Digital Test Instrument will be included.

Conclusions will summarize the unique attributes of digital test instrumentation, the benefits which can be achieved through standardization, and the tradeoffs associated with utilizing class extensions.

I. INTRODUCTION

The Interchangeable Virtual Instruments foundation was formed in August of 1997, by a group of companies representing instrumentation manufacturers, systems integrators, Test Program Set (TPS) developers, and end users of test equipment. The goal of the IVI Foundation is to reduce the overall cost of test by defining standard instrument driver programming interfaces to common instrument classes. These interfaces will ultimately comprise a set of generic instrument drivers that can be used to control a variety of instruments in each class, regardless of instrument manufacturer.

There are two key benefits to be achieved through the implementation of IVI drivers: protection against instrumentation obsolescence and ease of TPS rehost. By using a generic instrument driver, the TPS is programmed at an instrument independent level of abstraction. In contrast, programming at the driver level today requires the TPS to reference instrument-specific functionality and driver specific syntax. As instrumentation changes, either through replacement or re-host, these tight ties to the instrument restrict the transportability of the TPS software, greatly increasing the cost of re-integrating the TPS.

As an instrument supplier, Teradyne joined the IVI foundation in 1998 to help define the IVI driver class for digital test instrumentation. As an ever-increasing number of test applications require high performance digital test resources, the need for a well-structured class defining digital test capabilities becomes more and more apparent. This paper will describe a proposed class structure for digital instrumentation, which will be refined as Teradyne continues to work with the members of the IVI foundation and the user community.

II. DIGITAL ARCHITECTURE OVERVIEW

The key to defining an IVI class for digital is to define a common language for digital testing and a common architecture to support this language.

A digital test needs to specify four parameters in order to properly generate the stimulus and response required by the UUT. These parameters are patterns, timing, levels, and pattern control.

The **patterns** contain the logic values that are required to test the UUT. The pattern data is combined with the timing information to create the correct stimulus and measure the correct response at the UUT. Each logic state needs to be able to indicate if the logic applies to stimulus or response. Pattern data specifies both directionality (input, output or bi-directional) and data (high, low or tristate (X)). In addition, the pattern data should also include format information, allowing the pattern to represent common sequences of logic values within a single cycle.

The **timing** indicates when the instrument should drive and sense. This data defines the basic cycle time and the time that every edge should occur.

The **levels** assign voltage and current values to the logic values in the pattern data. Levels include input voltage for logic 1 and 0 stimulus data, and output thresholds for logic 1 and 0 response data.

Pattern control includes loops, waits, jumps and other pattern sequencing modifiers. These can be used for initializing UUTs and compressing pattern sequences for highly repetitive sequences.

The next step in defining a digital class is to look at the types of instruments that are available and define a common architecture.

An **I/O port** is the simplest type of digital instrument. I/O ports are a simple way to control a small number of digital channels, but their lack of speed, programmable timing and programmable logic levels limits their ability to test the UUT at its specifications.

Digital Word Generators or DWGs add a state machine and local memory to make a more general-purpose instrument. Patterns are cached in local memory on the instrument. This gives some control over timing, since the memory can be loaded at an arbitrary speed, and then applied from memory at a fixed clock rate. The number and type of I/O channels is configurable and may have several types of fixed logic levels, and even programmable logic levels, to choose from.

Performance digital instruments provide highly flexible channels with individually selectable logic families and timing. Performance digital instruments are most commonly used when it is necessary to emulate the interface between the UUT and its final system as closely as possible. Timing is implemented differently than that of DWGs by adding specialized circuitry to control the timing of signals, leaving the pattern memory to supply the digital test data without directly controlling timing. This allows the performance digital instrument to program in cycles, rather than by events. This saves

memory and more closely matches the UUT's operation.

The performance digital instrument will be used as the basis for the common architecture, since both digital word generators and I/O ports can map into the characteristics of the performance digital instrument. This architecture has several defining characteristics, mostly related to timing and flexibility. The major components of the performance digital instrument architecture are shown in Figure 1 below.

