# High Speed Digital Reads and Simultaneous Digital Writes With the Keithley KPCI-3100 Series and DriverLINX®

**by**
**Matt Holtz**
**Keithley Instruments, Inc.**

## Introduction

This application note discusses high-speed digital reads and simultaneous digital writes on Keithley's KPCI-3100 Series of data acquisition boards using DriverLINX. The KPCI-3100 Series boards include the KPCI-3101, 3102, 3103, 3104, 3110, and 3116. The features described in this document are:

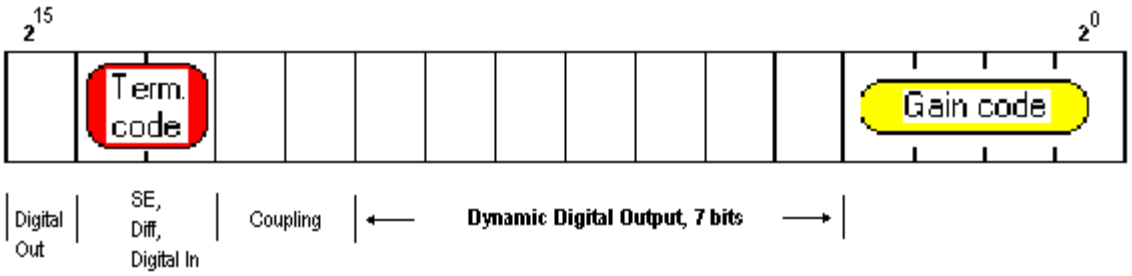
- High speed digital reads using the analog input subsystem.
- Writing digital data while reading the analog input subsystem.

These features are implemented within the analog input subsystem in DriverLINX and through the boards' ability to scan analog input (AI) and digital I/O in the same channel-gain queue. This allows the user to read data from several AI channels and 16-bits of digital input synchronously. The reading of the digital inputs are paced with the same timer as the analog-to-digital (A/D) converter. Also, because the digital lines are written or read as an entry in the channel-gain queue, there is no time lost in switching between the analog and digital subsystems, nor is extra programming required. If the user is reading only the 16-bits of digital input, then the hardware can sample at speeds of 3MHz. The use of high speed digital writes are constrained to the speed of the A/D converter.

## Background

To use this document effectively, it is first necessary to understand the concept behind the channel-gain queue. The channel-gain queue is memory on the board that stores channels and gains in a given order. For instance, the user could specify channel 0/gain -1, channel 1/gain -2, and channel 0/gain -1 as three entries in the channel-gain queue. When the start data acquisition command is given to DriverLINX, the A/D converter will read those channels at the specified gains in the specified order and return the data to the software for it to process.

A user can implement the high-speed digital reads as an entry on the board's analog input channel-gain queue. In this case, ask for channel 0 but specify a special gain code to indicate the 16-bits of digital data are desired and not analong input channel 0. The gain property of the DriverLINX service request is 16bits wide, however, only a few of the lower order bits are required to uniquely identify the avaiable analog input ranges of the board: gains of 1, 2, 4 and 8 combined with Unipolar or Bipolar yield only 8 possible combinations. The upper bits of the gain property are used to select special features of the board. If the requested gain for channel 0 is set to a value corresponding to bit 13 is set to a one (8192), this signifies that the 16-bits of digital data are desired instead of analog input channel 0.




The high-speed reads have several important properties worth noting:

- The digital reads performed are neither polled nor read with interrupts. Rather, they are directly transferred over the PCI bus with bus mastering, just like they would be in the digital input subsystem, allowing for the high-speed streaming. There is a tremendous amount of data at 3MHz, so more buffers must be allocated.
- These digital reads are deterministic. These values are read at a known time-in the best case, 333ns from each

other (3MHz). This is a paced, clocked, and buffered acquisition.

The real value behind digital reads from within the analog input subsystem is that no switching occurs between subsystems. This allows analog and digital data to be read without losing any time switching subsystems. Therefore, the user could, for example, read analog channel 0, analog channel 1, digital data, analog channel 2, analog channel 3, then repeat. The read is implemented directly on the channel-gain queue to allow this.