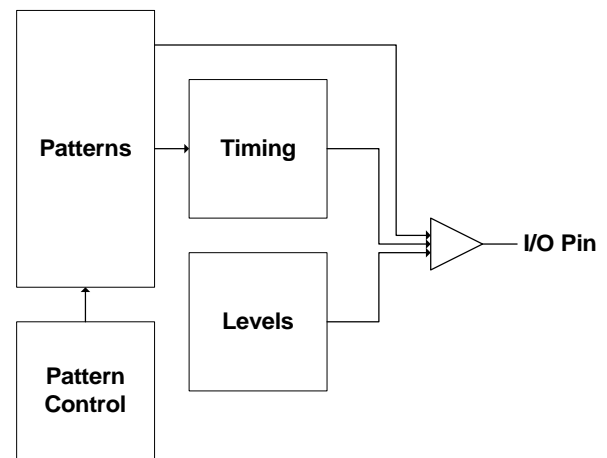


Figure 1: Performance Digital Instrument Architecture

III. AN IVI CLASS FOR DIGITAL

Capability Classes

Work is currently underway in the IVI foundation to define a class or classes for digital instruments. A central aspect of the IVI driver architecture is the ability to expand the functionality of the driver by adding additional capability classes. In the proposed IVI Digital model below, performance digital will define the envelope of capabilities for the IVI digital class. This class will build upon simple digital as its core, adding capability classes incrementally, to achieve a feature rich performance digital driver class while allowing TPSs with less demanding requirements to build upon the core of the driver. By adding the additional functions in a new capability class, the base driver remains intact, and existing code is not impacted. As a result, there is a path for growth of the driver as technology evolves, allowing programmers take advantage of new features in instrumentation without impacting existing TPSs.

In order to define a class for digital, the capabilities of the base class first need to be defined. The base capabilities should be those that are available in a

majority of the digital instruments. The basis of the proposed digital instrument class will be the simple I/O port. Capability classes will be added until the full feature set of a performance digital instrument is defined. This will define a broad spectrum of digital capabilities into which a majority of the digital instruments can be placed.

The capabilities of the base class can be divided into the four parameters discussed earlier: patterns, timing, levels, and pattern control. A block diagram showing how the capabilities of a digital instrument could be grouped is shown in figure 2 below.

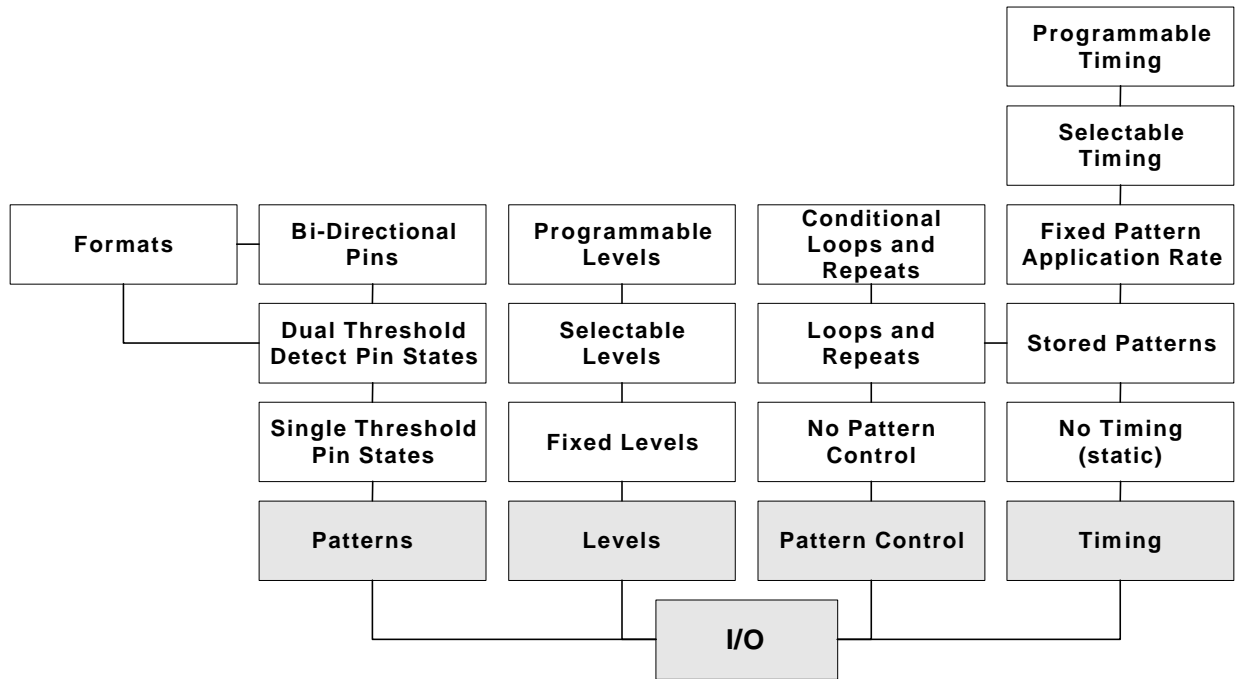


Figure 2: Digital Instrument Capabilities

The proposed class structure starts with a class for a simple digital instrument. The simple digital instrument will be based on the following capabilities: single threshold pin states, fixed levels, no instrument based pattern control, and no timing programmability. Capability classes can then be added until the functionality of a performance digital instrument is achieved, including formatted bi-directional pins with dual threshold pin states, programmable levels, pattern control constructs for conditional looping and repeating, and programmable timing. As shown in figure 2, each

group of capabilities can be traversed independently to describe digital instruments with varying functionality. A brief description of each of the performance parameters is provided in table 1.

Additionally, the IVI architecture is designed to allow for an instrument specific driver, which fall outside the class driver. This enables test programmers to make calls to instrument specific functions to take advantage of special features.

Performance Parameter	Description
Single Threshold Pin States	Ability to drive either high or low and detect above or below a single threshold. A pin state describes the action a pin is performing during a pattern.
Dual Threshold Pin States	Ability to detect using two thresholds: above the high threshold, below the low threshold, or between the two.
Bi-directional Pins	Ability to drive and detect on a single pin, in some cases at the same time.
Formats	Ability to specify a return format for the pin. Formats allow a pin to perform multiple actions on a single pattern.
Fixed Levels	Input and output levels are assigned to a fixed set of voltages.
Selectable Levels	Ability to select from predefined sets of voltages and currents.
Programmable Levels	Ability to specify drive and detect voltage and current levels.
Loops and Repeats	Ability to repeat a single pattern or a group of patterns.
Conditional Loops and Repeats	Ability to conditionally repeat a single pattern or a group of patterns.
No Timing (static)	Pattern application rate is non-deterministic, and varies by CPU speed.
Stored Patterns	Patterns are loaded and stored on the instrument.
Fixed Pattern Application Rate	Patterns are applied from instrument memory at a fixed deterministic rate.
Selectable Timing	Ability to select from predefined pattern application rates.
Programmable Timing	Ability to specify pattern rate and edge placement.

Table 1. Digital performance parameters

Digital Bus Emulation

Although a performance digital instrument coupled with applications software can perform as a bus emulator for many common communications protocols, bus emulators are not currently being considered to be part of the same IVI class as performance digital. While performance digital can be programmed to function as a bus emulator, dedicated bus emulator instruments cannot function as general purpose digital instruments. There is ongoing work within the IVI foundation to define an IVI class for bus emulators. Once that class is defined, it is fairly straightforward to create applications specific software which programs a performance digital instrument to conform to the appropriate bus protocol.

Range checking

IVI drivers provide software portability by defining a generic driver that provides a common syntax for controlling instrument functionality, but that only solves half of the problem. In order to replace instrumentation or re-host to a new platform, an analysis must be performed to determine if the capabilities of the new instrument will meet the requirements of the TPS. The IVI architecture addresses this aspect of the problem through range checking.

By definition, there will be instruments with varying performance characteristics that belong to the same IVI class. Range checking provides a means by which to

gauge transportability as defined by the actual performance characteristics of the hardware. Additionally, range checking can be used in software simulation mode, allowing for off-line TPS software generation and validation.