It is possible to implement the simultaneous digital writes with analog reads in a similar fashion. First, select the channel and gain as usual, then modify the gain by merging onto it the digital data, along with a hardware-specific value. It is important to remember that this data is synchronous with the analog input read, so it is being written at whatever speed the A/D converter is sampling. If the analog input subsystem is set to 250KHz, then the digital data will be written at 250KHz.

## Application

The KPCI-3100 Series boards all have multiple banks of digital channels. The KPCI-3101/2/3/4 have two banks of 8 bits and one bank of 7 bits (channels A, B, and C respectively). The KPCI-3110/16 have two banks of 8 bits and one bank of 2 bits (channels A and B, and the dynamic digital outputs, respectively). On the KPCI-3101/2/3/4, channel C may be operated either as normal digital bits or as dynamic digital output bits (dynamic digital output bits is the name for the bits that are modified when doing analog reads with digital writes). The 2 bits of the KPCI-3110/16 can only be operated as dynamic digital outputs.

## Walkthrough

Keithley developed example applications to demonstrate the principles outlined in this document. One type performs 3MHz digital reads from within the analog input subsystem. The other performs simultaneous analog reads and digital writes using a channel-gain queue. These programs can be downloaded from our web site. Only the DLL or ActiveX API of DriverLINX provide access to these features of the KPCI3100 Series boards. Therefore, this feature of the KPCI3100 driver is not available from the TestPoint or LabVIEW API due to the additional software layers.

## Application 1. Digital Read Using Analog Input

To implement a digital read using analog input at 3MHz, follow this procedure:

1. Setup a channel-gain queue with one entry.
2. Select channel 0.
3. Set the gain code property to CHAN_SEDIFF_DIG (8192).
4. Take the data received and treat it as 16 digital values on a 16-bit board or 12 digital values on a 12-bit board, not as a voltage.

The code for implementing this procedure is listed below:

```
' For boards having unique, product-specific attributes
Const CHAN_OEM_SHIFT   As Integer = (-2) ^ 15
Const CHAN_OEM_MASK    As Integer = (1 * CHAN_OEM_SHIFT)
' use std attribs
Const CHAN_OEM_DEFAULT As Integer = (0 * CHAN_OEM_SHIFT)
' use OEM attribs
Const CHAN_OEM_FLAG    As Integer = (1 * CHAN_OEM_SHIFT)

' For boards supporting programmable single-ended vs. differential inputs
Const CHAN_SEDIFF_SHIFT   As Integer = 2 ^ 13
Const CHAN_SEDIFF_MASK    As Integer = (3 * CHAN_SEDIFF_SHIFT)
' use static config
Const CHAN_SEDIFF_DEFAULT As Integer = (0 * CHAN_SEDIFF_SHIFT)
' digital input chan
Const CHAN_SEDIFF_DIG     As Integer = (1 * CHAN_SEDIFF_SHIFT)
```

```
. . .

    DriverLINXSR1.Sel_chan_N = 1
    DriverLINXSR1.Sel_chan_format = DL_tNATIVE
    DriverLINXSR1.Sel_chan_list(0) = 0
    DriverLINXSR1.Sel_chan_gainCodeList(0) = CHAN_SEDIFF_DIG
```

The code sets individual properties of the service request.

- The first property tells DriverLINX that only one channel is needed in the channel-gain queue (Step 1 above).
- The second property, Sel_chan_list, tells DriverLINX how to store the data in the buffer. DL_tNATIVE means store it in the format dictated by the hardware.
- The third property selects the channel for the first entry in the channel-gain queue. In this case, that value should be 0 (Step 2).
- The fourth property sets up the digital read (Step 3). Elsewhere in the code this digital value is processed bit by bit (Step 4).

It may seem confusing that CHAN_OEM_FLAG is set to (-2) ^ 15 instead of 2 ^ 15, which would seem to make more sense. The answer lies with VisualBasic. VisualBasic integers are always signed, so bit 15 is the sign bit, and the value the user ORs must be -32768, not 32768 (which will yield an overflow error).

Note that for a real application, a user would probably want to acquire analog data in addition to digital data. This application was designed to show the simplest way to implement the analog input/digital input feature, so it performs no analog conversions. The example could easily be modified by adding more channels and gains to the list.