The two primary parameters that define performance for digital instruments are timing and levels. The proposed IVI digital driver will implement range checking for these performance features, as these are the characteristics that most influence TPS software transportability for digital test programs. Range checking for timing will include accuracy, frequently referred to as skew, resolution, and flexibility. Range checking for levels will include voltage range, current source and sink, voltage programming resolution, and accuracy. Consideration will also be given to range checking for additional features, including pin formats, pattern memory depth, and pin direction control (drive, detect, or bi-directional).

IV. TERADYNE M9-SERIES DIGITAL EXAMPLE

IVI drivers expose the functionality that is common within an instrument class, however utilization of IVI drivers does not preclude access to the unique features of a given instrument. The IVI architecture provides for an IVI class driver, as well as an IVI specific driver, allowing calls to IVI class and IVI specific drivers to co-exist within a TPS. The example below illustrates how a TPS might program the drive and detect levels of a digital test instrument using a *VXIplug&play* driver, then how it might be done using the IVI structure.

The function call below sets up the driver/detector levels on the Teradyne M9-Series digital test instrument using the *VXIplug&play* driver. It sets the voltage levels $V_{IH} = 3.3V$, $V_{IL} = 0.0V$, $V_{OH} = 2.0V$, $V_{OL} = 0.2V$, sets the programmable load values $V_{COM} = 1.5V$, $I_{OH} = -50.0e-6$ A, $I_{OL} = 50.0e-6$ A, and sets the slew rate to medium.

```
terM9_setLevelSet(vi, 0, TERM9_SCOPE_SYSTEM,  
3.3, 0.0, 2.0, 0.2, 1.5, -50.0e-6, 50.0e-6,  
TERM9_SLEWRATE_MEDIUM);
```

Setting levels using an IVI digital class driver might look something like this:

Set the drive and detect voltage levels, which are part of the core class:

```
IviDigital_LevelSetConfigure(vi, 0,  
3.3, 0.0, 2.0, 0.2);
```

Set the programmable load levels, which are part of a class extension:

```
IviDigital_LoadConfigure(vi, 0,  
1.5, -50.0e-6, 50.0e-6);
```

Set the slew rate, which is unique to this instrument, using the IVI specific driver:

```
terM9_DigitalLevelSetSlewRate(vi,0,  
TERM9_SLEWRATE_MEDIUM);
```

As shown in the example above, the TPS developer always has the option of going outside of the IVI class and making calls to the instrument specific driver, however there are tradeoffs to be considered in this approach. Restricting the TPS to use only the functions of the IVI driver may limit the available functionality to the least common denominator of the instruments within that class. However once the test program moves outside of the IVI domain, there is risk that the instrument specific code will not be transportable to

other instruments in the future. Ultimately, the scope of any future re-host task will be bounded the number of instrument specific functions employed, therefore the value of the additional functionality needs to be weighed against this risk to transportability. Alternatively, an additional instrument feature, such as a programmable slew rate, could be implemented as a class extension. As other instruments add this feature over time, there will be a well-defined class extension, thereby protecting the test program software in the event of a future re-host.

V. CONCLUSIONS

As performance digital test equipment moves from the realm of proprietary architectures into standard instrumentation, the need for employing industry standards in both the hardware and software architectures grows. Only through the use of standards can any level of instrument interchangeability be achieved. The VXI architecture has provided a well-structured standard for instrumentation, allowing an evolutionary path for previous generation digital test equipment to move forward into a standard hardware architecture; and the *VXIplug&play* standard has provided a level of interchangeability at the Application Development Environment (ADE). The IVI standard can now build on the successes of VXI, by bringing a higher level of standardization to the TPS software that controls these instruments.

The benefits to be achieved through the use of these standards can be seen both in the Automated Test Equipment (ATE) itself as well as in the TPS software. Instrumentation interchangeability provides a path for graceful replacement of obsolete instrumentation, allowing fielded systems to have a longer life cycle. From a TPS perspective, making use of the IVI driver architecture provides a common software driver layer for TPSs, independent of the target ATE. Programming at this higher level of abstraction protects the test program from changes that may take place at the instrument level, whether through replacement of an instrument in the current ATE, or re-host of the TPS to a new target tester.

These benefits result in a lower overall cost of ownership for the ATE and TPSs at all stages of the life cycle, from acquisition through test development, deployment, and ultimately ATE upgrade or TPS re-host.