### Application 2. Simultaneous Digital Write While Reading Analog Data

To implement a simultaneous digital write while reading analog data, follow this procedure:

1. Use any given entry on a channel-gain queue.
2. Take the digital data and mask out any unused bits by bitwise ANDing the data with 127 (for KPCI-3101/2/3/4) or 3 (for KPCI-3110/16). The constants DATA_MASK_3101 and DATA_MASK_3110 are defined for this purpose.
    - The KPCI-3101/2/3/4 has 7 dynamic digital bits that can be set by this method, so 0 to 127 are valid values.
    - The KPCI-3110/16 has 2 dynamic digital bits, so valid values are 0 to 3.
3. On the selected entry (mentioned in Step 1), bitwise OR the data onto the gain after shifting it left 4 bits (that is, multiply it by 2^4). This value is defined as DIGDATA_MULTIPLIER in code. Then set bit 15 by ORing (-2) ^ 15. This value is defined as CHAN_OEM_FLAG in code.
4. Whenever the KPCI board comes across this entry (from Step 3) in the channel-gain queue, the dynamic digital output bits will be set accordingly.

The following code implements this procedure:

```
'Choose 3110/16 or 3101/2/3/4.
Const HAVE_3110_OR_3116 As Boolean = True
'3110/16 have two Dyn Dig outputs.
Const DATA_MASK_3110 As Integer = 3
'3101/2/3/4 have 7 Dyn Dig outputs.
Const DATA_MASK_3101 As Integer = 127

' For boards having unique, product-specific attributes
Const CHAN_OEM_SHIFT   As Integer = (-2) ^ 15
Const CHAN_OEM_MASK    As Integer = (1 * CHAN_OEM_SHIFT)
' use std attribs
Const CHAN_OEM_DEFAULT As Integer = (0 * CHAN_OEM_SHIFT)
' use OEM attribs
Const CHAN_OEM_FLAG    As Integer = (1 * CHAN_OEM_SHIFT)

' For boards supporting programmable single-ended vs. differential inputs
Const CHAN_SEDIFF_SHIFT   As Integer = 2 ^ 13
Const CHAN_SEDIFF_MASK    As Integer = (3 * CHAN_SEDIFF_SHIFT)
' use static config
```

```
Const CHAN_SEDIFF_DEFAULT As Integer = (0 * CHAN_SEDIFF_SHIFT)
' digital input chan
Const CHAN_SEDIFF_DIG     As Integer = (1 * CHAN_SEDIFF_SHIFT)
'When merging data, multiply by this
Const DIGDATA_MULTIPLIER As Integer = 2 ^ 4
. . .

DriverLINXSR1.Sel_chan_N = NumberOfChannels
DriverLINXSR1.Sel_chan_format = DL_tNATIVE

For i = 0 To NumberOfChannels - 1
     DriverLINXSR1.Sel_chan_list(i) = LogicalChannels(i)
     DriverLINXSR1.Sel_chan_gainCodeList(i) = _
     DriverLINXSR1.DLGain2Code(ChannelGains(i)) _
                                      Or (DigitalData(i) * DIGDATA_MULTIPLIER) _
                                      Or CHAN_OEM_FLAG
Next i
```

In this example (as in the previous example), the code sets the individual properties of the service request.

- The first property set is Sel_chan_N or the number of entries on the channel-gain list.
- The second property, Sel_chan_list, tells DriverLINX how to store the data in the buffer. DL_tNATIVE means store it in the format dictated by the hardware.
- The third property has each of its values (it is an array) set to the current channel list.
- The fourth sets the gains of those respective channels, sets bit 15 (CHAN_OEM_FLAG), and sets the data (DigitalData (I) * DIGDATA_MULTIPLIER) (Step 3). It is important to note that the DigitalData() array already has masked data. That is, it has already been ANDed with a data mask elsewhere in the code (Step 2).

There is one more thing to note about writing dynamic digital outputs. Whenever any one of the gains in the channel-gain queue has CHAN_OEM_FLAG set, DriverLINX will assume that every gain in the channel-gain queue has had CHAN_OEM_FLAG set. This means that the user cannot simply not set the CHAN_OEM_FLAG in an entry and expect the digital output not to change. Rather, the output would change. It would change to the initial value of the gaincode property: 0. This means that the user must program every entry with the digital value desired. There is no "continue last value" feature.